1 Anderson, Rob

1.1 Appendix: MLV and ARV Propulsion

Author: Rob Anderson

1.1.1 Sizing Algorithm

A sizing algorithm exist which sizes the propulsion systems for both the MLV and the ARV. The algorithm began as a pen and paper method to estimate propellant masses for the descent and ascent stages of the MLV, and gradually evolved into a full Matlab script (lander_sizing). The script now includes all of the MLV's propulsion systems, and calculates a number of system parameters. Because the calculated propulsion numbers are highly dependent on the vehicle payloads, we translated the Matlab code into an Excel spreadsheet (lander_sizing.xls), which we then pasted into the team's Parameter Tracking Spreadsheet (numbers.xls) created by Jeri Lynn Metzger. In this way, all propulsion calculations are automatically updated as soon as any other vehicle component changes. This spreadsheet is also modified to perform the same task for the ARV (arv_sizing.xls).

1.1.1.1 MLV

Without knowing anything else about the mass or shape of the MLV, the sizing code was created. By doing this, we give everyone involved a better idea of how the mass adds up when certain velocity increments, specific impulses, propulsive mass fractions are selected. We begin by explaining the algorithm that was created to complete this task.

1.1.1.2 Theory

We base the sizing of the MLV propulsion systems on an algorithm derived from the ideal rocket equation,

$$\Delta v = I_{sp} g_0 \ln(MR)$$

Figure 1-1

where the mass ratio, *MR*, is given by

$$MR = \frac{m_{propellant} + m_{payload} + m_{inert}}{m_{payload} + m_{inert}}$$

Figure 1-2

To initialize the sizing, we estimate a velocity increment the DPS is capable of delivering. We then also assume a specific impulse by selecting a preliminary engine. With these values, we can solve Figure 1-1 for the mass ratio *MR*. The equation for mass ratio is given in Figure 1-2, but we don't have enough variables to solve for any of the unknown masses. However, if we assume a certain propulsive mass fraction, based on the preliminary choice of propulsion system, we can solve for the unknown mass quantities. The equation for the propulsive mass fraction is given in Figure 1-3.

$$\lambda = \frac{m_{propellant}}{m_{propellant} + m_{inert}}$$

Figure 1-3

If we this equation for the inert mass, m_{in}, we have

$$\lambda(m_p + m_{in}) = m_p$$
$$\lambda m_{in} = m_p - \lambda m_p$$
$$m_{in} = m_p\left(\frac{1-\lambda}{\lambda}\right)$$

Figure 1-4

Substituting this equation into Figure 1-2 and solving for the propellant mass, we have,

$$m_p = \frac{\lambda m_{pl}(MR-1)}{1-MR(1-\lambda)}$$

Figure 1-5

Thus, if we assume a payload mass, or the mass of the MLV ascent stage, we can calculate the propellant mass required for the estimated Δv, I_{sp}, and λ. This then enables us to determine if the estimates and assumptions are realistic, and points us in the direction we should go for the next iteration. As stated above, the design constantly evolves, so this algorithm was written into a Matlab code which could be ran quite quickly with new numbers to observe the effect of changing various parameters. Eventually this code came to include all of the propulsion systems of the MLV, and also sized the propellant tanks. The code is named lander_sizing.m, and is found and described in Section 1.1.1.5.2.

1.1.1.3 Staging

The first thing we decide when sizing the MLV, is whether is will be a 1-, 2-, or 3-stage vehicle. To accomplish this, we use the above sizing method. However, for the 2- and 3-stage vehicles we split the velocity increment into portions, and assign a certain portion of the Δv to each stage. Beginning with the uppermost stage, we calculate propellant and inert masses using the determined Δv portion and the estimated payload mass, specific impulse, and propulsive mass fraction. We then add the masses calculated masses to the payload mass, and this becomes the payload for the next-lower stage.

By performing these calculations over a range of values for λ, we begin to get an idea of what staging scheme we should be using. A Matlab script was written to do just this, named staging.m. The results of staging.m are seen in Figure 1-6 and Figure 1-7. In each plot, the one-stage vehicle represents one stage for both the descent and the ascent. The two-stage vehicle represents one stage for the descent, and one stage for the ascent. The three stage vehicle represents one stage for the descent, and two stages for the ascent. It should be noted that as the number of stages increases, the propulsive mass fraction will naturally decrease. This is due to the fact that the amount of structural mass required for the vehicle will increase, thus increasing the inert mass and lowering λ. In other words, the propulsive mass fraction achieved with the three-stage vehicle will not be as high as that achieved by a one-stage vehicle. A two-stage vehicle will fall somewhere in between. This must be taken into account when comparing the plots or the results will be misconstrued. The complexity, and thus risk, must also be taken into account when selecting a staging scheme. The more parts that are involved, the more risk there is that the vehicle will fail. Due to the fact that the curves for the two-stage and three-stage are quite close in the region between lambda values of 0.80 and 0.87, we chose the two-stage vehicle as the preferred choice for the MLV. Safety is paramount in this mission, and launching a dual-stage vehicle from the surface seems out of the question. Thus, one stage is used for the descent, and one stage is used for the ascent.

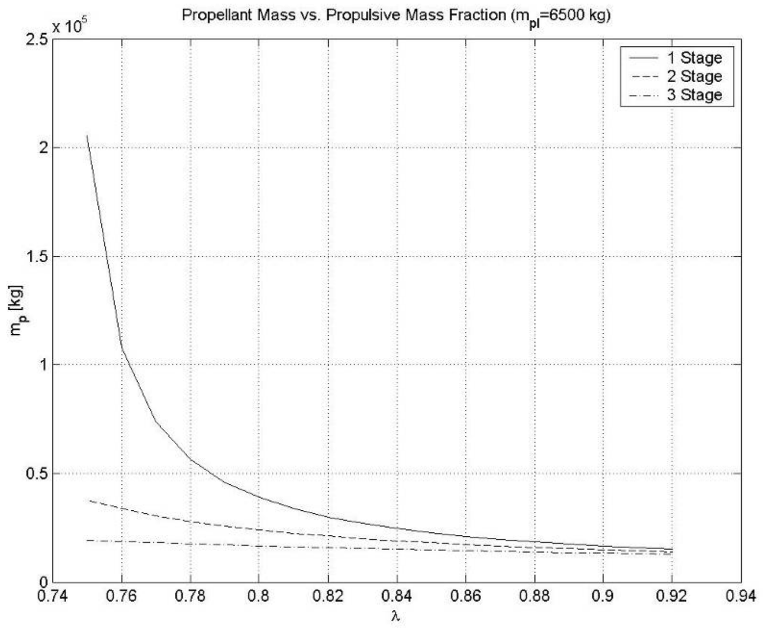


Figure 1-6: Propellant Mass vs. Propulsive Mass Fraction

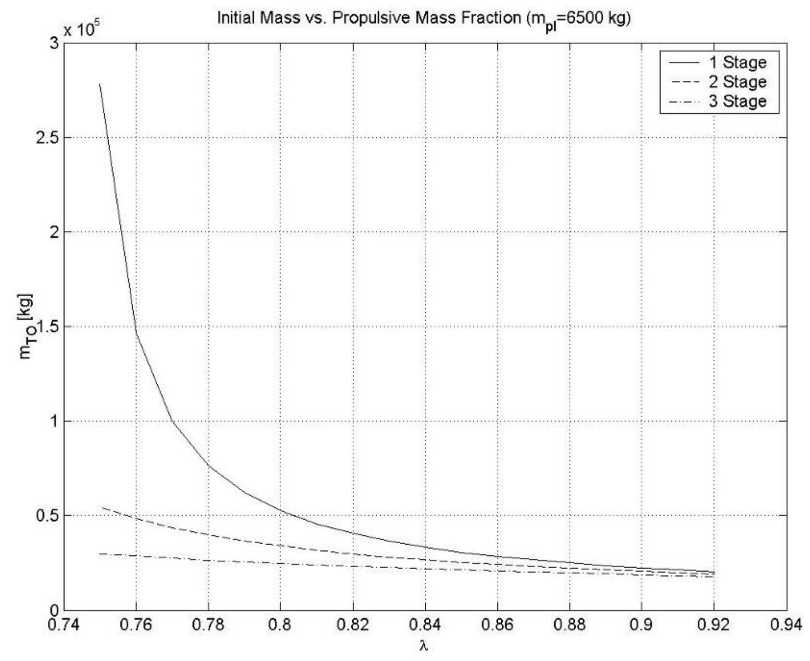


Figure 1-7: Total Mass vs. Propulsive Mass Fraction

1.1.1.4 Sizing and Engine Selection

With a staging scheme selected, a preliminary engine is selected and the vehicle is sized using the sizing code (see rear of this appendix for script). As the vehicle evolved, many engines were considered for the descent and ascent propulsion roles. Initially, all of the engines considered used storable bipropellant combinations. With such a long voyage to Mars, and a long stay on the Martian surface, it seemed that storable propellants were the only alternative. However, with the introduction of Zero Boil-Off (ZBO) technology, it was possible to consider engine which use high performing cryogenic propellant combinations. Because high performance and high thrust are paramount in this mission, the RL10B-2 was selected for the DPS and APS.

1.1.1.5 Matlab Scripts

The following sections give a brief description of the Matlab codes used in sizing propulsion systems.

1.1.1.5.1 staging.m

The code staging.m operates as follows. The inputs are the required velocity increments for each stage, the specific impulse of each stage, the payload mass, the gravitational constant, and a range of propulsive mass fractions. The code calculates the inert mass, propellant mass, and total mass of each stage for each vehicle. The results are then plotted as mass vs. propulsive mass fraction for comparison.

```
% staging.m
% Rob Anderson
% AAE 450

% Compares propellant masses required for 1-stage, 2-stage, and 3-stage vehicle for
% several propulsive mass fractions

clear all
close all
clc

del_v1=1000;            % velocity increment of stage 1 [m/s]
del_v2=5131;            %    "         "     "    "   2 [m/s]
del_v3=0;               %    "         "     "    "   3 [m/s]
Isp_1=465;              % specific impulse of first stage [s]
Isp_2=465;              %     "       "    "  second stage [s]
Isp_3=465;              %     "       "    "  third stage [s]
g0=9.81;                % gravitational constant [m/s^2]
lambda=[.75:.01:.92];   % propulsive mass fraction
m_pl=4000;              % payload mass [kg]



% Single Stage
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

MR=exp((del_v1+del_v2+del_v3)/(Isp_1*g0));      % mass ratio
mp=lambda.*m_pl.*(MR-1)./(1-MR.*(1-lambda));    % propellant mass [kg]
m_in=mp.*(1-lambda)./lambda;                    % inert mass [kg]
m_total=m_pl+m_in+mp;                           % total mass [kg]
m_total_1=m_total';

% Dual Stage
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Stage 2
MR_2=exp((del_v2+del_v3)/(Isp_2*g0));
mp2=lambda.*m_pl.*(MR_2-1)./(1-MR_2.*(1-lambda));
m_in2=mp2.*(1-lambda)./lambda;
m_stage2=m_pl+m_in2+mp2;

% Stage 1
MR_1=exp(del_v1/(Isp_1*g0));
mp1=lambda.*m_stage2.*(MR_1-1)./(1-MR_1.*(1-lambda));
m_in1=mp1.*(1-lambda)./lambda;
m_stage1_2=m_stage2+m_in1+mp1;

inert_stage2_2=m_in2'
mp_stage2_2=mp2'
inert_stage1_2=m_in1'
mp_stage1_2=mp1'
mp_total_2=(mp1+mp2)'
m_total_2=m_stage1_2'

% Triple
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Stage 3
MR_3=exp(del_v2/(2*Isp_3*g0));
mp3=lambda.*m_pl.*(MR_3-1)./(1-MR_3.*(1-lambda));
m_in3=mp3.*(1-lambda)./lambda;
m_stage3=m_pl+m_in3+mp3;

% Stage 2
MR_2=exp(del_v2/(2*Isp_2*g0));
mp2=lambda.*m_stage3.*(MR_2-1)./(1-MR_2.*(1-lambda));
m_in2=mp2.*(1-lambda)./lambda;
m_stage2=m_stage3+m_in2+mp2;

% Stage 1
MR_1=exp(del_v1/(Isp_1*g0));
mp1=lambda.*m_stage2.*(MR_1-1)./(1-MR_1.*(1-lambda));
m_in1=mp1.*(1-lambda)./lambda;
```

```
m_stage1_3=m_stage2+m_in1+mp1;

inert_stage3_3=m_in3'
mp_stage3_3=mp3'
total_stage3_3=m_stage3'
inert_stage2_3=m_in2'
mp_stage2_3=mp2'
total_stage2=m_stage2'
inert_stage1_3=m_in1'
mp_stage1=mp1'
mp_total_3=(mp1+mp2+mp3)'
m_total_3=m_stage1_3'

% Data Output
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


figure(1)
plot(lambda,mp,'-k',lambda,mp_total_2,'--k',lambda,mp_total_3,'-.k')
title('Propellant Mass vs. Propulsive Mass Fraction (m_p_l=6500 kg)')
xlabel('\lambda')
ylabel('m_p [kg]')
legend('1 Stage','2 Stage','3 Stage')
grid on

figure(2)
plot(lambda,m_total_1,'-k',lambda,m_total_2,'--k',lambda,m_total_3,'-.k')
title('Initial Mass vs. Propulsive Mass Fraction (m_p_l=6500 kg)')
xlabel('\lambda')
ylabel('m_T_O [kg]')
legend('1 Stage','2 Stage','3 Stage')
grid on
```

1.1.1.5.2 lander_sizing.m

This code is used to size the propulsion system for the MLV, including the RCS. There are several inputs. They include the velocity increment required by each system (RCS, DPS, APS), the specific impulse of each system, the propulsive mass fraction of each system, the crew module mass, an estimate of the total vehicle mass, the mixture ratio of each propulsion system, and the density of each propellant used. The code calculates the inert mass and propellant masses required by each system. The script also determines the tank volume required by each propellant. The output can be altered by the user. This Matlab script was translated into an excel file for use in the Parameter Tracking Spreadsheet (numbers.xls) created by Jeri Lynn Metzger. Another section was added to the spreadsheet to calculate numbers for the additional MLV RCS system, RCS-1, which came about late in the project. The new section was then copied and modified to perform the calculations for the ARV RCS system. The calculations for these systems are not included in staging.m.

```
% lander_sizing.m
% Rob Anderson
% AAE 450

% Purpose:  Performs basic sizing of Mars lander given certain
%           assumptions and user-inputed engine variables.

% Note:  All mass values in [kg] and volumes in [kg/m^3]

clear all
close all
clc

del_v=30;               % RCS velocity increment [m/s]
del_v1=1000;            % velocity increment of descent stage [m/s]
del_v2=4800;            %    "         "     "  ascent stage [m/s]
Isp_rcs=280;            % RCS specific impulse [m/s]
Isp_1=465;              % specific impulse of descent stage [s]
Isp_2=465;              %     "       "    "  ascent stage [s]
g0=9.81;                % gravitational constant [m/s^2]
lambda_rcs=.4;          % RCS propulsive mass fraction
lambda_1=.80;           % descent propulsive mass fraction
lambda_2=.87;           % ascent propulsive mass fraction
m_module=3250;          % crew module mass [kg]
m_est=20100;            % estimate of vehicle mass [kg]
rho_rcs_ox=1440;        % RCS oxidizer density [kg/m^3]
rho_rcs_f=878;          % RCS fuel density [kg/m^3]
O_F_rcs=1.64;           % RCS oxidizer to fuel ratio
rho_me_ox=1142;         % ME oxidizer density [kg/m^3]
rho_me_f=71;            % ME fuel density [kg/m^3]
O_F_me=5.5;             % ME oxidizer to fuel ratio


% RCS Sizing
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

MR_rcs=exp(del_v/(Isp_rcs*g0));                  % mass ratio
mp_rcs=lambda_rcs.*m_est.*(MR_rcs-1)./(1-MR_rcs.*(1-lambda_rcs));
m_in_rcs=mp_rcs.*(1-lambda_rcs)./lambda_rcs;      % inert mass
m_pl=m_module+m_in_rcs+mp_rcs;     % payload for descent stage

% Mass & Volume Calculations
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
% Ascent Stage
MR_2=exp(del_v2/(Isp_2*g0));                        % mass ratio
mp2=lambda_2.*m_pl.*(MR_2-1)./(1-MR_2.*(1-lambda_2)); % prop. mass
m_in2=mp2.*(1-lambda_2)./lambda_2;                      % inert mass
m_stage2=m_pl+m_in2+mp2;                            % stage mass
m_rcs_ox=mp_rcs.*(O_F_rcs/(O_F_rcs+1));           % rcs ox. mass
m_rcs_f=mp_rcs/(1+O_F_rcs);                      % rcs fuel mass
V_rcs_ox=m_rcs_ox/rho_rcs_ox;              % rcs ox. tank volume
V_rcs_f=m_rcs_f/rho_rcs_f;               % rcs fuel tank volume
m_me2_ox=mp2.*(O_F_me/(O_F_me+1));       % main engine ox. mass
m_me2_f=mp2/(1+O_F_me);                % main engine fuel mass
V_me2_ox=m_me2_ox/rho_me_ox;     % main engine ox. tank volume
V_me2_f=m_me2_f/rho_me_f;       % main engine fuel tank volume

% Descent Stage
MR_1=exp(del_v1/(Isp_1*g0));
mp1=lambda_1.*m_stage2.*(MR_1-1)./(1-MR_1.*(1-lambda_1));
m_in1=mp1.*(1-lambda_1)./lambda_1;
m_stage1=m_stage2+m_in1+mp1;
m_me1_ox=mp1.*(O_F_me/(O_F_me+1));
m_me1_f=mp1/(1+O_F_me);
V_me1_ox=m_me1_ox/rho_me_ox;
V_me1_f=m_me1_f/rho_me_f;

% Data Output
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

mp_rcs                                           % RCS prop. mass
m_in_rcs                                         % RCS inert mass
m_pl                                               % payload mass
V_rcs_ox                                    % RCS ox. tank volume
V_rcs_f                                   % RCS fuel tank volume
m_in2                                          % ascent inert mass
mp2                                            % ascent prop. mass
V_me2_ox                              % ascent main engine ox. mass
V_me2_f                             % ascent main engine fuel mass
m_in1                                         % descent inert mass
mp1                                           % descent prop. mass
V_me1_ox                      % descent main engine ox. tank volume
V_me1_f                     % descent main engine fuel tank volume
m_stage1                                      % total vehicle mass
```

1.1.2 MLV Descent Throttlability

The following paper was written to convey the challenges that will be encountered in modifying the RL10B-2 for 10:1 throttlability.

Ares Descent Propulsion Feasibility Study

The most critical component of the Mars Landing Vehicle (MLV), and perhaps of the success of the manned mission itself, is the ability of the descent propulsion system (DPS) to safely place the vehicle and crew on the surface of Mars. In essence, the powered descent is a controlled fall. As a result of the ever-changing weight and velocity of the MLV, the DPS thrust must be able to be varied over a wide range of values to facilitate this aspect of the mission. This creates a necessity for throttlability of the descent main engine, the Pratt & Whitney RL10.

The obvious reference for investigating the feasibility and development requirements for this type of engine modification is the Lunar Module (LM) descent propulsion system of the infamous Apollo program. In designing the LM, Grumman set a 10:1 throttlability requirement for the descent propulsion. The engine was a pressure-fed hypergolic system with a maximum thrust production of 44.4 kN. The development of the design was first discussed with Boeing's RocketDyne in February of 1963. The same month, the management team announced that a RFP's would be accepted for a parallel development program. According to Assistant Program Manager Frank Canning, this was due to the fact that the descent stage posed what he called the LEM's "biggest development problem." The year 1963 saw contracts set at roughly $19 million awarded to both RocketDyne and Space Technology Laboratories (STL) to develop the descent propulsion system for the LM. It was anticipated that progress would be slow, and the predictions were, indeed, correct.

In March of 1965, STL stated the major development issues of the descent propulsion were providing combustion stability, high performance, and good erosion characteristics over the entire throttling range. Later that same year, when questioned as to the cause of delays in descent propulsion development, Grumman explained that the issues were of weight, chamber erosion, mixtures, valves, combustion instability, and throttle mechanisms. The design of lightweight, man-rated, throttlable engine with high thrust output was proving to be no walk in the park, but progress was being made. Delivery of the operational descent propulsion system engines was near completion in February of 1968, giving a development time of almost exactly 5 years. This was done with the work of no less than two large propulsion companies and a slew of scientists and engineers. The operation of the LM descent propulsion system however, proved to be nearly flawless.

The development issues that trouble the LM teams would no doubt present themselves in the development of a similar engine for the MLV. This is believed due to the fact that throttling range is assumed to be the same; that is, 10:1. Several factors make our design more challenging, though. As stated, the LM descent propulsion utilized pressure-fed hypergolic propellants. The MLV requires the use of pump-fed cryogenic propellants due to the more demanding nature of the mission as well as the larger thrust and performance requirements, which are governed by the mass of the deliverable payload. The pump-fed system is, by nature, more complex, and the control of valves, plumbing pressures, and ignition system may each prove to be problematic when throttling over a 10:1 range.

Also, the much higher chamber pressures may result in significantly more issues in terms of combustion instability over this throttling range. The implication is that thrust chamber will no doubt require a quite robust design, and this will most likely involve an increase in the dry mass. One particular advantage that our team has over the LM team, however, is that the RL10 has an existing knowledge base that goes back 30 years. Therefore, it can be assumed that a large pool of engineers who are already familiar with the engine, as well as a large collection of test and performance data, would be available to in order to help gain insight throughout the course of the development.

With everything stated, I find it reasonable to believe that the development of a throttlable version of the RL10 for the application of a manned-lander is, as challenging as it may be, both possible and feasible. Based on the 5-year development of the comparatively simplistic LM descent propulsion system, I believe a good initial estimate of the development time required by this engine to be approximately 7-8 years. Based on the cost of the LM descent propulsion program, assuming only one contract is awarded, and including the extended development, the estimated cost of developing this engine approximately $175 million.

"LM Descent Propulsion." Encyclopedia Astronautica. 27 February 2005. Available: http://www.friends-partners.org/partners/mwade/craft/lmdlsion.htm

1.1.3 Resources

The following resources were used throughout the semester.

[1] "Apollo Lunar Module." www.iridis.com. 17 Jan 2005. Available: http://www.iridis.com/glivar/Apollo_Lunar_Module

[2] Bullock, J. R., J. R. Santiago. "RL60-The Next Step in the Evolution of Upperstage Engines." 40th AIAA/ASME/SAE/ASEE Joint Propulsion Conference. 12-14 July. Ft. Lauderdale, FL. AIAA-2004-3529

[3] "Delta II: Second Stage Storable Liquid Rocket Engine." Space and Missile Propulsion. Aerojet.com. 28 February 2005Available: http://www.aerojet.com/program/display.pl?program_ID=16

[4] Heppenheimer, T. A. *Development of the Shuttle, 1972-1981*. History of the Space Shuttle, Vol. 2. Smithsonian Institution Press, 2002.

[5] Hill, Phillip and Carl Peterson. *Mechanics and Thermodynamics of Propulsion*. 2nd Ed. Addison-Wesley, 1992.

[6] Humble, Ronald W., Gary N. Nelson, and Wiley J. Larson. *Space Propulsion Analysis and Design*. 1st Ed., Revised. McGraw-Hill, 1995.

[7] Huzel, Dieter K. and David H. Huang. *Design of Liquid Propellant Rocket Engines*. 2nd Ed. Rocketdyne Division, North American Rockwell, Inc. NASA SP-125, 1972.

[8] Karakulko, Witalij and Scott R. Frazier. "Shuttle Evolution: Improved Aft Propulsion System." 25th Joint Propulsion Conference; Monterey, CA; July 10-12. AIAA Paper 1989-2518.

[9] Larson, Wiley J. and Linda K. Pranke. *Human Spaceflight: Mission Analysis and Design*. McGraw-Hill.

[10] "Launch Escape Subsystem." 28 February 2005. Availabe: http://apollomaniacs.web.infoseek.com.jp/apollo.lese.htm

[11] "Launch Escape Tower." 28 February 2005. Available: http://www.braeunig.us./space/specs/apollo.htm

[12] "LM Descent Propulsion." Encyclopedia Astronautica. 27 February 2005. Available: http://www.friends-partners.org/partners/mwade/craft/lmdlsion.htm

[13] Mc Hugh, B. "Numerical Analysis of Existing Liquid Rocket Engines as a Design Process Starter." Applied Technology Associates: Colorado Springs, CO. AIAA Paper 1995-2970.

[15] Popp, M., J. R. Bullock, J. R. Santiago. "Development Status of the Pratt & Whitney RL60 Engine." 36th Joint AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit. 7-10 July 2002. Indianapolis, IN. AIAA-2002-3587.

[16] Pritchard, E. Brian. *Mars: Past, Present, and Future*. Progress in Astronautics and Aeronautics. Vol. 145. AIAA, 1992.

[17] "Shuttle Propulsion Systems." The Winter Annual Meeting of The American Society of Mechanical Engineers; Pheonix, AZ; Nov. 14-19, 1982. The Aerospace Division, ASME.

[18] Sutton, George P. and Oscar Biblarz. *Rocket Propulsion Elements*. 7th Ed. Wiley-Interscience, 2001.

2 Bogenberger, Brienne

2.1 Crew Transport Vehicle Launch Window Schedule

Author: Brienne Bogenberger

Contributors: Kevin Kloster, Jackie Jaron, Geroge Pollock

2.1.1 Purpose

We created the Crew Transport Vehicle (CTV) Launch Window Schedule to organize and help in selecting mission opportunities.

2.1.2 Results

The mission specifications required that the human mission to Mars should take place between 2014 and 2030. In addition, the requirements required that the CTV must travel on a free-return trajectory outbound to Mars. To facilitate possible mission opportunities, a chart of total delta V's for each mission was complied. Figure 8 shows the delta V chart produced. All delta V numbers include two trajectory correction maneuvers (TCM) of 100 m/s and two apo-twist maneuvers of 500 m/s to align the CV for Mars landing and the return to Earth. The total delta V numbers also include the main capture and departure burns.

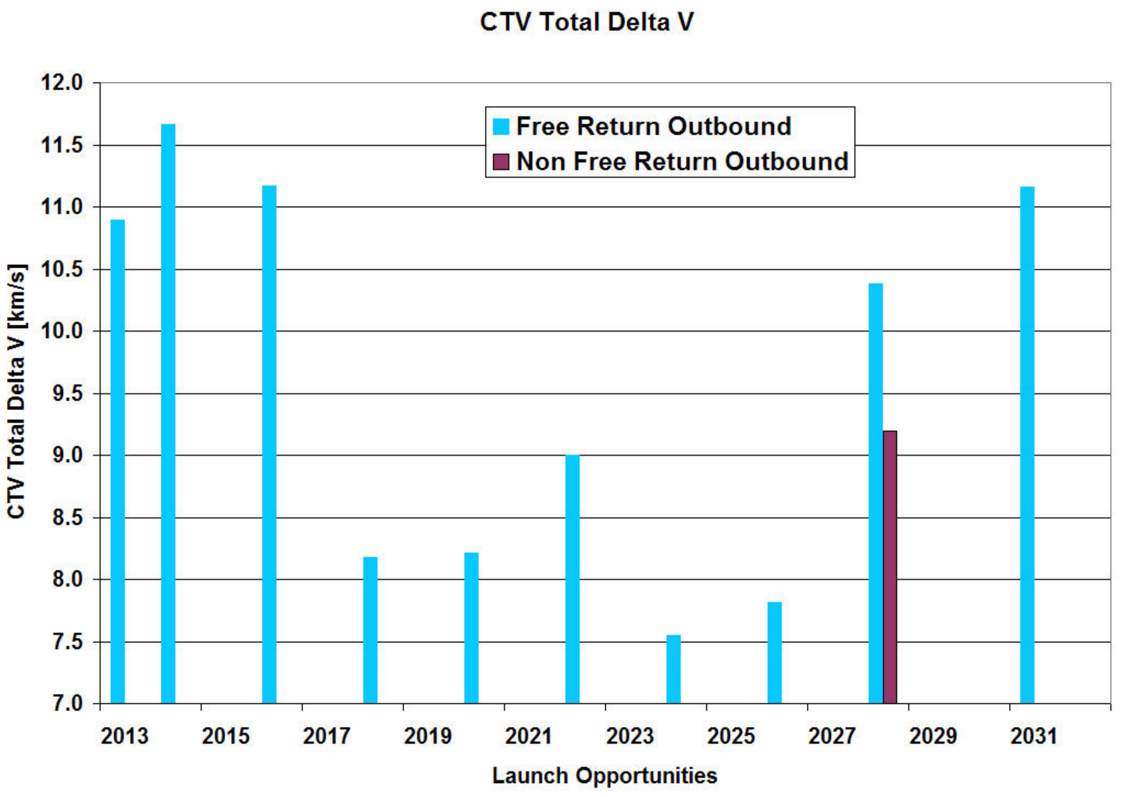


Figure 8 Delta V Chart of Mission Opportunities

We determined the mission opportunities based on the trough of required delta Vs. In order to meet the mission specification to provide six mission opportunities to allow for delays, we originally designed the CTV capable of performing a mission delta V of 10.38 km/s. After many debates, we decided to design the CTV for the non-free return outbound departing Earth in September of 2028. This decision reduced the delta V that the CTV was designed for to 9.2 km/s. We feel that the CTV will be well tested and that the reduction in mass of the CTV out-weighed the risk to the crew. The mission opportunities are listed in Table 1.

Table 1 Mission Opportunity Dates

Mission Opportunity	Earth Departure Date	Mars Arrival Date	Mars Departure Date	Earth Arrival Date	Free Return Earth Arrival Date
1	5/20/2018	1/1/2019	6/20/2020	12/24/2020	6/16/2021
2	7/18/2020	1/27/2021	7/5/2022	4/8/2023	7/3/2023
3	9/8/2022	4/5/2023	7/25/2024	5/13/2025	8/18/2025
4	10/5/2024	9/15/2025	8/1/2026	6/12/2027	9/26/2027
5	10/13/2026	7/1/2027	7/25/2028	7/10/2029	11/6/2029
6	9/25/2028	8/16/2029	8/1/2030	8/7/2031	N/A

3 Boopalan, Avanthi

Author: Avanthi Boopalan

3.1 Tank Insulation

3.1.1 Materials Used

Initially, for the external insulation of the cryogenic propellant tanks for the ELV, the NCFI (North Carolina Foam Industries) – 124 was considered. Because of the dangerous effects of the abrasion of the foam, we applied a coat of high strength phenolic honeycomb material.

The density of the NCFI foam used is 37 kg/m^3 [1] and the densities of the HRH – 10 phenolic honeycomb used as the outer coat and in the bulkhead are 32 kg/m^3 and 48 kg/m^3. [2]

3.1.2 MATLAB Code Listing

The tank structure for the first stage is ellipsoidal, cylindrical structure with spherical caps. The MATLAB functions that perform the task of estimating the insulation numbers are:

tank_insulation.m

This is a driver file that calls *ti.m*, which takes in as inputs, the tank size parameters and outputs the volume and mass of the insulation. For three different cases: Stage-I, liquid oxygen (LOX) tank external-insulation, stage-II external-insulation, and the common bulkhead between the LOX/liquid hydrogen (LH2) tanks.

ti.m: inputs (Tank thickness, inner radius, height, insulation thickness, density)

ti.m calls *vTank.m*, for the external volume with insulation and subtracts from it, the external volume of the tank. Using density values, the insulation mass and volume is calculated.

vTank.m: inputs (radius of the spherical caps, height of main cylinder)

vTank.m outputs the volume of an ellipsoidal tank.

3.1.2.1.1 tank_insulation.m

```
% Avanthi Boopalan
% AAE 450 - ELV Tank Insulation sizing

% Function inputs for ti
% (Tank thickness, inner radius, height, insulation thickness, density)

[v1 m1] = ti(0.03, 5.2, 8.4752, 0.025, 36.75) % LOX Insulation, Stage 1
[v4 m4] = ti(0.03+.025, 5.2, 1.06, 0.004, 32) % Top Coat
[v2 m2] = ti(0.03, 5.2, 1.06, 0.036, 36.75) % Outer Insulation stage 2
[v5 m5] = ti(0.03+.036, 5.2, 1.06, 0.004, 32) % Top Coat
[v3 m3] = ti(0.03, 5.2, 0, .045, 48) % Common bulkhead

v3 = v3*.5
m3 = m3*.5
V = v1 +v2 + v3+v4+v5 % Total Volume of insulation
M = m1 + m2 + m3+m4+m5 % Total Mass
```

3.1.2.1.2 ti.m

```
function [vol, mass] = ti(ot1, or1, h, t1, rho)
% Tank insulation volume and mass
% Inputs: Tank thickness, inner radius, height, insulation thickness, density
% Avanthi Boopalan

r1 = or1 + ot1; % Radius of tank
r2 = r1 + t1; % Radius of tank + insulation thickness
vol = vTank(r2,h) - vTank(r1,h); % Insulation Volume
mass = vol.*rho; % Insulation mass
```

3.1.2.1.3 vTank.m

```
function vol = vTank(r,h)
% If h = 0, gives volume of sphere
% Inputs radius of spherical caps and height of the cylinder
% Avanthi Boopalan
vol = 4/3.*pi.*r.^3 + pi.*r.^2.*h;
% Volume of half-sphere + cylinder
```

References:

[1] "Thermal Protection System". NASA Facts. 2004. National Aeronautics and Space Administration. August 2004 <http://www.nasa.gov/pdf/63758main_TPS_FACT_SHEET.pdf>

[2] "HexWeb™ HRH® - 10 Aramid Fiber/Phenolic Honeycomb" Product Data. Hexcel Composites. 5 Apr. 2005. http://www.hexcelcomposites.com/NR/rdonlyres/etrxiwfc5mqidfvatrgqllazcptu6fb74qw4mipdp4d5zlwhzmr3we2cnjjm3jwyn7uhrkbcckbcn2pd2vu32bgbpig/HexWeb+HRH-10(EU).pdf

3.2 Solar heating and Thermal Cycles

Author: Avanthi Boopalan

Contributors: Kevin Kloster

3.2.1 Description

The crew quarters of the CTV is subjected to rotational motion (for artificial gravity). The solar heating of the crew quarters and the temperature cycles along the trajectory are analyzed.

3.2.2 Theory and Assumptions

Initially, a model of the crew quarters with an exposed half-surface of 50 m^2 was used. The temperature fluctuations for the exposed halves and the total cycles were calculated using Stefan-Boltzmann Law and the inverse square laws. See [1].

We make an assumption that the solar radiation is normal to the exposed surface, and also an initial temperature of 320 K to study the changes in temperature.

Table 3-1 Temperature Cycles

Loading Results	
Cycles	Hot/Cold absolute difference
9.5 x 106	0.011K (max)

Table 3-1 shows the temperature cycles and the maximum temperature difference.

The Stefan Boltzmann Law is given as:

$$E_b = \sigma T^4$$

Where E_b is the irradiance of the sun in W/m^2, σ is the Stefan Boltzmann constant which has a value of 5.67E-8 W/m^2 K^4 and T is the temperature of the sun. Using the Sun's black body temperature (5777 K), we calculate the irradiance due to the sun which is maximum (1.4 W/m^2) when the Crew Transport Vehicle (CTV) is near earth, and minimum (0.5 W/m^2) when the CTV is near Mars. The value of irradiance decreases with the inverse square law. We used a MATLAB script which follows the analysis section. The script uses these laws to estimate the temperature fluctuations along the points of the trajectory.

3.2.3 Analysis

Using a MATLAB script, the temperature fluctuations for different points on the trajectory are studied, Figure 3-1 shows that the temperature change is minor.

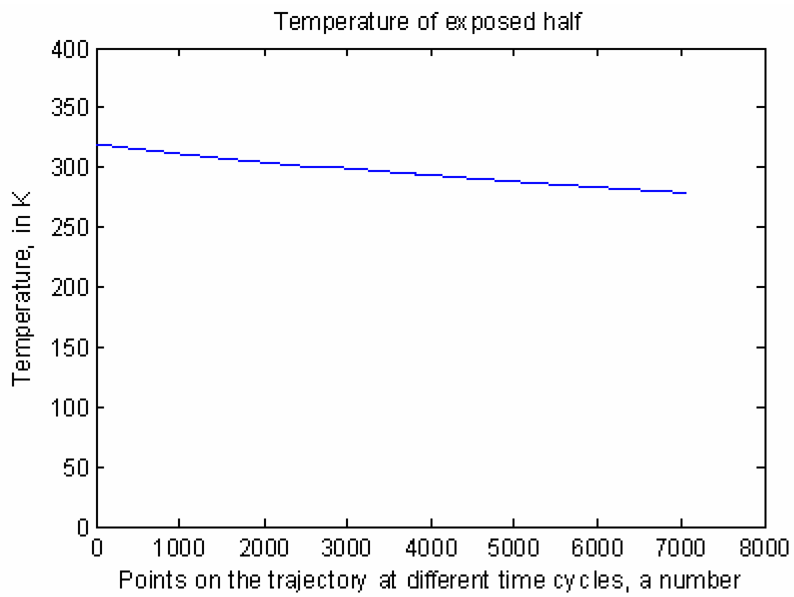


Figure 3-1 Temperature Fluctuations Estimation

The MATLAB code, temp_est.m estimates the temperature cycles. In the code, we first calculate the temperature as a function of the angle that is subtended by the trajectory of the CTV. The angle is theta, which has a value of zero at the trajectory starting point, Earth parking orbit and 180 degrees near Mars. Using inverse square law we calculate the irradiance due to the sun along the CTV path. We use aluminum as the material which has an emissivity of 0.82 and an absorptivity of 0.14. By applying the energy equation, we calculate the temperature of the body along the different points in the trajectory.

3.2.4 MATLAB Script: temp_est.m

```
% Estimates temperature fluctuation between exposure and non-exposure
% of the cylindrical half shell that represents the crew quarters of
% the CTV. Outputs the Cycles ~ 9.5 million
clear all
close all
% Half side temperature estimation
T = 5777; % Sun's Temp. Kelvin
sigma = 5.67e-8; % Stefan- Boltzmann Constant W/sq.met/quad.K
E = sigma .* T.^4; % Stefan- Boltzmann Law, 6.3153e+007
theta = 0; % Temperature estimation near earth orbit
           % ie., when theta = r , which is the distance between
           % the sun and CTV becomes the Sun - Earth distance
           % which is an approx. to the earth orbit distance
P = 185029700.203e3; % m, from DnC gp
Rsun = 6.955e8; % m
e = 0.23685; % eccentricity
r = P./(1+e.*cos(theta)); % m, Earth-Sun distance
arc = 0;
dth = 0.001;
for theta = 0:0.001:pi
    r = P./(1+e.*cos(theta));
    arc = r.*dth + arc;
end

vel = arc./(1.5*365*24*3600);
t_hot = 5;
dist_hot = t_hot*vel;
cycles = arc/dist_hot


% Material: Aluminum
alfa = 0.14; % Absorptivity
emis = 0.82; % Emissivity
Cp = 875; % J/(kg.K)
rho = 2770; % kg/(cu.m)

r_crew = 5; % m
h_crew = 3; % m
t_crew = 0.02; % m = 2 cm
A = r_crew.*h_crew.*pi; % Half Area of Crew cylinder, the exposed side
vol = 0.5.*pi.*h_crew.*(r_crew.^2 - (r_crew - t_crew).^2); % Volume of half-shell
m = rho.*vol; % Mass of half shell
I0 = E.*Rsun.^2./r.^2; % Irradiance
G0 = I0.*A.*alfa;
G = G0;
t = 1;

T0 = 320; % Initial guess - AOI
dt = 1;
flag = 1;
dth = .001;
k = 1;

% Plotting the beginning portion of the trajectory to get the maximum.

for theta = 0:.001:.002;
    r = P./(1+e.*cos(theta));
    I0 = E.*Rsun.^2./r.^2; % Irradiance
    G0 = I0.*A.*alfa;
```

```
        G = G0;
        txp = r*dth./vel;
        txp = round(txp);
        Number = k
        theta
        section_time = txp./3600
        kilometers = r*dth./1000
        for c = 1:txp
            if rem(c,5) == 0
                kn(k) = k;
                T1(k) = (1./(m.*Cp)).*(G-A.*emis.*sigma.*T0.^4).*dt + T0; % Equation - AOI
                dT(k) = abs(T1(k) - T0);
                T0 = T1(k);
                k = k+1;
                if flag == 1
                    G = 0;
                    flag = 0;
                else
                    G = G0;
                    flag = 1;
                end
            end
        end
    end

    plot(kn , T1)
    title('Temperature of exposed half')
    xlabel('Number')
    % Number is a dummy variable that depends on the distance and time
    ylabel('Temperature, in K')
    figure(2)
    plot(kn, dT)
    title('Abs. difference of exposed half')
    xlabel('Number')
    ylabel('Temperature, in K')
```

References:

[1] Incropera, Frank, P. and DeWitt, David P. Fundamentals of Heat and Mass Transfer, 5th Edition. USA: John Wiley & Sons, Inc., 2002.

3.3 Design of a Mars Ground Transportation Vehicle

3.3.1 Description

During the design process for the Mars Habitat Vehicle (MHV), splitting of the MHV was considered due to massive heat-shield design concerns. The figure below shows a design for Mars ground transportation vehicle:

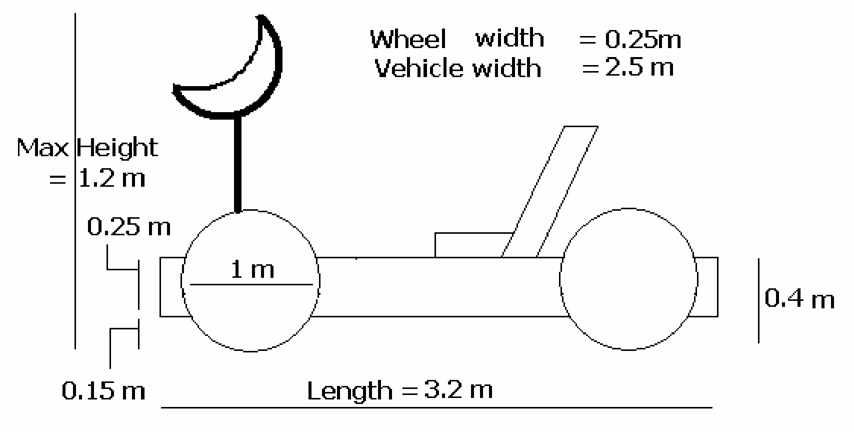


Figure 3-2 Design for Mars Ground Transportation Vehicle

3.3.2 Requirements:

The design was considered assuming one kilometer distance between MHV halves, for a stay of 500 days (16 months). With two passes per month, the total distance traveled for the stay on Martian surface is 70 km. The payload is assumed to be two astronauts and their suits amounting to 250 kg. Also other equipment on-board the vehicle is assumed to be 250 kg. So, a total of 500 kg, is considered as the carry-on mass requirement. Based on the design presented in Figure 3-2, the folded volume of the ground transportation is four cubic meters, with a mass of 430 kg. This design is not considered as the MHV is not divided in the final design. The design is based on the Apollo Lunar Rover Vehicle [1]

References:

[1] Williams, David, R. The Apollo Lunar Roving Vehicle. 11 Dec. 2003. NASA Goddard Space Flight Center. 5 Apr. 2005.

<http://nssdc.gsfc.nasa.gov/planetary/lunar/apollo_lrv.html>

4 Bradley, Kathryn

4.1 Appendix

4.1.1 Radiation Code

The Matlab code we develop calculates the total radiation exposure the astronauts experience during the three different mission timelines. It breaks down the radiation the astronauts are exposed to for each leg of the mission as well as the total radiation exposure for the entire mission.

```
%Kathryn Bradley
%AAE 450 Senior Design
%Radiation Shielding Calculator
%16 February 2005

close all
clear all
clc

%rate of exposure without shielding
GCR_rate = .021; %rem/hr
Solar_rate = .001; %rem/hr
Reactor_radiation_rate = 0; %rem/hr

%Blocking due to 10g/cm^2 CO2 on surface of Mars
Atmospheric_block = .0205; %rem/hr

%Blocking due to Van Allen belt in Earth Orbit
Van_allen_block = .01513; %rem/hr

%User inputs total time in each location
In_transit_time = [593 1091 1095] %days
Decent_to_mars = [2 2 0] %days
On_martian_surface_time = [500 2 0] %days
In_earth_orbit_time = [5 5 5] %days

Total_trip_time = In_transit_time + Decent_to_mars + On_martian_surface_time +
In_earth_orbit_time % total trip time in days
Total_trip_time_year = Total_trip_time/365.25; %total time in years

Days_hours = 24; %hours in a day

In_transit_time_hours = In_transit_time.*Days_hours; %hours
Decent_to_mars_hours = Decent_to_mars.*Days_hours; %hours
On_martian_surface_time_hours = On_martian_surface_time.*Days_hours; %hours
In_earth_orbit_time_hours = In_earth_orbit_time.*Days_hours; %hours

%radiation rate exposure in location
Radiation_rate_in_transit = GCR_rate + Solar_rate + Reactor_radiation_rate; %rem/hr
Radiation_rate_decent_to_mars = GCR_rate + Solar_rate; %rem/hr
Radiation_rate_on_surface = (GCR_rate + Solar_rate) - Atmospheric_block +
Reactor_radiation_rate; %rem/hr
```

```
Radiation_rate_in_earth_orbit = (GCR_rate + Solar_rate) - Van_allen_block +
Reactor_radiation_rate; %rem/hr

%Radiation exposure in location
Radiation_exposure_in_transit = Radiation_rate_in_transit.*In_transit_time_hours %rem
Radiation_exposure_decent_to_mars =
Radiation_rate_decent_to_mars.*Decent_to_mars_hours %rem
Radiation_exposure_on_surface =
Radiation_rate_on_surface.*On_martian_surface_time_hours %rem
Radiation_exposure_in_earth_orbit =
Radiation_rate_in_earth_orbit.*In_earth_orbit_time_hours %rem

Total_mission_exposure = Radiation_exposure_in_transit +
Radiation_exposure_decent_to_mars + Radiation_exposure_on_surface +
Radiation_exposure_in_earth_orbit %rem
Average_exposure_per_day = Total_mission_exposure./Total_trip_time %rem/day
Average_exposure_per_year = Total_mission_exposure./Total_trip_time_year %rem/yr
```

This code is used to calculate the radiation exposure and the shielding that is necessary to reduce cancer risks to an acceptable rate and also keep our astronauts alive during the total duration of the mission.

4.1.2 Excel Spreadsheet

We then develop an excel spreadsheet to determine the need for shielding of the nuclear reactors to prevent the astronauts from receiving radiation sickness. The three sheets show that without shielding the astronauts are exposed to large amounts of radiation even with a distance of up to 50m from the reactor. In addition, the sheet with a shielded reactor is used to calculate the shielding needed to bring the total mission radiation exposure to approximately 500 rem. These sheets were created for the original mission specifications of a 500 day stay on the planet and an 1100 day mission length. The radiation exposures are seen below in the tables below. (Table 2, Table 3, Table 4, Table 5, Table 6, Table 7, Table 8, Table 9, Table 10)

Table 2 Unshielded C-3 Mission Radiation Exposure with Nuclear Reactor 23 meters away

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	593	66920	380.94
CTV to Surface	2	1.06	1.06
Mars Surface	500	18	18
Total Earth Orbit	5	562.42	100
Total	1100	67501.48	500

Table 3 Unshielded C-2 Mission Radiation Exposure with Nuclear Reactor 23 meters away

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	1091	123120	398.87
CTV to Surface	2	1.06	1.06
Mars Surface	2	.07	.07
Total Earth Orbit	5	562.42	100
Total	1100	123683.55	500

Table 4 Unshielded B-2 Mission Radiation Exposure with Nuclear Reactor 23 meters away

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	1095	123570	400
CTV to Surface	0	0	0
Mars Surface	0	0	0
Total Earth Orbit	5	562.42	100
Total	1100	124132.42	500

Table 5 Unshielded C-3 Radiation Exposure with Nuclear Reactor 50 meters away

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	593	16965	420.94
CTV to Surface	2	1.06	1.06
Mars Surface	500	18	18
Total Earth Orbit	5	141.22	60
Total	1100	17125.28	500

Table 6 Unshielded C-2 Radiation Exposure with Nuclear Reactor 50 meters away

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	1091	31211	438.87
CTV to Surface	2	1.06	1.06
Mars Surface	2	.07	.07
Total Earth Orbit	5	141.22	60
Total	1100	31353.35	500

Table 7 Unshielded B-2 Radiation Exposure with Nuclear Reactor 50 meters away

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	1095	31326	440
CTV to Surface	0	0	0
Mars Surface	0	0	0
Total Earth Orbit	5	141.22	60
Total	1100	31467.22	500

Table 8 Unshielded C-3 Radiation Exposure with No Nuclear Reactor

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	593	313.1	470.12
CTV to Surface	2	1.06	1.06
Mars Surface	500	28	28
Total Earth Orbit	5	.82	.82
Total	1100	342.98	500

Table 9 Unshielded C-2 Radiation Exposure with No Nuclear Reactor

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	1091	576.05	498.05
CTV to Surface	2	1.06	1.06
Mars Surface	2	.07	.07
Total Earth Orbit	5	.82	.82
Total	1100	578	500

Table 10 Unshielded B-2 Radiation Exposure with No Nuclear Reactor

	Time (days)	Radiation Exposure (rem)	Total Allowable Exposure for Mission (rem)
Total Transit	1095	578.16	499.18
CTV to Surface	0	0	0
Mars Surface	0	0	0
Total Earth Orbit	5	.82	.82
Total	1100	578.98	500

Once the sheets are generated we put these values into a more organized equation sheet within excel. We calculate the shielding rates based on our shielding and plug this into the sheet to check that the radiation exposure is within the limits we have set for total exposure. The mission timelines for the normal missions can be seen in Table 11. The radiation dosages for those missions can be seen in Table 12.

Table 11 Mission Timeline for Worst Case Normal Mission by Kathryn Bradley and Conrad Golbov

Mission Time *days	Mission B-2	Mission C-1	Mission C-2
Earth Orbit	5	5	5
In Transit	989	1002	791
Mars Surface	0	1	250
Total Mission	994	1008	1046

Table 12 Radiation Dosage for Worst Case Normal Mission by Kathryn Bradley and Conrad Golbov

Radiation Rates After Shielding	Rates (rem/hr)	Mission B-2 (rem)	Mission C-1 (rem)	Mission C-2 (rem)
Earth Orbit	.004	.464	.464	.464
In Transit	.019	450.984	456.912	360.696
Mars Surface	.002	0	.056	14
Total Mission		451.448	457.432	375.160

We must also consider the radiation exposure for the free-return worst case mission scenario. The timeline for the free-return missions can be seen in Table 13. The radiation dosage can be seen in Table 14.

Table 13 Mission Timeline for Worst Case Free-Return Mission

Mission Time *days	Mission B-2	Mission C-1	Mission C-2
Earth Orbit	5	5	5
In Transit	1118	1101	1115
Mars Surface	0	0	0
Total Mission	1123	1106	1120

Table 14 Radiation Dosage for Worst Case Free-Return Mission

Radiation Rates After Shielding	Rates (rem/hr)	Mission B-2 (rem)	Mission C-1 (rem)	Mission C-2 (rem)
Earth Orbit	.004	.464	.464	.464
In Transit	.019	509.808	502.056	508.44
Mars Surface	.002	0	0	0
Total Mission		510.272	502.52	508.904

We then create an excel spreadsheet that allows us to keep track of the mass and materials that are used in the different shielded areas. We make a sheet that will allow us to track the shielded room, exterior shielding, reactor shielding and also the engine shielding masses and volumes. James Kallimani took over the engine and reactor shielding and as such only the shielded room and exterior shield will be tracked in this section of the report. The mass, volume and materials used in shielding can be seen in Table 15 and Table 16.

Table 15 CTV/MHV Shielded Room Mass, Volume and Arial Density

Aluminum Thickness (cm)	5
Polyethylene Thickness (cm)	13
Total Thickness (cm)	18
Density of Aluminum (g/cm^3)	2.78
Density of Polyethylene (g/cm^3)	1
Total Arial Density (g/cm^2)	26.9
Room Surface Area (m^2)	24
Total Shield Mass (kg)	6456

Table 16 CTV Exterior Shield Mass, Volume and Arial Density

Aluminum Thickness (cm)	1
Polyethylene Thickness (cm)	4
Total Thickness (cm)	5
Density of Aluminum (g/cm^3)	2.78
Density of Polyethylene (g/cm^3)	1
Total Arial Density (g/cm^2)	6.78
Room Surface Area (m^2)	190
Total Shield Mass (kg)	12,882

4.1.3 Catia Modeling

Next we design the shielded room within Catia. Aaron Sippel assisted the author in the development of the Catia model which can be seen below in Figure 3.

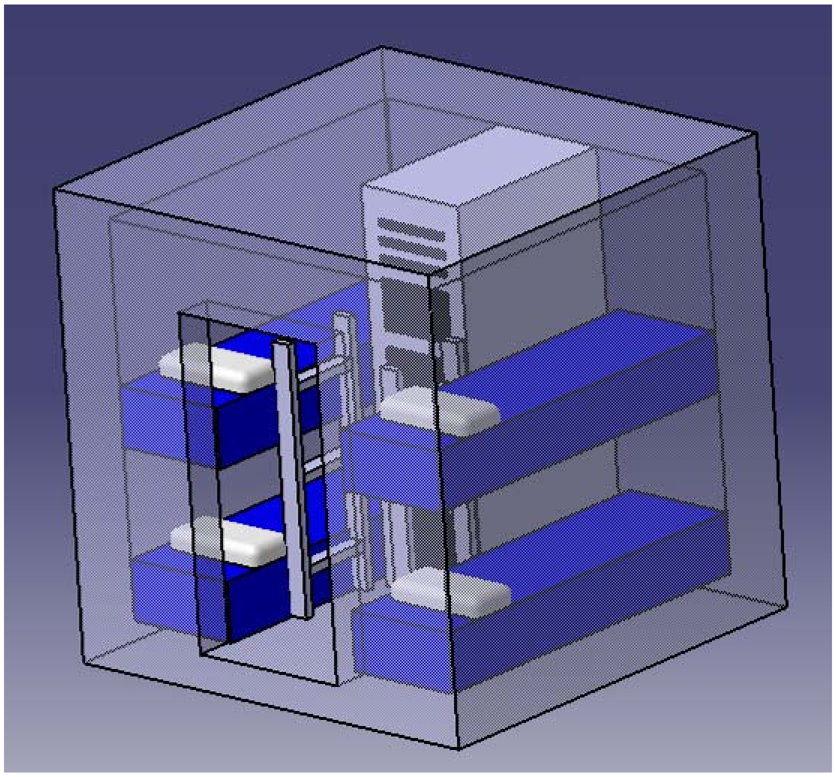


Figure 3 Catia model of CTV/MHV Shielded room, developed by Kathryn Bradley and Aaron Sippel

References:

[1] NCRP Report 98, *Guidance on Radiation Received in Space Activities.* National Council on Radiation Protection and Measurements. July 31, 1989. Bethesda, MD.

[2] Stilwell, Don, Ramzy Boutros, and Janis H. Connolly. "Crew Accommodations." _Human Spaceflight: Mission Analysis and Design_. Ed. Wiley J. Larson and Linda K. Prank. New York: McGraw-Hill, 1999. 575-606.

[3] Shibasaki, Kiyoto. *Signature of Energy Release and Particle Acceleration Observed by the Nobeyama Radioheliograph.* http://solar.nro.nao.ac.jp/user/shibasak/CESRA2001/CESRA2001.pdf

4.2 Docking

It is also important to consider docking for the different vehicles. We first research the different docking technologies that are available for our use. Then we must perform calculations to prove the need for autonomous docking. The calculation that is performed to determine separation distance, can be seen below.

4.2.1 Calculating Maximum Separation

Calculate the maximum angular separation between chaser and target vehicles using Equation 1:

Equation 1

$$\cos(\lambda_{\max}) = \frac{R_p}{(R_p + H)}$$

Maximum angular separation to remain in contact can be calculated using Equation 2:

Equation 2

$$2\lambda_{\max}$$

Calculate the maximum allowable separation on orbit to stay in contact using Equation 3:

Equation 3

$$l = r2\lambda_{\max}$$
$$r = R_p + H$$

Next we plot the separation distance for different altitudes to determine the maximum separation distance between the craft to remain in contact Figure 4.

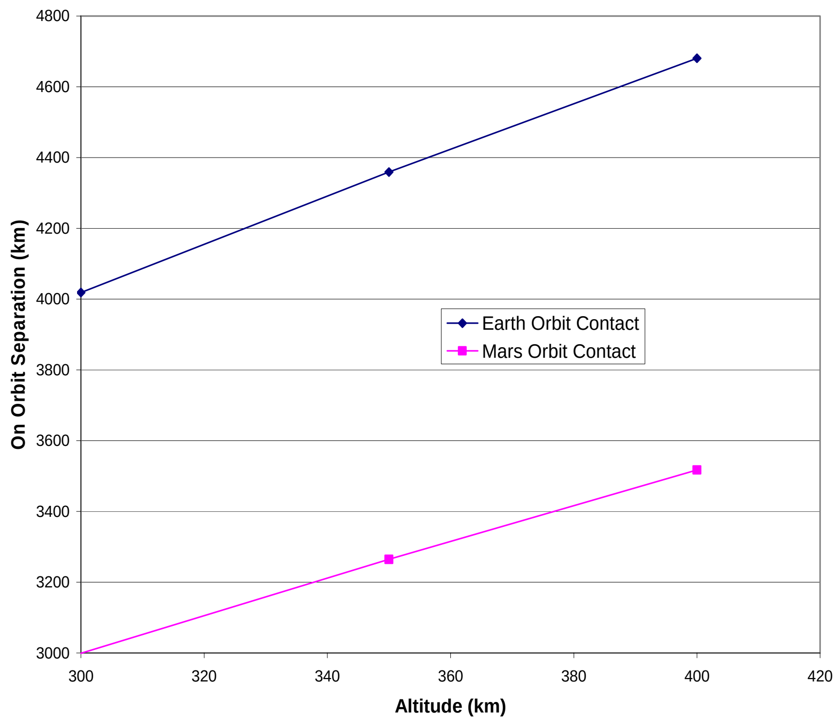


Figure 4 Separation Distance between craft at Mars and Earth

We then create a table with the results of the separation distance calculation at both Mars and Earth. These results can be seen in Table 17 and Table 18.

Table 17 Separation Distance at Earth with Radius = 6378.14km

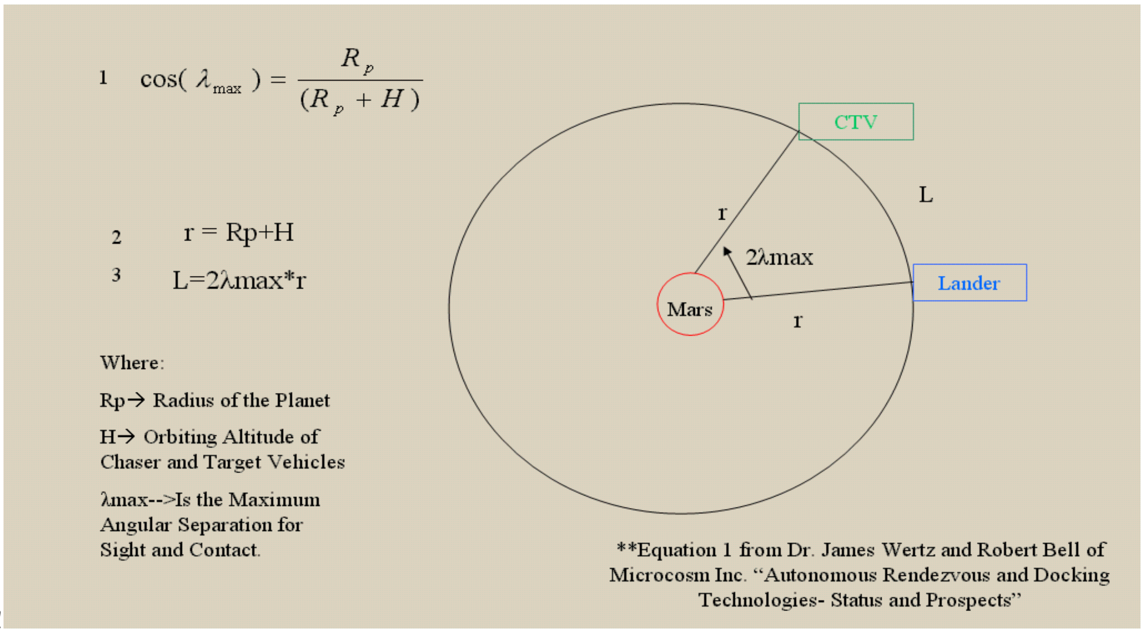
Altitude (km)	Separation Angle (rad)	2*Separation Angle (rad)	Separation Distance (km)
300	.30	.60	4018.6
350	.32	.65	4359.4
400	.35	.69	4680.5

Table 18 Separation Distance at Mars with Radius = 3397km

Altitude (km)	Separation Angle (rad)	2*Separation Angle (rad)	Separation Distance (km)
300	.41	.81	2999.2
350	.44	.87	3264.8
400	.46	.93	3517.1

The sketch below better illustrates what separation distance is and how we may calculate it at Earth and Mars Figure 5.

Error!



1 $\cos(\lambda_{\max}) = \dfrac{R_p}{(R_p + H)}$

2 $r = Rp+H$

3 $L=2\lambda max*r$

Where:

Rp→ Radius of the Planet

H→ Orbiting Altitude of Chaser and Target Vehicles

λmax-->Is the Maximum Angular Separation for Sight and Contact.

**Equation 1 from Dr. James Wertz and Robert Bell of Microcosm Inc. "Autonomous Rendezvous and Docking Technologies- Status and Prospects"

Figure 5 Sketch of Separation Distance

References:

[1] Wertz, James R. & Bell, Robert. "Autonomous Rendezvous and Docking Technologies Status and Prospects" Microcosm Inc. SPIE AeroSense Symposium, Paper No. 5088-3. 2003.

[2] Michigan Aerospace Corporation. Autonomous Rendezvous and Docking, Astronautics, Defense Avionics and Commercial Remote Sensing Division. www.michiganaero.com/satellitedocking/ard.shtml

[3] Tchoryk, Pete, Hays, Anthony B. & Pavlich, Jane C. "A Docking Solution for On-Orbit Satellite Servicing: Part of the Responsive Space Equation" Michigan Aerospace Corporation. AIAA Publication. 1st Responsive Space Conference. April 1-3, 2003. Redondo Beach, CA.

5 Browning, Pete

5.1 Guidance and Control Hardware Appendix

Author: Pete Browning

5.1.1 Computer Network

We model the structure of the network of computers after the computer network of the International Space Station (ISS) currently orbiting the Earth. **Error! Reference source not found.** shows a thorough breakdown of the components we use to operate the CTV systems.

Table 5-1 Individual Hardware Component - Mass, Power, and Volume Breakdown [1]

Component	Operating Units	Spare Units	Total Units	Unit Mass (kg)	Total Mass (kg)	Unit Power (W)	Total Power (W)	Unit Volume (m^3)	Total Volume (m^3)
Tier 1 Command Computers	3	3	6	3.05	18	60.0	180	0.00316	0.019
Tier 1 Emergency Computer	1	1	2	3.05	6	60.0	60	0.00316	0.006
Tier 2 Subsystem Computers	4	4	8	3.05	24	60.0	240	0.00316	0.025
Tier 3 Subsystem Computers	8	8	16	3.05	49	60.0	480	0.00316	0.051
RF Hubs	3	3	6	0.34	2	12.5	38	0.00118	0.007
Personal Workstations	4	0	4	3.05	12	60.0	240	0.00316	0.013
File Server	1	1	2	3.05	6	60.0	60	0.00316	0.006
Replacement Batteries	0	2	2	0.37	1	-	-	0.00039	0.001
Ethernet Cable [m]	1000	10	1010	0.03	30	-	-	0.00002	0.020
Coaxial Cable [m]	1750	18	1768	0.03	53	-	-	0.00002	0.035
Safety Factor (2%)	-	-	-	-	4	-	26	-	0.037
Totals					**206**		**1323**		**0.22**

The ISS is an excellent model for the CTV hardware since it is very massive, currently operational, and life sustaining. The ISS currently employs over 50 laptop computers on board [2]. We choose to bring 32 since the ISS uses some of the computers to operate scientific experiments, which are not present on the CTV. We assume that 16 computers provide the required monitoring and control over all systems, and 16 serve as redundant backups.

We estimate the length of ethernet and coaxial cables to supply based on the 67 meter length of the CTV. Several cables are run the length of the truss to supply data from the communications dishes, main engines, power reactors, and fuel tanks to the crew compartment. We assume that micrometeorites will damage some of the cables, and therefore redundant cables are also provided. The cables also connect systems in and around the Crew Compartment.

The wireless access points serve as a backup to the ethernet and coaxial cables in case of emergency. Crucial systems necessary for survival are accessible via wireless connection.

5.1.2 References

[1] Mars Or Bust, LLC. "Mars Or Bust: A Mars Surface Habitat Design for ASEN 5158." Dec. 2003. University of Colorado. 5 Apr. 2005. <http://www.colorado.edu/ASEN/project/mob/C3Report3 wo charts.doc>

[2] "Laptop Computers on ISS." July 2003. Marex-MG. 5 Apr. 2005. <http://www.marexmg.org/hardware/computers.html>

5.2 Pointing Method Appendix

Author: Pete Browning

5.2.1 Analysis and Results

The basis of the pointing method is drawn from Euler's Law,

$$M = \dot{H} = \frac{\Delta H}{\Delta t} \Rightarrow \Delta H = M\,\Delta t$$

Equation 5-1

where M is the moment about the center of mass and $\dot{H}$ is the rate of change of the angular momentum vector. The moment is composed of the cross product of the moment arm radius vector ($\vec{r}$) and the force vector ($\vec{F}$) which are perpendicular ($\vec{r} \times \vec{F} = rF = M$), therefore

$$\Delta H = r\,F\,\Delta t$$

Equation 5-2

Using the equation for specific impulse, $I_{sp} = F/\dot{m}\,g_o$ where I_{sp} is the specific impulse, F is the force, $\dot{m}$ is the mass flow rate, and g_o is the acceleration due to gravity at Earth's surface = 9.81 (m/s^2), the force can be solved for and substituted into Equation 5-2, thus:

$$\Delta H = r\,I_{sp}\,g_o\dot{m}\,\Delta t$$

Equation 5-3

The amount of propellant used per unit of time is shown to be: $\Delta m_{propellant} = \dot{m}\,\Delta t$, which is substituted into Equation 5-3. We then solve the new equation, $\Delta H = r\,I_{sp}\,g_o\,\Delta m_{propellant}$ for propellant mass:

$$\Delta m_{propellant} = \left[\frac{\Delta H}{H}\right]\frac{H}{r\,I_{sp}\,g_o}$$

Equation 5-4

where H is the angular momentum, and specifically $\frac{\Delta H}{H}$ is the total angular displacement if the angular momentum vector. We can write the angular momentum (H) as H=Iω, where I is the inertia in the spin plane, and ω is the spin rate. A computer code is generated to solve total angular displacement for the free return mission duration. Thus Equation 5-4 becomes:

$$\Delta m_{propellant} = \left[\frac{\Delta H}{H}\right]\frac{I\omega}{r\,I_{sp}\,g_o}$$

Equation 5-5

We solve the total angular displacement, $\frac{\Delta H}{H}$, using the orbital geometry. We know the keplarian elements of the free-return trajectory and Earth's orbit from their periods. [See Section 2.3.2.3] Thus, we can calculate the positions of the CTV and Earth at every time instant. The time interval chosen is one day. We subtract the position of the CTV from the position of the Earth each day and find the vector pointing from the CTV to the Earth for each day of the mission. Next the daily angular displacement is calculated with trigonometry as shown in the figure below:

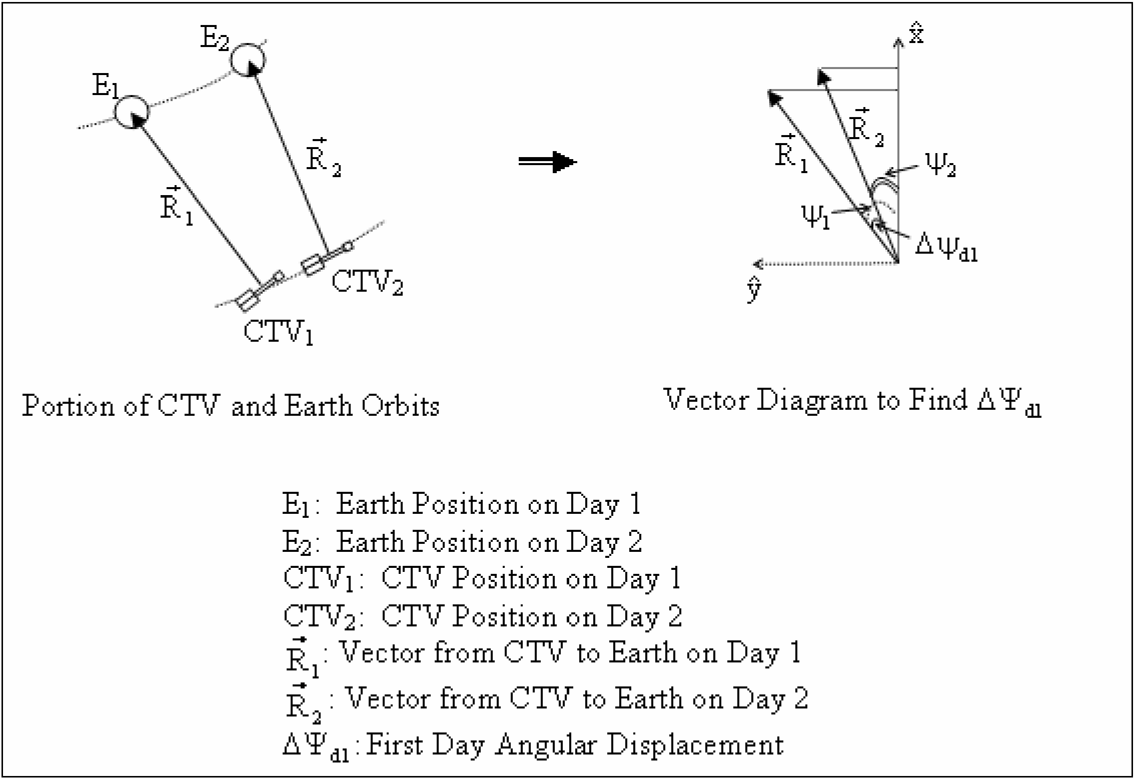

E_2
E_1
$\vec{R}_2$
$\vec{R}_1$
CTV_2
CTV_1
Portion of CTV and Earth Orbits
$\hat{x}$
$\vec{R}_2$
$\vec{R}_1$
Ψ_2
Ψ_1
$\Delta\Psi_{d1}$
$\hat{y}$
Vector Diagram to Find $\Delta\Psi_{d1}$
E_1: Earth Position on Day 1
E_2: Earth Position on Day 2
CTV_1: CTV Position on Day 1
CTV_2: CTV Position on Day 2
$\vec{R}_1$: Vector from CTV to Earth on Day 1
$\vec{R}_2$: Vector from CTV to Earth on Day 2
$\Delta\Psi_{d1}$: First Day Angular Displacement

Figure 5-1 Changing Radius Vectors from the CTV to Earth

The first daily angular displacement, $\Delta\Psi_{d1}$, is found using the following equation:

$$\Psi_1 = \tan^{-1}\left(\frac{\vec{R}_1 \cdot \hat{x}}{\vec{R}_1 \cdot \hat{y}}\right) + \sin^{-1}\left[\frac{\vec{R}_1 \cdot \hat{z}}{\sqrt{\left(\vec{R}_1 \cdot \hat{x}\right)^2 + \left(\vec{R}_1 \cdot \hat{y}\right)^2}}\right]$$

$$\Psi_2 = \tan^{-1}\left(\frac{\vec{R}_2 \cdot \hat{x}}{\vec{R}_2 \cdot \hat{y}}\right) + \sin^{-1}\left[\frac{\vec{R}_2 \cdot \hat{z}}{\sqrt{\left(\vec{R}_2 \cdot \hat{x}\right)^2 + \left(\vec{R}_2 \cdot \hat{y}\right)^2}}\right]$$

$$\Delta\Psi_{d1} = \left|\Psi_2 - \Psi_1\right|$$

Equation 5-6

Note that the second term of each angle accounts for the third dimension because the CTV is traveling on an inclined orbit to Mars with respect to the Earth. We show in Figure 5-2, the daily angular displacement throughout the free-return mission. For the first day, $\Delta\Psi_{d1}$ is about 1.03°.

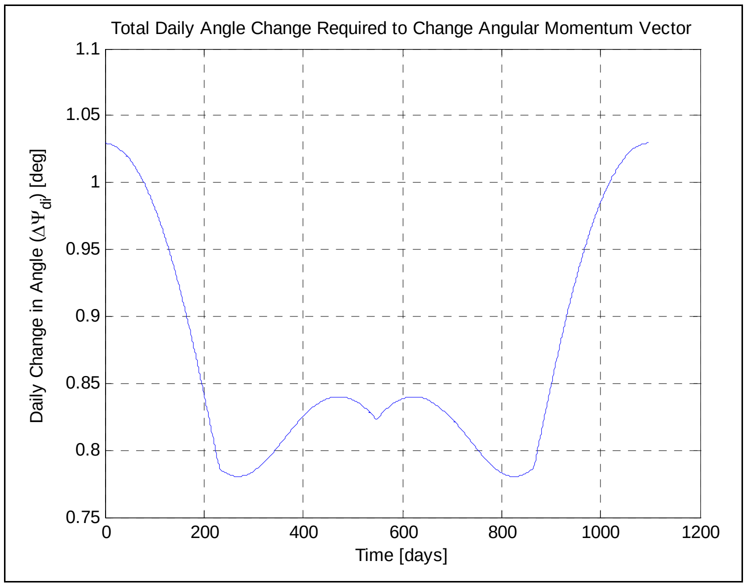


Figure 5-2 Daily Angle Change [deg] Required to Change Angular Momentum Vector

This process is repeated for each day of the mission. We sum all individual daily angular displacements to find the total mission angular displacement:

$$\frac{\Delta H}{H} = \sum_{i=1}^{1100} \Delta\Psi_{di} = 16.5 \text{ rad } (\approx 950^{\circ})$$

Equation 5-7

The XIPS engines aboard the CTV have an Isp of 3800 seconds. The inertia, I, in the spin plane is calculated to be 1.6×10^8 Nm, and to achieve artificial gravity the spacecraft must have a spin rate, ω, of 0.4985 rad/sec (4.8 rpm), and the moment arm, r, is equal to 39.4 meters. Substituting all of these values and the result of Equation 5-7 into Equation 5-5 results in the total mass of propellant needed to change the angular momentum vector so that the CTV can communicate with Earth:

$$\begin{aligned}\Delta m_{\text{propellant}} &= \left[\sum_{d=1}^{1100} \Delta\Psi_d\right] \frac{I\omega}{r\,Ispg_o} \\ &= \left[16.5 \text{ rad}\right] \frac{(1.6\times10^8 \text{ kg-m}^2)(0.50 \frac{\text{rad}}{\text{sec}})}{(39.4 \text{ m})(3800 \text{ sec})(9.81 \frac{\text{m}}{\text{sec}^2})} \\ &= 897.5 \text{ kg}\end{aligned}$$

Equation 5-8

Figure 5-3 shows the daily allotment of the propellant. Also shown is the total propellant required for the mission, which is the result of Equation 5-8.

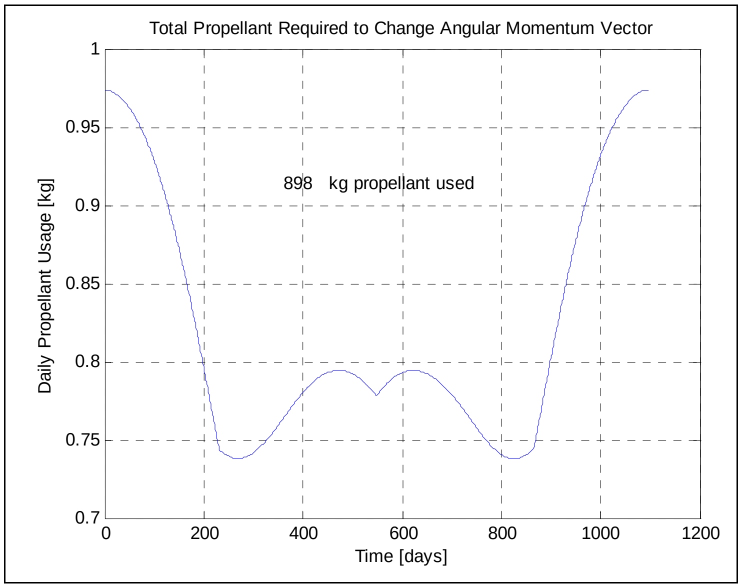


Figure 5-3 Daily Propellant Usage [kg] Required to Change Angular Momentum Vector

We use the unbalanced turn analysis from "Galileo Maneuver Analysis" by Professor James M. Longuski [1] in order to analyze the individual pulses by the XIPS array. Each daily angular displacement, $\Delta\Psi_{di}$, is divided up into approximately 5484 smaller angles, n, to change the direction of the angular momentum vector over the course of a day. The following equation from the unbalanced turn analysis assumes small angles less than 0.02 radians for each daily angular displacement:

$$\tan\left(\frac{\Delta\Psi_{di}}{n}\right) \approx \frac{\tan\left(\Delta\Psi_{di}\right)}{n} = \frac{F_{di}\, r\Delta t}{I\omega} \quad [1]$$

Equation 5-9

Assuming that the XIPS array will burn for 3.7 seconds, yields approximately 87.5 percent effectiveness as shown from Figure 5-4.

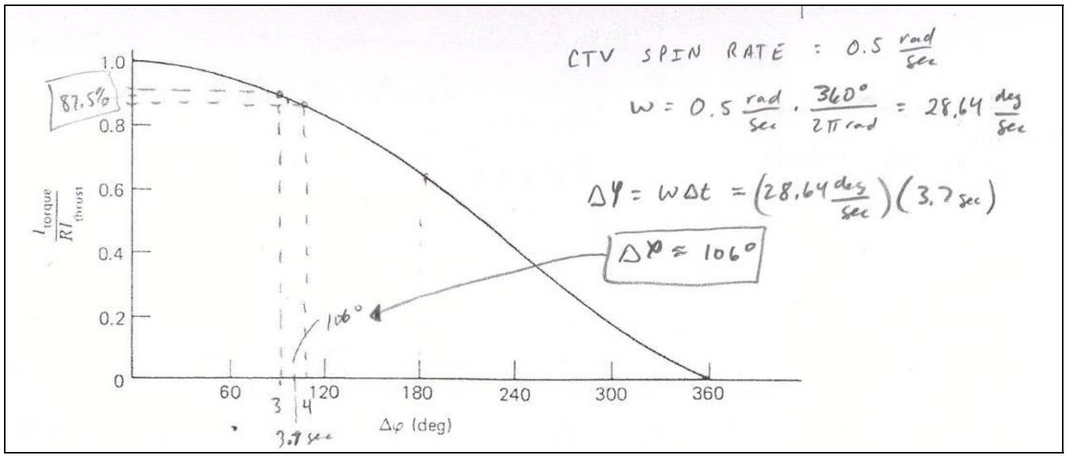


Figure 5-4 Thrust Effectiveness in Momentum Precession [2]

The first day, for example, requires a certain force for each of the 5484 pulses, n. The force is found by rearranging Equation 5-9 and substituting in the known values, including the effectiveness, ξ:

$$F_{d1} = \frac{I\omega\xi}{r\Delta t\, n}\tan\left(\Delta\Psi_{d1}\right)$$

$$= \frac{(1.6\times10^{8}\ \text{kg-m}^{2})(0.50\,\tfrac{\text{rad}}{\text{sec}})(0.875)}{(39.4\ \text{m})(3.7\,\text{sec})(5484\,\text{pulses})}\tan\left(1.03^{\circ}\right)$$

$$= 1.56\,\tfrac{\text{N}}{\text{pulse}}$$

Equation 5-10

The result of Equation 5-10 shows the force/pulse required to change the angular momentum vector 1.03°. In other words, for the first daily angular displacement, $\Delta\Psi_{d1}$ (1.03°), the engines pulse for 3.7 seconds, producing a total force of 1.56 Newtons each pulse, 5484 times per day. Figure 5-5 illustrates the required force for each of the 5484 daily pulses.

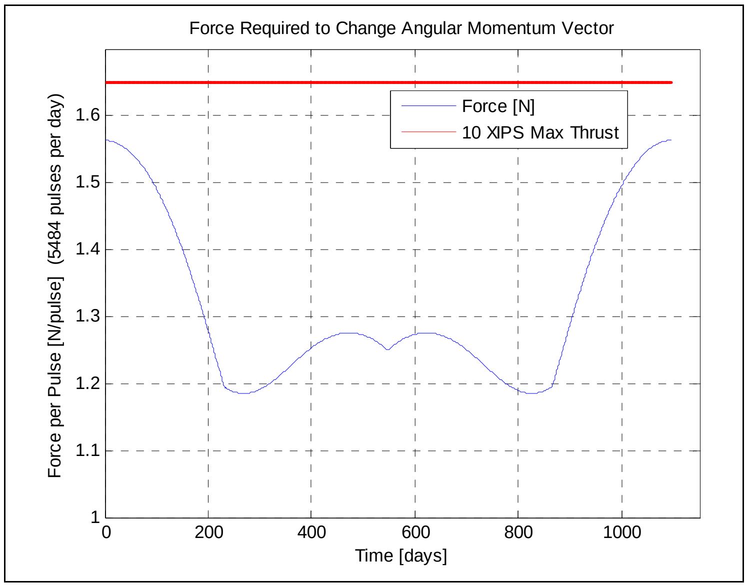


Figure 5-5 Force per Pulse [N/pulse] of each of the 5484 Daily Pulses Required to Change Angular Momentum Vector

Almost all 10 XIPS ion thrusters (0.165 N each) are operating each pulse. Figure 5-5 also shows that all pulses are below the maximum thrust produced by the 10 XIPS array. In order to keep constant communications, we operate the XIPS ion thrusters approximately 25 percent of the total mission duration or about 6600 hrs. 6600 hours of operation is feasible with current XIPS technology [3]. The analysis presented is based on one array of 10 XIPS thrusters. The CTV has two arrays allocated for pointing the angular momentum vector. An extra array is supplied in case of failure of the first; however the two arrays can operate simultaneously if needed.

5.2.2 References

[1] Longuski, James M. "Galileo Maneuver Analysis" AAS/AIAA Astrodynamics Specialist Conference, Lake Tahoe, Nevada (1981): pg 3.

[2] Kaplan, Marshall H., "Modern Spacecraft Dynamics & Control." John Wiley & Sons, 1976.

[3] Xenon Ion Propulsion. Oct 2000. The Boeing Company. 5 Apr. 2005 <http://www.boeing.com/defense-space/space/bss/factsheets/xips/xips.html>

5.2.3 Pointing Method Computer Code (Free-Return Trajectory)

The purpose of this computer code is to obtain the fuel required to "point" the angular momentum vector at Earth for the final "Dragster" design. The code also provides a visual representation of the pointing vector.

Inputs: Keplarian orbital elements for Earth, Mars, and the free-return trajectory.

Outputs: Total mission angular displacement, total propellant mass, and plots of the daily fuel, force/pulse, and angle change throughout the mission duration.

```
% Pete Browning
% Ray Wright
% AAE 450
% Change in H vector


clear all
close all
clc

chg_ang = 0;

% Spacing is one degree or about one day
% -------------------------------------------
thetas_e = linspace(0,360*3,1095)*pi/180;
thetas_t = linspace(0,360*2,1095)*pi/180;


% Orbital Elements for Earth
% -------------------------------------------
a_e = 1.49597807e8;
e_e = 0;
r_e = (a_e*(1-e_e^2))./(1 + e_e*cos(thetas_e));
x_e = r_e.*cos(thetas_e - (chg_ang*pi/180));
y_e = r_e.*sin(thetas_e - (chg_ang*pi/180));
Per_e = 2*pi*sqrt(a_e^3/1.327e11);


% Orbital Elements for Mars
% -------------------------------------------
a_m = 2.2792e8;
e_m = 0.09341233;
r_m = (a_m*(1-e_m^2))./(1 + e_m*cos(thetas_e));
x_m = r_m.*cos(thetas_e);
y_m = r_m.*sin(thetas_e);


% Orbital Elements for Transfer Ellipse
% -------------------------------------------
Per = 1.5*365*24*60*60;  % Period in seconds
a_t = ((Per/2/pi)^2*1.327e11)^(1/3);
e_t = 1-(a_e/a_t);
r_t = (a_t*(1-e_t^2))./(1 + e_t*cos(thetas_t));
x_t = r_t.*cos(thetas_t - (chg_ang*pi/180));
y_t = r_t.*sin(thetas_t - (chg_ang*pi/180));
z_t = r_t.*sin(40*pi/180);
z_t = z_t-z_t(1);
```

```
[row] = find(z_t == max(z_t));
inclin = atan(z_t(row(1))/(x_e(1)-x_t(row(1))))*180/pi;


% Change in positions
% -------------------------------------------
x_v = abs(x_e(2:(end-1)) - x_t(2:(end-1)));
y_v = abs(y_e(2:(end-1)) - y_t(2:(end-1)));


% Calculate Change in Angular Momentum Vector
% -------------------------------------------
ang_v = atan(y_v./x_v);
abs_ang_v = abs(ang_v);
abs_ang_v_z = asin(z_t(2:(end-1))./sqrt(x_v.^2 + y_v.^2));

for n = 1:length(abs_ang_v)-1
delta_ang_v(n) = abs(abs_ang_v(n) - abs_ang_v(n+1));
delta_ang_z(n) = abs(abs_ang_v_z(n) - abs_ang_v_z(n+1));
delta_ang(n) = delta_ang_v(n) + delta_ang_z(n);
end

sum_ang = sum(delta_ang)

num_del = 0;
for j = 1:length(delta_ang)-1
    if abs(delta_ang(j) - delta_ang(j+1)) > 0.0005
        delta_ang(j+1) = (delta_ang(j) + delta_ang(j+2))/2;
        num_del = num_del + 1;
    end
end

% Load Inertia Matrix Created by Brenden Eash
% ------------------------------------------------------------------
[I_CTV,CTV_CM,m_dry,m_wet,r_mom] = CTV_imatrix_dragster(1,0);
I_spin = I_CTV(3,3);
w_spin = sqrt(9.8/r_mom(1));
r_mom = r_mom(1);

% Calculate Total Force and Total Propellant Mass Per Day
% --------------------------------------------------------
for cnt = 1:length(delta_ang)
    num_pulses = 6855*(4/5);
    delh_h_mag_check(cnt) = tan(delta_ang(cnt)/num_pulses);
    angle_change(cnt) = atan(delh_h_mag_check(cnt))*180/pi;
    F_1rev(cnt) = (delh_h_mag_check(cnt))*(I_spin*w_spin)/(r_mom*((3.7/.875)));
    F_day(cnt) = F_1rev(cnt)*num_pulses;
    mass_prop_1rev = (delh_h_mag_check(cnt))*(I_spin*w_spin)/(r_mom*3800*9.8);
    mass_prop(cnt) = mass_prop_1rev*num_pulses;
end
delta_ang_deg = delta_ang*180/pi;
sum_prop = sum(mass_prop)


% Plot Results
% --------------------------------------------------------
figure
plot3(x_e,y_e,y_e*0,'b',x_t,y_t,z_t,'g',0,0,0,'y*',x_t(200),y_t(200),z_t(200),'go',x_e(200),y_e(200),0,'r^')%,x_m,y_m,'r')
legend('Earth Orbit','Transfer Orbit','Sun','CTV Locations','Earth Locations')
xlabel('Distance from Center of Sun [km]')
ylabel('Distance from Center of Sun [km]')
title('CTV Orbit')
grid on
axis equal
hold on

% plot(x_e(107),y_e(107),'k*',x_t(107),y_t(107),'k*')
```

```
for n = [200:100:300]
plot3(x_t(n),y_t(n),z_t(n),'go',x_e(n),y_e(n),0,'r^')

plot3(linspace(x_e(n),x_t(n),2^10/4),((y_t(n) - y_e(n))/(x_t(n) - x_e(n)))*...
    (linspace(x_e(n),x_t(n),2^10/4)) + y_e(n) - ...
    ((y_t(n) - y_e(n))/(x_t(n) - x_e(n)))*x_e(n),linspace(0*x_e(n),z_t(n),2^10/4),'r-')
end
axis equal

% sum_ang = 15.7041271332768

figure
plot(linspace(1,365*3,length(F_1rev)),F_1rev)
xlabel('Time [days]')
ylabel('Force per Pulse [N/pulse]  (5484 pulses per day)')
title('Force Required to Change Angular Momentum Vector')
hold on
plot(linspace(1,365*3,length(F_1rev)),1.65,'r')
legend('Force [N]','10 XIPS Max Thrust')
axis([0 1150 1.0 1.7])
grid on

figure
plot(linspace(1,365*3,length(mass_prop)),mass_prop)
text(360,max(mass_prop)-.06,num2str(ceil(sum(mass_prop))))
text(440,max(mass_prop)-.06,' kg propellant used')
xlabel('Time [days]')
ylabel('Daily Propellant Usage [kg]')
title('Total Propellant Required to Change Angular Momentum Vector')
grid on

figure
plot(linspace(1,365*3,length(delta_ang)),delta_ang*180/pi)
xlabel('Time [days]')
ylabel('Daily Change in Angle (\Delta\Psi_d_i) [deg]')
title('Total Daily Angle Change Required to Change Angular Momentum Vector')
grid on
```

5.2.4 Pointing Method Computer Code (Nominal Trajectory)

The purpose of this computer code is to obtain the fuel required to “point” the angular momentum vector at Earth for the final “Dragster” design, and to compare the results with the free-return trajectory code. The code results in lower fuel cost; however the free-return trajectory must be an option. Thus the analysis is designed around the free-return case.

Inputs: Keplarian orbital elements for Earth, Mars, and the free-return trajectory.

Outputs: Total mission angular displacement, total propellant mass, and plots of the daily fuel, force/pulse, and angle change throughout the mission duration.

```
% Pete Browning
% Ray Wright
% AAE 450
% Change in H vector NOMINAL TRAJECTORY (Non-Free Return)
```

```
clear all
close all


chg_ang = -175;

% Spacing is one degree or one day
Total_TOF = 968;    % Days
out_theta_star = 126.7039;      % Degrees
out_TOF = 151;          % Days
stay_mars = 606;    % Days
in_theta_star = 317+(126.7039);        % Degrees
in_TOF = 211;       % Days
earth_arrive_theta_star =(355.0398 - 203.0344) + in_theta_star;


thetas_e = linspace(0,360*(Total_TOF/365),Total_TOF)*pi/180;
% thetas_t = linspace(0,out_theta_star,out_TOF)*pi/180;
% thetas_t = [thetas_t,linspace(out_theta_star,out_theta_star,stay_mars)*pi/180];
% thetas_t = [thetas_t,linspace(out_theta_star,in_theta_star,in_TOF)*pi/180];


% Orbital Elements for Earth
% -------------------------------------------
a_e = 1.49597807e8;
e_e = 0;
r_e = (a_e*(1-e_e^2))./(1 + e_e*cos(thetas_e));
x_e = r_e.*cos(thetas_e - (chg_ang*pi/180));
y_e = r_e.*sin(thetas_e - (chg_ang*pi/180));
Per_e = 2*pi*sqrt(a_e^3/1.327e11);


% Orbital Elements for Mars
% -------------------------------------------
a_m = 2.2792e8;
e_m = 0.09341233;
r_m = (a_m*(1-e_m^2))./(1 + e_m*cos(thetas_e));
x_m = r_m.*cos(thetas_e);
y_m = r_m.*sin(thetas_e);


% Orbital Elements for Transfer Ellipse
% -------------------------------------------
for Per = [out_TOF,stay_mars,in_TOF]*24*60*60 ;  % Period in seconds
    if Per == out_TOF*24*60*60
        Per = 1.5*365*24*60*60;  % Period in seconds
        a_t = ((Per/2/pi)^2*1.327e11)^(1/3);
        e_t = 1-(a_e/a_t);
        thetas_t1 = linspace(0,out_theta_star,out_TOF)*pi/180;
        r_t1 = (a_t*(1-e_t^2))./(1 + e_t*cos(thetas_t1));
    end
    if Per == stay_mars*24*60*60
        a_t = a_m;
        e_t = e_m;
        thetas_t2 = linspace(out_theta_star,in_theta_star,stay_mars)*pi/180;
        r_t2 = (a_t*(1-e_t^2))./(1 + e_t*cos(thetas_t2 - (chg_ang*pi/180)));
    end
    if Per == in_TOF*24*60*60
        a_t = 193080478.7762;
        e_t = 0.228161;
        thetas_t3 = linspace(in_theta_star,earth_arrive_theta_star,in_TOF)*pi/180;
        r_t3 = (a_t*(1-e_t^2))./(1 + e_t*cos(thetas_t3 - (295*pi/180)));
    end

end

thetas_t = [thetas_t1,thetas_t2,thetas_t3];
```

```
r_t = [r_t1,r_t2,r_t3];

x_t = r_t.*cos(thetas_t - (chg_ang*pi/180));
y_t = r_t.*sin(thetas_t - (chg_ang*pi/180));

x_v = abs(x_e(2:(end-1)) - x_t(2:(end-1)));
y_v = abs(y_e(2:(end-1)) - y_t(2:(end-1)));

%%% If you want to neglect terms near Earth use the following:
% % x_v = abs(x_e(6:(end-5)) - x_t(6:(end-5)));
% % y_v = abs(y_e(6:(end-5)) - y_t(6:(end-5)));

ang_v = atan(y_v./x_v);
abs_ang_v = abs(ang_v);

for n = 1:length(abs_ang_v)-1
delta_ang(n) = abs(abs_ang_v(n) - abs_ang_v(n+1));
end

sum_ang = sum(delta_ang)

num_del = 0;
for j = 1:length(delta_ang)-1
   if abs(delta_ang(j) - delta_ang(j+1)) > 0.0005 && j~= 964
      delta_ang(j+1) = (delta_ang(j) + delta_ang(j+2))/2;
      num_del = num_del + 1;
   end
end

delta_ang = delta_ang(1:end);

r_e = [x_e;y_e];
r_t = [x_t;y_t];
r_v = r_e - r_t;
r_v = r_v';


[I_spin,w_spin,r_mom] = I_and_w;


for cnt = 1:length(delta_ang)
   if cnt <= out_TOF
      del_h_dir(cnt,1) = r_v(cnt+1,1) - r_v(cnt,1);
      del_h_dir(cnt,2) = r_v(cnt+1,2) - r_v(cnt,2);

      % delh_h_dir(cnt) = norm(del_h_dir(cnt,:))./norm(r_v((2:end),:));
      delh_h_mag_check(cnt) = norm([1;1]-((r_e(:,cnt) - r_t(:,cnt))./(r_e(:,cnt+1) - r_t(:,cnt+1))));
      angle_change(cnt) = atan(delh_h_mag_check(cnt))*180/pi;

      delh_h_mag_check(cnt) = tan(delta_ang(cnt));

      F(cnt) = (delh_h_mag_check(cnt))*(I_spin(1)*w_spin(1))/(r_mom(1)*24*60*60/(2*pi/w_spin(1))*5.8*(.80));
      mass_prop(cnt) = (delh_h_mag_check(cnt))*(I_spin(1)*w_spin(1))/(r_mom(1)*3800*9.8);
      %delh_h_dir(1,cnt) = del_h_dir(1,cnt)/r_v(1
   end
   if cnt > out_TOF && cnt <= stay_mars+out_TOF
      del_h_dir(cnt,1) = r_v(cnt+1,1) - r_v(cnt,1);
      del_h_dir(cnt,2) = r_v(cnt+1,2) - r_v(cnt,2);

      % delh_h_dir(cnt) = norm(del_h_dir(cnt,:))./norm(r_v((2:end),:));
      delh_h_mag_check(cnt) = norm([1;1]-((r_e(:,cnt) - r_t(:,cnt))./(r_e(:,cnt+1) - r_t(:,cnt+1))));
      angle_change(cnt) = atan(delh_h_mag_check(cnt))*180/pi;

      delh_h_mag_check(cnt) = tan(delta_ang(cnt));

      F(cnt) = 0;
      mass_prop(cnt) = 0;
      %delh_h_dir(1,cnt) = del_h_dir(1,cnt)/r_v(1
   end
```

```
    if cnt > stay_mars+out_TOF
        del_h_dir(cnt,1) = r_v(cnt+1,1) - r_v(cnt,1);
        del_h_dir(cnt,2) = r_v(cnt+1,2) - r_v(cnt,2);

        % delh_h_dir(cnt) = norm(del_h_dir(cnt,:))./norm(r_v((2:end),:));
        delh_h_mag_check(cnt) = norm([1;1]-((r_e(:,cnt) - r_t(:,cnt))./(r_e(:,cnt+1) - r_t(:,cnt+1))));
        angle_change(cnt) = atan(delh_h_mag_check(cnt))*180/pi;

        delh_h_mag_check(cnt) = tan(delta_ang(cnt));

        F(cnt) = (delh_h_mag_check(cnt))*(I_spin(2)*w_spin(2))/(r_mom(2)*24*60*60/(2*pi/w_spin(2))*5.5*(.80));
        mass_prop(cnt) = (delh_h_mag_check(cnt))*(I_spin(2)*w_spin(2))/(r_mom(2)*3800*9.8);
        %delh_h_dir(1,cnt) = del_h_dir(1,cnt)/r_v(1
    end
end

% F = [F(1:(22)),F(24:(end))];
% mass_prop = [mass_prop(1:(23)),mass_prop(23:(end))];
% angle_change = [angle_change(1:(23)),angle_change(23:(end))];

delta_ang_deg = delta_ang*180/pi;
sum_mass = sum(mass_prop)

r_vector = sqrt(r_v(1,:).^2 + r_v(2,:).^2);



figure
plot(x_e,y_e,'b',x_t,y_t,'g',0,0,'y*')%,x_m,y_m,'r')
legend('Earth Orbit','Transfer Orbit','Sun')
xlabel('Distance from Center of Sun [km]')
ylabel('Distance from Center of Sun [km]')
title('CTV Orbit with Mars Arrival')
grid on
axis equal
hold on
% plot(x_e(95),y_e(95),'k*',x_t(95),y_t(95),'k*')
plot(x_e(26),y_e(26),'m*',x_t(26),y_t(26),'m*')
for n = [25:25:150]
plot(x_t(n),y_t(n),'go',x_e(n),y_e(n),'r^')

plot(linspace(x_e(n),x_t(n),2^10/4),((y_t(n) - y_e(n))/(x_t(n) - x_e(n)))*...
    (linspace(x_e(n),x_t(n),2^10/4)) + y_e(n) - ...
    ((y_t(n) - y_e(n))/(x_t(n) - x_e(n)))*x_e(n),'r-')
end
% plot(x_e(437),y_e(437),'y*',x_t(437),y_t(437),'y*')
% plot(x_e(660),y_e(660),'b*',x_t(660),y_t(660),'b*')
% plot(x_e(898),y_e(898),'r*',x_t(898),y_t(898),'r*')
% plot(x_e(1001),y_e(1001),'c*',x_t(1001),y_t(1001),'c*')

% sum_ang = 15.7041271332768

% ct = 1;
% figure
% for xn = 1:length(y_e)/5:length(y_e)
% plot(linspace(0,x_e(xn),10),linspace(0,y_e(xn),10),'b',linspace(0,x_t(xn),10),linspace(0,y_t(xn),10),'g')
% hold on
% xnc(ct) = xn
% ct = ct + 1;
% end


figure
plot(linspace(1,(length(F)),length(F)),F)
xlabel('Time [days]')
ylabel('Daily Force [N]')
title('Total Force Required to Change Angular Momentum Vector with Mars Arrival')
hold on
plot(linspace(1,365*3,length(F)),1.65,'r')
```

```
legend('Force [N]','10 XIPS Max Thrust')
grid on

figure
plot(linspace(1,(length(F)),length(mass_prop)),mass_prop)
text(360,max(mass_prop)-.06,num2str(ceil(sum(mass_prop))))
text(440,max(mass_prop)-.06,' kg propellant used')
xlabel('Time [days]')
ylabel('Daily Propellant Usage [kg]')
title('Total Propellant Required to Change Angular Momentum Vector with Mars Arrival')
grid on

figure
plot(linspace(1,(length(F)),length(delta_ang)),delta_ang*180/pi)
xlabel('Time [days]')
ylabel('Daily Change in Angle [deg]')
title('Total Daily Angle Change Required to Change Angular Momentum Vector with Mars Arrival')
grid on
```

5.3 Preliminary Spin Analysis Appendix

Author: Pete Browning

5.3.1 Preliminary CTV Spin Analysis

In the early stages of design, we estimated the force and fuel cost associated with spinning the CTV to impose artificial gravity. The initial design of the CTV is shown in Figure 5-6 below.

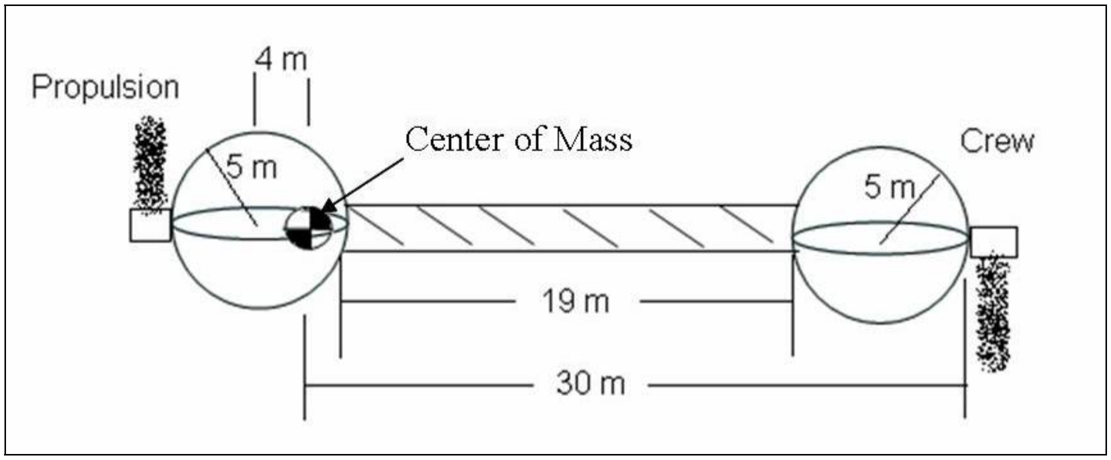


Figure 5-6 Initial "Flyswatter" Design

We achieve artificial gravity when the centripetal acceleration, a_c, at the Crew Compartment is equal to 9.81 m/s^2. Centripetal acceleration is the rate of change of the tangential velocity and is related to the tangential speed (v) and angular velocity (ω):

$$a_c = \frac{v^2}{r} = \omega^2 r$$

Equation 5-11

where r is the moment arm, or radius from the center of mass to the outer Crew Compartment wall. We use the Equation 5-11, solve for angular velocity, and substitute in know values (See Figure 5-6 for 'r' value):

$$\omega = \sqrt{\frac{a_c}{r}} = \sqrt{\frac{9.81\frac{m}{sec^2}}{30\,m}} = 0.57\,\frac{rad}{sec} = 5.5\,\frac{rev}{min}$$

Equation 5-12

According to the result of Equation 5-12 the crew experiences Earth gravity when the CTV is spinning at 5.5 rev/min. Euler's Equations of Motion are used to determine the force required to spin the CTV:

$$M_x = I_x\dot{\omega}_x + \left(I_z - I_y\right)\omega_y\omega_z$$
$$M_y = I_y\dot{\omega}_y + \left(I_x - I_y\right)\omega_y\omega_x$$
$$M_z = I_z\dot{\omega}_z + \left(I_y - I_x\right)\omega_x\omega_y$$

Equation 5-13

where M is the moment, I is the inertia $\dot{\omega}$ is the angular acceleration, and ω is the angular velocity (subscripts indicate axis). We assume the CTV spins about the $\hat{z}$ axis and there are no angular velocity components in the $\hat{x}$ and $\hat{y}$ directions. Therefore Euler's EOMs reduce to only one equation:

$$M_z = I_z\dot{\omega}_z + \left(I_y - I_x\right)\cancel{\omega_x}\cancel{\omega_y}$$

$$M_z = I_z\dot{\omega}_z$$

Equation 5-14

Next, the result of Equation 5-14 is integrated with respect to time (t) to find the relationship between angular velocity, time, and moment.

$$\int M_z\,dt = \int I_z\dot{\omega}_z\,dt$$

$$M_z t = I_z\omega_z$$

$$M_z = \frac{I_z\omega_z}{t}$$

Equation 5-15

The moment is composed of the cross product of the moment arm radius vector ($\vec{r}$) and the force vector ($\vec{F}$) which are perpendicular ($M = \vec{r}\times\vec{F} = rF$), therefore

$$F\,r = \frac{I_z\omega_z}{t}$$

Equation 5-16

We wrote a computer code (See Section 5.3.3) to calculate the force required to spin the CTV by varying the moment arm (r) and the time (t). The minimum and maximum force values are shown in Table 5-2 for various times.

Table 5-2 Force Necessary for Artificial Gravity for Various Times and Moment Arms (r)

Time	10 min	30 min	1 hr	1 day	7 days

Min. Force [N] (r = 30 m)	150.87	50.29	25.14	1.05	0.15
Max. Force [N] (r = 9 m)	502.89	167.63	83.81	3.49	0.50

The first row of Table 5-2 contains the force values for various times when the force is applied 30 meters from the center of mass, or at the Crew Compartment. The second row of Table 5-2 contains the force values for various times when the force is applied 9 meters from the center of mass, or at the Propulsion Compartment. From this analysis we choose to place the thrusters on the Crew Compartment to create artificial gravity.

The fuel cost is calculated using the equation for specific impulse:

$$I_{sp} = \frac{F}{\dot{m}\, g_o}$$

Equation 5-17

where I_{sp} is the specific impulse, F is the force, $\dot{m}$ is the mass flow rate, and g_o is the acceleration due to gravity at Earth's surface = 9.81 (m/s^2). The amount of propellant used per unit of time is equal to the mass flow rate: $\frac{\Delta m_{propellant}}{\Delta t} = \dot{m}$, which is substituted into Equation 5-17 and rearranged:

$$\Delta m_{propellant} = \frac{F\,\Delta t}{I_{sp}\, g_o}$$

Equation 5-18

For the initial "Flyswatter" design, we calculated the fuel cost to be 30 kg using an engine with an I_{sp} of 300 sec. The cost is very low; therefore we concluded artificial gravity was possible with minimal propellant.

Even though the design changed drastically from the initial "Flyswatter" concept, we understand that creating artificial gravity is feasible and inexpensive fuel-wise using thrusters.

5.3.2 Reference

[1] Longuski, James M. "AAE 340 Dynamics and Vibrations Lectures." Purdue University. Fall 2003.

5.3.3 Preliminary Spin-up Analysis Computer Code

The purpose of this computer code is to form an initial analysis on the force required to create artificial gravity on the preliminary "Flyswatter" design. The code also provides initial inertia calculations which are expanded as the design improves.

Inputs: Mass of crew quarters, mass of propellant pod, mass of truss, distance from the center of mass to the center of mass of the crew quarters, distance from the center of mass to the center of mass of the power pod, and length of the truss.

Outputs: Force required for artificial gravity for various times.

```
% Pete Browning
% AAE 450
% "Flyswatter" Spin-up Analysis


clear all
close all
clc

M_crew = 13793;   % One side - kg
M_prop = 386027;
M_boom = 25000;
M_total = M_crew + M_prop;

r_crew = 25;       % distance from CM Crew Vehicle to CM crew - meters
r_prop = 4;        % distance from CM Crew Vehicle to CM prop - meters
l_boom = 19;   % boom length
r=5;        % diameter of sphere - meters

I_crew = [2/5*M_crew*r^2 0 0; 0 2/5*M_crew*r^2 0; 0 0 2/5*M_crew*r^2];
I1=I_crew + [0 0 0; 0 r_crew^2 0; 0 0 r_crew^2];

I_prop = [2/5*M_prop*r^2 0 0; 0 2/5*M_prop*r^2 0; 0 0 2/5*M_prop*r^2];
I2=I_prop + [0 0 0; 0 r_prop^2 0; 0 0 r_prop^2];

I_boom = [0 0 0; 0 M_boom*l_boom^2/12 0; 0 0 M_boom*l_boom^2/12];
I3=I_boom + [0 0 0; 0 l_boom^2 0; 0 0 l_boom^2];


I = I1 + I2 + I3;
I_z = I(3,3);

w_z = 5.457877*2*pi/60; % final spin rate [rad/sec]

n = 0;
for time=[600,1800,3600,86400,172800,259200,345600,432000,518400,604800];   % 1 day - seconds
n = n + 1;
M_z = w_z*I_z./time ;
F_crew = M_z./(r_crew + r);

time_disp = time/3600/24;
time_day = time/3600;
g = 1/time;   %change in acceleration (g's) per second
```

```
M_z = w_z*I_z./time;
F_prop = M_z./(r_prop + r);
%  F_prop_total = F_prop*

M_z=w_z*I_z./time;
for F_little = M_z./(r_crew + r)*.50
 F_big = (M_z - ((r_crew + r)*F_little))/(r_prop + r);
 F_total = F_little + F_big;
%      plot(F_little,F_big,'bo',F_little,F_total,'go')
%      xlabel('Little')
%      ylabel('BIG')
%
%      hold on
end

F_both = F_little + F_big;
Mz = (F_little*(r_crew + r))+(F_big*(r_prop + r));
M_z_(n) = M_z;

% % fprintf('                    |Crew-Located Thruster|Propusion-Located Thruster|Thrusters On Each End|\n')
% % fprintf('--------------------|---------------------|--------------------------|---------------------|\n')
% % %  fprintf('g loads [g"s/hr]    |       %.2f        |          %.2f           |        %.2f         |\n',g,g,g)
% % fprintf('spin-up time [days] |       %.2f        |          %.2f           |        %.2f         |\n',time_disp,time_disp,time_disp)
% % fprintf('Force [N]           |      %4.2f        |         %4.2f           |       %4.2f         |\n\n',F_crew,F_prop,F_both)
% %
t = 0:60:time;
gs = 0:g*60:1;
plot(t,gs)
hold on


time_disp_(n) = time_disp;
F_crew_(n) = F_crew;
F_prop_(n) = F_prop;
F_both_(n) = F_both;

end

time_disp_
F_crew_
F_prop_
F_both_
M_z_
```

5.3.4 Modified Spin-up Analysis Computer Code

The purpose of this computer code is to obtain the force requirements to create artificial gravity for the updated preliminary “Cross” design. The code also provides more detailed inertia calculations.

Inputs: Mass and dimensions of the crew quarters, propellant pod, trusses, center section, fuel, ARV, and MLV.

Outputs: Location of center of mass, the inertia matrix, and updated force values required for artificial gravity for various times.

```
% Pete Browning
% AAE 450
% Inertia Model of CTV
% Updated Spin-up Force Values


clear all
close all
clc

M_crew = 74800;     % Crew Cabin [kg]
M_mid = 40200;      % Middle Structure Mass
M_fuel = 125000;    % Fuel Mass
M_middle = M_mid + M_fuel; % Total Middle Mass
M_power = 10000;    % Power Mass
M_boom = 30000/2;     % One Boom
M_ARV = 7500;       % ARV Mass
M_MLV = 30300;      % MLV Mass
M_CTV_wet = M_crew + M_middle + M_power + 2*M_boom + M_ARV + M_MLV;

r_crew = 7.5/2;  % Radius of Crew Cabin
h_crew = 5;      % Height of Crew Cabin
% Inertia is Right Circular Cylinder
vol_crew = (pi*r_crew^2*h_crew);
Icrew = [(1/2)*M_crew*r_crew^2 0 0;0 (M_crew/12)*(3*r_crew^2 + h_crew^2) 0;0 0 (M_crew/12)*(3*r_crew^2 + h_crew^2)];

r_power = 7.5/2;  % Radius of Power Pod
h_power = 5;      % Height of Power Pod
% Inertia is Right Circular Cylinder
vol_power = (pi*r_power^2*h_power);
Ipower = [(1/2)*M_power*r_power^2 0 0;0 (M_power/12)*(3*r_power^2 + h_power^2) 0;0 0 (M_power/12)*(3*r_power^2 + h_power^2)];

r_boom = 1/2; % Radius of Boom [m]
h_boom_crew = 21;  % Height of Boom [m]
h_boom_power = 21;
% % Inertia is Thin Circular Cylindrical Shell
% Iboom = [M_boom*r_boom^2 0 0;0 (M_boom/12)*(6*r_boom^2 + h_boom^2) 0;0 0 (M_boom/12)*(6*r_boom^2 + h_boom^2)];
% Inertia is Right Circular Cylinder
Iboom_crew = [(1/2)*M_boom*r_boom^2 0 0;0 (M_boom/12)*(3*r_boom^2 + h_boom_crew^2) 0;0 0 (M_boom/12)*(3*r_boom^2 +
h_boom_crew^2)];
Iboom_power = [(1/2)*M_boom*r_boom^2 0 0;0 (M_boom/12)*(3*r_boom^2 + h_boom_power^2) 0;0 0 (M_boom/12)*(3*r_boom^2 +
h_boom_power^2)];

r_ARV = 4/2;  % Radius of ARV
h_ARV = 4;    % Height of ARV [m]
% Inertia is Right Circular Cone
vol_ARV = (1/3)*pi*r_ARV^2*h_ARV;
% ARV attached opposite to where MLV docks
IARV = [(3*M_ARV/80)*(4*r_ARV^2 + h_ARV^2) 0 0;0 (3*M_ARV/80)*(4*r_ARV^2 + h_ARV^2) 0;0 0 (3/10)*(M_ARV*r_ARV^2)];
y_arv = 0;
% % ARV attached out of plane
% IARV = [(3*M_ARV/80)*(4*r_ARV^2 + h_ARV^2) 0 0;0 (3/10)*(M_ARV*r_ARV^2) 0;0 0 (3*M_ARV/80)*(4*r_ARV^2 + h_ARV^2)];
% y_arv = r_crew + h_ARV/4;

r_MLV = 10/2;  % Radius of MLV [m]
h_MLV = 20;    % Height of MLV [m]
% Inertia is Right Circular Cylinder
vol_MLV = pi*r_MLV^2*h_MLV;
IMLV = [(M_MLV/12)*(3*r_MLV^2 + h_MLV^2) 0 0;0 (M_MLV/12)*(3*r_MLV^2 + h_MLV^2) 0;0 0 (1/2)*(M_MLV*r_MLV^2)];

r_middle = 8/2; % Radius of Middle Section
h_middle = 20;
% Inertia is Right Circular Cylinder
Imiddle = [(M_middle/12)*(3*r_middle^2 + h_middle^2) 0 0;0 (M_middle/12)*(3*r_middle^2 + h_middle^2) 0;0 0 (1/2)*(M_middle*r_middle^2)];

% Coordinate System is located in the center of the midddle section back by
% the main engine

% Find Center of Mass
% [X Y Z]

boom_pos = 10;

power_cm = [-(r_middle + h_boom_power + h_power/2), 0, -(h_MLV + boom_pos)];
left_boom_cm = [-(r_middle + h_boom_power/2), 0 , -(h_MLV + boom_pos)];
middle_cm = [0,0,-(h_MLV + boom_pos)];
MLV_cm = [0, 0, -h_MLV/2];
right_boom_cm = [r_middle + h_boom_crew/2, 0 , -(h_MLV + boom_pos)];
crew_cm = [r_middle + h_boom_crew + h_crew/2, 0, -(h_MLV + boom_pos)];
ARV_cm = [r_middle + h_boom_crew + h_crew/4, y_arv, -(h_MLV + boom_pos + r_crew + h_ARV/4)];
CTV_cm = (M_power*power_cm + M_boom*left_boom_cm + M_middle*middle_cm + M_MLV*MLV_cm + ...
    M_boom*right_boom_cm + M_crew*crew_cm + M_ARV*ARV_cm)/(M_CTV_wet);
CTV_cm_disp = CTV_cm + [0 0 (h_MLV + boom_pos)]

opposite = CTV_cm_disp(3) + h_middle/2;
gimbal_ang = 90 - atan(opposite/CTV_cm(1))*180/pi;




% Parallel Axis Theorem (NEED TO CHECK FOR ACCURACY)
d_power = power_cm - CTV_cm;
I_power = Ipower + [M_power*(d_power(2)^2 + d_power(3)^2)  0 0; 0 M_power*(d_power(1)^2 + d_power(3)^2) 0; 0 0 M_power*(d_power(2)^2 +
d_power(1)^2)];

d_left_boom = left_boom_cm - CTV_cm;
I_left_boom = Iboom_power + [M_boom*(d_left_boom(2)^2 + d_left_boom(3)^2) 0 0; 0 M_boom*(d_left_boom(1)^2 + d_left_boom(3)^2) 0; 0 0
M_boom*(d_left_boom(2)^2 + d_left_boom(1)^2)];

d_middle = middle_cm - CTV_cm;
I_middle = Imiddle + [M_middle*(d_middle(2)^2 + d_middle(3)^2) 0 0; 0 M_middle*(d_middle(1)^2 + d_middle(3)^2) 0; 0 0
M_middle*(d_middle(2)^2 + d_middle(1)^2)];

d_MLV = MLV_cm - CTV_cm;
I_MLV = IMLV + [M_MLV*(d_MLV(2)^2 + d_MLV(3)^2) 0 0; 0 M_MLV*(d_MLV(1)^2 + d_MLV(3)^2) 0; 0 0 M_MLV*(d_MLV(2)^2 +
d_MLV(1)^2)];

d_right_boom = right_boom_cm - CTV_cm;
I_right_boom = Iboom_crew + [M_boom*(d_right_boom(2)^2 + d_right_boom(3)^2) 0 0; 0 M_boom*(d_right_boom(1)^2 + d_right_boom(3)^2) 0; 0 0
M_boom*(d_right_boom(2)^2 + d_right_boom(1)^2)];

d_crew = crew_cm - CTV_cm;
I_crew = Icrew + [M_crew*(d_crew(2)^2 + d_crew(3)^2) 0 0; 0 M_crew*(d_crew(1)^2 + d_crew(3)^2) 0; 0 0 M_crew*(d_crew(2)^2 + d_crew(1)^2)];

d_ARV = ARV_cm - CTV_cm;
I_ARV = IARV + [M_ARV*(d_ARV(2)^2 + d_ARV(3)^2) 0 0; 0 M_ARV*(d_ARV(1)^2 + d_ARV(3)^2) 0; 0 0 M_ARV*(d_ARV(2)^2 +
d_ARV(1)^2)];

I_total = I_power + I_left_boom + I_middle + I_MLV + I_right_boom + I_crew + I_ARV




I_z=I_total(3,3);
big_omega=0;
w_z = 5.457877*2*pi/60; % final spin rate [rad/sec]

ct = 1;
for time=[86400,172800];   %1 & 2 days - seconds
 M_z_T = w_z*I_z./time;
 M_z_P = w_z*1.012e8/time;
 F_crew(ct)=M_z_T./(d_crew(1) + h_crew/2 + CTV_cm_disp(1));
 F_crew_P(ct)=M_z_P./(d_crew(1) + h_crew/2 + CTV_cm_disp(1));

 fprintf('Force: %.2f [N]    Time: %.2d [days]\n',F_crew(ct),time/60/60/24)
 times(ct) = time;
 ct = ct + 1;
end
```

6 Burke, Caley

6.1 General Earth Launch Vehicle Work

Author: Caley Burke

We initially consider an alternative design from the stacked Earth Launch Vehicle (ELV) configuration given in the report. We base the first design off the Russian Energia. This vehicle is similar to the Space Shuttle, but with the option of having an unmanned payload container replacing the orbiter position. We employ the container option in Figure 3-7: Initial ELV design.

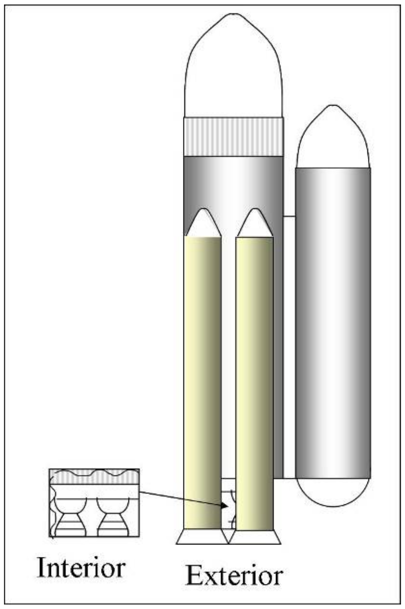


Figure 3-7: Initial ELV design (Based on Energia[1])

We reject this design due to the limitations it puts on the payload. We envision a maximum payload size of a 5.5 m diameter for this design. The payloads of Project Legend require a 10m diameter capability. Due to this incompatibility, we switch the design to the more standard stacked, in-line vehicle with a larger diameter possibility discussed within the report.

References:

[1]"LV Energia with the transportation container". S.P.Korolev Rocket and Space Corporation Energia. 24 Jan. 2005. http://www.energia.ru/english/energia/launchers/vehicle_energia-c.html

Resources:

[1]Isakowitz, Steven J. International Reference Guide to Space Launch Systems. 1991: AIAA, Washington, D.C.

6.2 Solid Motor Stage

Author: Caley Burke

We chose the Solid Rocket Boosters (SRBs) because of their large thrust capability. The mass of the Orbiter plus a maximum payload on the Space Shuttle is 105 metric tons. When considering the loaded Orbiter plus payload as the Shuttle payload, the SRBs routinely lift payloads on the same order of magnitude as necessary for Project Legend.

We also consider the high level of safety the SRBs carry. Since they are normally used in manned flight, the safety rating is necessarily higher than the usual unmanned ratings.

Resources:

[1] Shuttle Reference Manual. NASA. 1998. 5 Feb. 2005.
<http://spaceflight.nasa.gov/shuttle/reference/shutref/verboseindex.html>

[2]Isakowitz, Steven J. International Reference Guide to Space Launch Systems. 1991: AIAA, Washington, D.C.

6.3 Stage I Engines

Author: Caley Burke

We examine several engines before deciding on the RD-180 engines for the Earth Launch Vehicle (ELV): the Space Shuttle Main Engine (SSME), RD-0120, and RS-84. The engines' respective thrust values are as follows: 2,278 kN, 1,961 kN, and 4,902 kN. We reject the SSME and RD-180 engines as their thrust is lower than the RD-180, which has a thrust of 4152 kN. We dismiss the RS-84 engine because, though it has a higher thrust level than the RD-180, it is still in the development process and may not be ready in time to accommodate the Project Legend schedule.

The Two_Stage_Mass_SRB_Estimate MATLAB function estimates the propellant mass for Stage I (this code and explanation can be found in the Vehicle Launch Trajectory Appendix). The oxidizer to fuel ratio of 2.72 divides the total propellant mass into its LOX and RP-1 values.

We base the initial estimate of the propellant mass fraction of 0.93 off of the Saturn V Stage I[1]. With the structural mass for the ELV Stage I, we find the actual propellant mass fraction to be 0.932, which is a nominal ~0.2% error from the starting estimate.

References:

[1]Isakowitz, Steven J. International Reference Guide to Space Launch Systems. 1991: AIAA, Washington, D.C.

Resources:

[1]"Space Products: RD-180". Pratt & Whitney. 30 Jan. 2005. <http://www.pratt-whitney.com/prod_space_rd180.asp>

[2]Humble, Ronald W., Gary N. Henry, and Wiley J. Larson. Space Propulsion Analysis and Design. 1995: McGraw-Hill, New York.

[3] Sutton, George P. and Oscar Biblarz. Rocket propulsion elements. 2001: John Wiley & Sons, New York.

[4]Wade, Mark. Encyclopedia Astronautica. Dec. 2003. Space Wire. 24 Jan. 2005. <http://www.astronautix.com/engines/index.htm>

6.4 Vehicle Launch Trajectory

Author: Caley Burke

6.4.1 Theory

We estimate original propulsion masses and then volumes for the Earth Launch Vehicle (ELV) from the following functions: Two_Stage_Mass_SRB_Estimate; prop_size. We employ the main trajectory code, Matlab file Thrust_code_Trajectory.m, to predict the trajectory of the ELV. The main file calls on the following eight functions: RD180_properties; SSME_properties; thrust_alt; atmosMetric; atmosMetric_rev; SRB_Thrust_Profile_M_dot_2; dragRocket; pva.

The Two_Stage_Mass_SRB_Estimate function finds the propellant masses for Stages I and II. The inputs are the expected propellant mass fractions and the fractions of the total change in velocity each stage is expected to bear, along with the actual Isp values for the stages. The overall mass for each stage is then calculated using Equation 3-19. The propellant masses are then calculated with the propellant mass fraction.

$$m_{inert} = m_{pay} \frac{\left(1 - e^{\frac{\Delta v}{Isp * g_0}}\right)}{\left(e^{\frac{\Delta v}{Isp * g_0}}\left(1-\zeta\right) - 1\right)} \text{ [Ref. 1]}$$

Equation 3-19

The prop_size function finds the volumes of each propellant for each stage. For each stage, we divide the total propellant mass into fuel and oxidizer propellant using the oxidizer to fuel ratio (Equation 3-20). We multiple each mass by the density of its fluid, which come from Ref. 4 & 5.

$$m_{oxidizer} = \frac{m_{prop} * \frac{O}{F}}{\frac{O}{F}+1}, m_{fuel} = \frac{m_{prop}}{\frac{O}{F}+1}$$

Equation 3-20

The Thrust_code_Trajectory code calculates each of the following values at every second of launch: position (spherical coordinates), velocity, acceleration, thrust (F), vehicle mass, local flight-path angle (γ), drag, Mach number (M), and dynamic pressure (q). We employ the forces from Figure 3-8: Force on a Vehicle in Planar Ascent Trajectory (based on Reference 1).

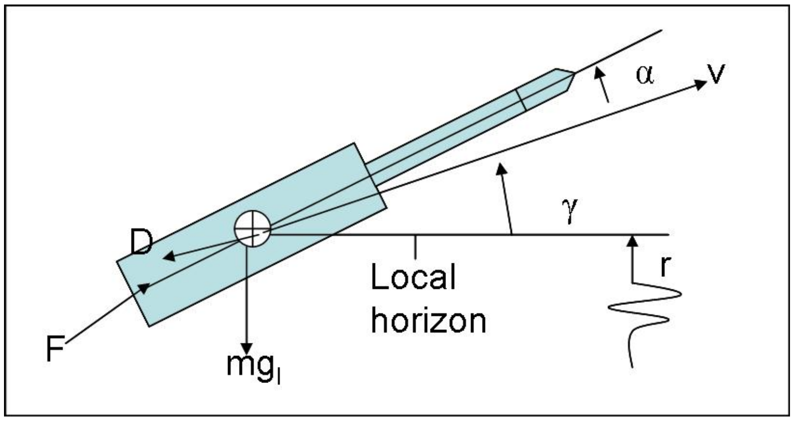


Figure 3-8: Force on a Vehicle in Planar Ascent Trajectory (based on Reference 1)

We find the thrust at any given second by figuring which engines are being used at that point, altering the vacuum thrust appropriately to the pressure at the current altitude, and then multiplying the thrust by any throttle present. The main throttle points are during the maximum dynamic pressure and the end of the Stage I burn. The mass of the vehicle comes from which stages are still attached and subtracting the fuel mass burnt from the current stage(s).

We assume the angle of attack (α) to be zero throughout the entire flight. This allows us to employ the gravity turn, following a launch tower clearance pitch-over of ~2º. The gravity turn finds how the local-flight path angle changes due to differences in the gravity force and the velocity centripetal acceleration of the vehicle (Equation 3-21). In further design, we control the angle of attack for a more optimal trajectory.

$$\frac{d\gamma}{dt} = -\frac{g_l \cos\gamma}{v}(rad/s) \textbf{ [Ref. 1]}$$

Equation 3-21

The main trajectory code generates the following figures: Figure 3-9: ELV Altitude Profile, Figure 3-10: ELV Acceleration Profile, and Figure 3-11: ELV Thrust Profile. The figures demonstrate the payload reaching its appropriate altitude and the propulsion qualities that get it there.

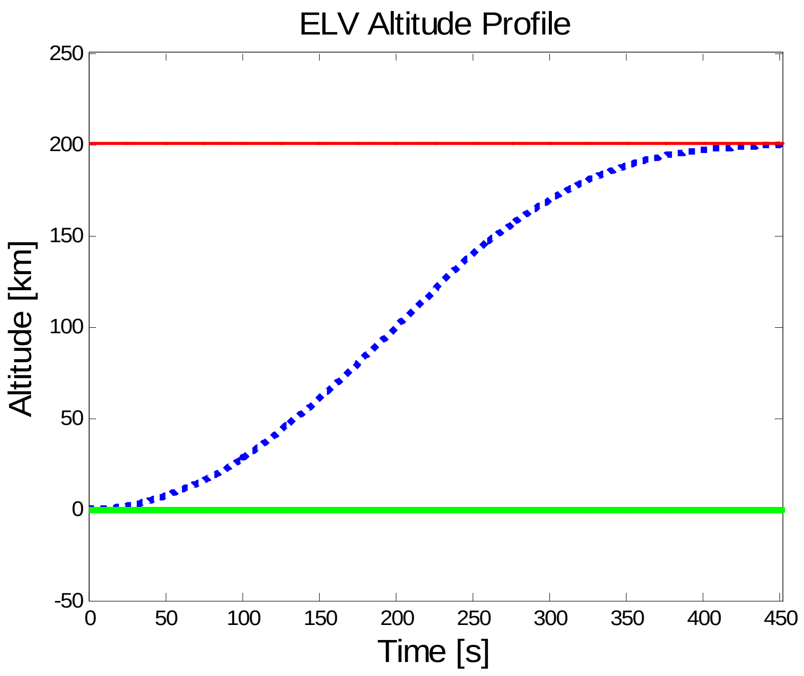


Figure 3-9: ELV Altitude Profile

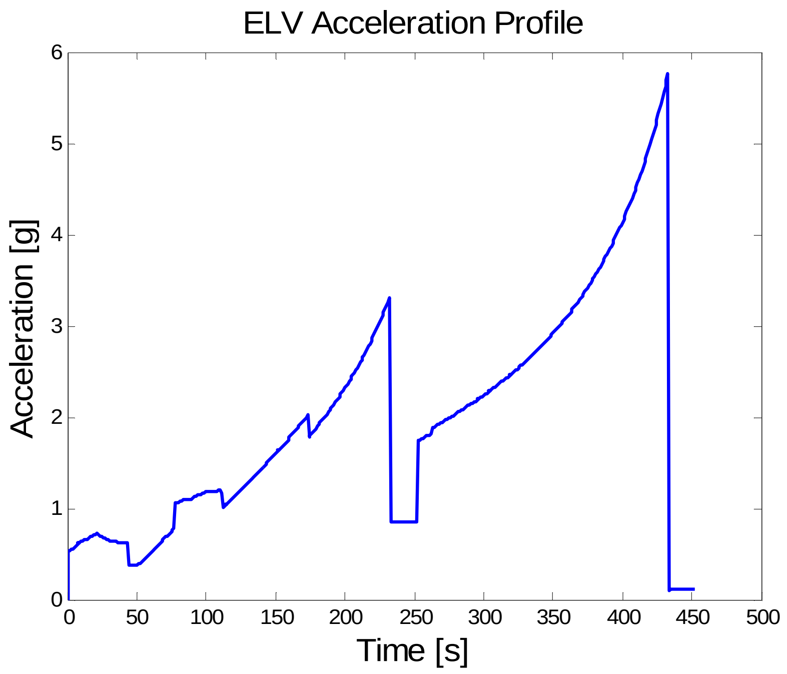


Figure 3-10: ELV Acceleration Profile

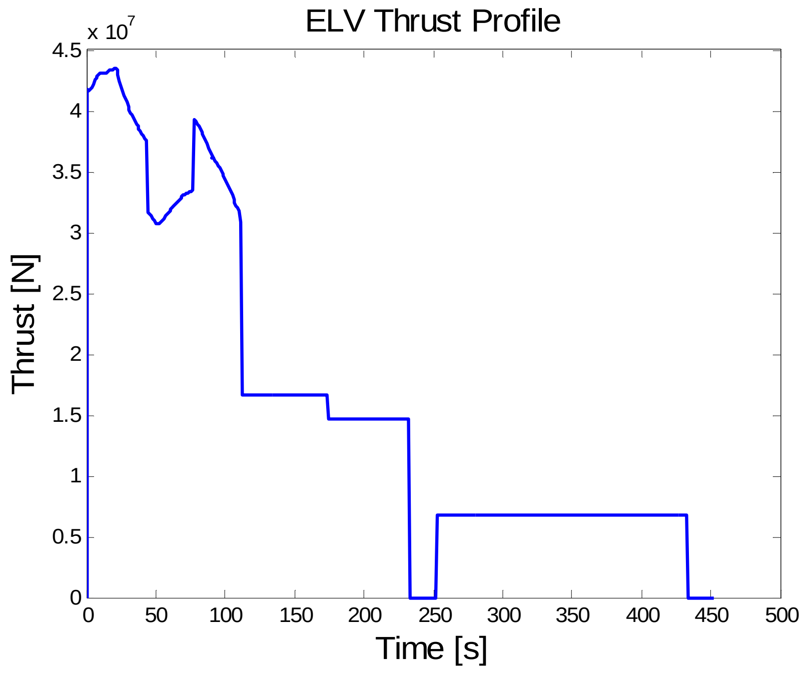


Figure 3-11: ELV Thrust Profile

6.4.2 Matlab Code

Thrust_code_Trajectory.m

```
% AAE 450
% ELV Trajectory Code
% Phil Hoff
% Caley Burke
% Original: 1-24-2005
% Last Modified: 3-21-2005

% Some of the Aerodynamics Code and functions modified from code provided by Phil
% Spindler

% Required functions to run code (eight total):
% RD180_properties; SSME_properties; thrust_alt; atmosMetric;
% atmosMetric_rev; SRB_Thrust_Profile_M_dot_2; dragRocket; pva


clc
close all
clear all

% Converts radians to degrees
rad2deg = 180/pi;

%*****************
% The above line indicates the numbers between the two lines can be changed

%height of orbit
h_o = 200; %km
h_o = h_o*1000; %m
%mass of payload
M_pay = 100e3; %kg, metric ton *10^-3

% Altitude of tower clearance, so vehicle no longer needs to be completely
% vertical
alt_begin_gturn = 600; %m
% Angle of pitch over
```

```
gturn_int = 0.03120; %radians

% These are the Mach number values between which the vehicle is at maximum
% dynamic pressure. During this time, the main engines are throttled to 65%
M_begin_maxq = 0.8;
M_end_maxq = 1.4;
maxq_throttle = 0.65;

% The main engines are throttled to 88.1% 60 seconds before the end of
% their burn
t_MECO_end = 60;
MECO_throttle = 0.881;

% Coast Time between MECO and Stage 2 Ignition 1
Coast(1) = 20;
% Coast Time after SECO I
Coast(2) = 20;

% Number of Strap-On SRBs
num_SRB = 2;
% Number of RD-180 engines
num_RD180 = 4;
% Number of SSME engines
num_SSME = 3;

%Diameters of the Stages
diam_S1 = 10.5; %m
diam_S2 = 10.5; %m
diam_pay = 10.5; %m

%*****************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Earth Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%gravitational parameter
u_E = 398600.485; % km^3/s^2
u_E = u_E*1000^3; %m^3/s^2
% radius of Earth
R_E = 6378.14; %km
R_E = R_E*1000; %m
% velocity of the Earth on the Surface at the Equator
% http://imagine.gsfc.nasa.gov/docs/ask_astro/answers/970401c.html
V_E = R_E*2*pi/(23*60*60 + 56*60 + 4.09053); %rad*m/s
% Earth's eastward rotation rate, same as V_E/R_E
w_E = 7.2921152e-5*[0,0,1]; %rad/sec
% gravity on Earth, at sea level
grav(1) = 9.8066; %m/s^2
% specific heat ratio
gam_atm = 1.4;
% specific gas constant
R_atm = 287; %J/(kg*K)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Launch Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Latitude of Launch
% based on "Pad 39-A is located at latitude 28 degrees, 36 minutes, 33
% seconds north;"
%http://spacelink.nasa.gov/NASA.Projects/Human.Exploration.and.Development.
%of.Space/Human.Space.Flight/Shuttle/Shuttle.Missions/Flight.049.STS-
46/Launch.Information/KSC.Launch.Area.Restrictions
lat = [28, 36, 33]; % [degrees, minutes, seconds]
lat_d = dot(lat, [1, 1/60, 1/3600]); %degrees
%latit_E_d = 51.6;
lat_r = lat_d/rad2deg; %radians
% Based on Launch Complex 39
% http://www.astronautix.com/sites/capveral.htm
long_d = -80.6125; %degrees
```

```
long_r = long_d/rad2deg; %radians
% Launch Azimuth
% Based on MER-B Launch
azimuth_d = 99; %degrees
azimuth_r = azimuth_d/rad2deg; %radians
% direction of azimuth
azimuth_dir = [cos(azimuth_r), sin(azimuth_r), 1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Orbit Variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Circular Orbit Velocity
Vc = sqrt(u_E/(R_E + h_o)); %m/s
%Velocity due to Earth Rotation
Vr = V_E * cos(lat_r); %m/s
% Estimated Gravity Losses
Vg = 1.4*1000; %m/s
% Estimated Drag Losses
Vd = 0.1*1000; %m/s
% Estimated Steering Losses
Vs = 0.2*1000; %m/s
%Estimated Delta Velocity Requirement
d_V = Vc + Vg + Vd + Vs - Vr; %m/s

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %Initial State
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Initial Acceleration
Accel = [0,0,0]; %m/s^2
%Initial Altitude
alt = 0; %m
%Initial Horizontal Distance traveled from Pad
range = 0; %m
%Initial Distance from Earth COM
R_mag = R_E + alt;
%Initial Position relative to Earth COM
R_V = R_mag*[cos(lat_r)*cos(long_r), cos(lat_r)*sin(long_r), sin(lat_r)];
%Initial Velocity Vector
Vvec = cross(w_E,R_V); %m/s
%Initial Velocity (Vertical)
V_v = dot(Vvec,R_V);%*R_V/norm(R_V); %m/s
%Initial Velocity (Horizontal WRT Inertial Position)
V_hvec = Vvec-V_v; %m/s


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Engine Properties
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For a single SRB, gives:
% Thrust and Mass Flow Profiles, Nozzle Exit Area, Mass at Lift-off, Inert
% Mass,  Nozzle Area Ratio, & Diameter

SRB_properties

% Each function gives the following for their engine:
% Vacuum Thrust, Nozzle Exit Area, Max Mass Flow Rate,
% Oxidizer to Fuel Ratio,  Vacuum Isp, & Sea Level Isp

RD180_properties
SSME_properties

Tvac_RD180 = Tvac_RD180_full;
Tvac_SSME = Tvac_SSME_full;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Stage Mass Estimation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % Masses of the 1st and 2nd Stages, based on actual structural numbers
    M_prop_S1 = 1.1138e+006;
    M_eng_S1 = num_RD180*5393;
    M_S1 = 76723.2 + M_prop_S1 + 609.19;

    M_prop_S2 = 2.7655e+005;
    M_eng_S2 = num_SSME*3177;
    M_S2 = 29630.9 + M_prop_S2 + 702.24 + 237.24;

    M_Fairing = 13.676e3;

    %Initial Mass of the SRBs all together
    M_SRBs = Mtot_SRB*num_SRB; %kg

    %Initial Vehicle Mass
    M_Vehicle = M_SRBs + M_S1 + M_S2 + M_pay + M_Fairing; %kg

    % Burn Time of Stage 1
    bt_S1 = M_prop_S1/(num_RD180*Mdot_RD180)+t_MECO_end*(1/MECO_throttle-1)+1;
    % Total Burn Time of Stage 2
    bt_S2 = M_prop_S2/(num_SSME*Mdot_SSME)+1;

    % Cross-sectional Areas of each stage
    A_SRB = pi*(diam_SRB/2)^2*num_SRB; %m^2
    A_S1 = pi*(diam_S1/2)^2; %m^2
    A_S2 = pi*(diam_S2/2)^2; %m^2
    A_pay = pi*(diam_pay/2)^2; %m^2
    % Initial Vehicle Cross-Sectional Area
    A_Vehicle = A_S1 + A_SRB; %m^2

    % Total time observing
    time_end = bt_S1 + bt_S2 + sum(Coast);
    %index of the time when SRBs weigh more than their thrust at 112 sec
    SRB_end = find(t_SRB>= 112, 1, 'first');
    % Time vector for launch
    t = [0 t_SRB(1:SRB_end)', 113:time_end];
    %index of the time when 1st Stage stops burn time
    MECO = find(t>=bt_S1, 1, 'first');
    %index of the time when 2nd Stage ignites 1, after Coast 1
    S2_ign1 = find(t>=(bt_S1+Coast(1)), 1, 'first');
    %index of the time when 2nd Stage stops burn time, 1
    SECO1 = find(t>=(bt_S1+Coast(1)+bt_S2), 1, 'first')
    % Length of the vector for time
    len_t = length(t);

    %Aerodynamics - Initial Values
    gam = pi/2;
    d_gam = 0;
    Drag = 0;
    V_vec = [0,0,0];
    Vel_dir = [0, 0, 1];
    V_i_vec = [0,Vr,0];
    %Centripetal Acceleration
    V_mag = 0;
    Mach = 0;
    %Dynamic Pressure
    q = 0;

    % Marks if altitude to being g-turn has been reached or not
    h = 0;
    % Marks if begin controlled g-turns on or not
    b = 0;
    delay = 11+160;

    for k = 1:(len_t - 1)

        % Change in time
        d_t(k+1) = t(k+1) - t(k);

        % Gravity Turn
        % Requires that the vehicle clear the tower before turning
        if alt(k) > alt_begin_gturn | t(k) > MECO ;
            if V_mag(k) > 0
                d_gam(k+1) = -(grav(k)-V_i_mag(k)^2/R_mag(k))*cos(gam(k))/V_i_mag(k);
            else
                d_gam(k+1) = 0;
            end

            if d_gam(k+1)>10/rad2deg
                d_gam(k+1) = 10/rad2deg;
            end
        else
            d_gam(k+1) = 0;
        end

        if alt(k) < alt_begin_gturn & t(k) < MECO;
            gam(k+1) = pi/2;
        % a pitchover manuever
        elseif h == 0
            gam(k+1) = pi/2 - gturn_int;
            h = 1;
        else
            % gravity turn is being implemented here
            gam(k+1) = gam(k)+d_gam(k+1)*d_t(k+1);
        end

        % Direction of Thrust due to gam and launch azimuth
        Thrust_dir(k+1,:) = [cos(gam(k+1)),cos(gam(k+1)),sin(gam(k+1))].*azimuth_dir; %[latitude,
    longitude, vertical]

        % Throttles 1st Stage Engines when going through Maximum Dynamic Pressure
        if Mach(k) > M_begin_maxq & Mach(k)< M_end_maxq
            Tvac_RD180 = Tvac_RD180_full*maxq_throttle;
        elseif k > (MECO - t_MECO_end)
            Tvac_RD180 = Tvac_RD180_full*MECO_throttle;
        else
            Tvac_RD180 = Tvac_RD180_full;
        end

        % Thrust, Mass, and Relevant CS Area of the SRBs at any given time
        if k < SRB_end
            T_SRB(k+1) = num_SRB*thrust_alt(alt(k), Ae_SRB, Tvac_SRB(k)); %N
            Mdot_SRBs(k+1) = num_SRB*mean(Mdot_SRB(k:k+1));% kg/s
            M_SRBs(k+1) = M_SRBs(k) - Mdot_SRB(k+1)*d_t(k+1); %kg
            A_SRB(k+1) = A_SRB(1); %m^2
        else
            T_SRB(k+1) = 0; %kN
            Mdot_SRB(k+1) = 0;% kg/s
            M_SRBs(k+1) = 0; %kg
            A_SRB(k+1)  = 0; %m^2
        end

        % Thrust, Mass, and Relevant CS Area of the 1st Stage at any given time
        if k < MECO
            T_S1(k+1) =  num_RD180*thrust_alt(alt(k), Ae_RD180, Tvac_RD180); %N
            d_M_S1(k+1) = num_RD180*Mdot_RD180;% kg/s
            M_S1(k+1) = M_S1(k) - d_M_S1(k+1)*d_t(k+1); %kg
            A_S1(k+1) = A_S1(1); %m^2
        else
            T_S1(k+1) = 0; %kN
            d_M_S1(k+1) = 0;% kg/s
            M_S1(k+1) = 0; %kg
            A_S1(k+1) = 0; %m^2
        end

        % Thrust, Mass, and Relevant CS Area of the 2nd Stage at any given time
        if k < S2_ign1
            T_S2(k+1) = 0; %N
            d_M_S2(k+1) = 0;% kg/s
            M_S2(k+1) = M_S2(k); %kg
            A_S2(k+1) = A_S2(1); %m^2
        elseif k < SECO1
            T_S2(k+1) =  num_SSME*thrust_alt(alt(k), Ae_SSME, Tvac_SSME); %N
            d_M_S2(k+1) = num_SSME*Mdot_SSME;% kg/s
            M_S2(k+1) = M_S2(k) - d_M_S2(k+1)*d_t(k+1); %kg
            A_S2(k+1) = A_S2(1); %m^2
        else
            T_S2(k+1) = 0; %kN
            d_M_S2(k+1) = 0;% kg/s
            M_S2(k+1) = 0; %kg
            A_S2(k+1) = 0; %m^2
        end

        % The mass of the fairing stays with the vehicle until 10 seconds after
        % 2nd Stage ignition
        if k < S2_ign1 +10
            M_Fairing(k+1) = M_Fairing(1);
        else
            M_Fairing(k+1) = 0;
        end

        % Total Thrust, Mass, and Relevant CS Area of the Vehicle
        Thrust(k+1) = T_S1(k+1) + T_SRB(k+1) + T_S2(k+1); %N
        M_Vehicle(k+1) = M_SRBs(k+1) + M_S1(k+1) + M_S2(k+1) + M_Fairing(k+1) + M_pay; %kg
        A_Vehicle(k+1) = max([A_S1(k+1), A_S2(k+1), A_pay]) + A_SRB(k+1); %m^2

        % Drag Calculations
        % [density, Temperature, Pressure] for the given altitude
        [rho_atm(k+1), Temp_atm, P_atm(k+1)]= atmosMetric(alt(k)); %[kg/m^3, K, N/m^2]
        if Temp_atm == 0
            Drag(k+1) = 0;
            Mach(k+1) = 0;
        else
            % Speed of Sound at altitude
            a_atm = sqrt(gam_atm*R_atm*Temp_atm);
            % Mach Number at given altitude and velocity
            Mach(k+1) = norm(V_vec(k,:))/a_atm;
            % Drag Coefficient
            CD = dragRocket(Mach(k+1));
            Drag(k+1) = .5*rho_atm(k+1)*V_mag(k)^2*A_Vehicle(k+1)*CD;
        end

        Force_vec(k+1,:) = Thrust(k+1)*Thrust_dir(k+1,:) - Drag(k+1)*Vel_dir(k,:);

        %Dynamic Pressure
        q(k+1) = (1/2)*rho_atm(k+1)*V_mag(k)^2;

        % Finds the position, velocity, acceleration, altitude, and range
        % of the vehicle at the current time
        if k == 1
            [R_mag(k+1), lat_r(k+1), long_r(k+1), V_vec(k+1,:),  V_mag(k+1), V_i_vec(k+1,:),
    V_i_mag(k+1), Accel(k+1,:), Accel_mag(k+1), alt(k+1), range(k+1), d_Vr(k+1)]= ...
                pva(R_mag(k), [lat_r(k), lat_r(k)], long_r(k), V_i_vec(k,:), Accel(k,:), Force_vec(k+1,:),
    M_Vehicle(k+1), d_t(k+1), range(k));
        else
            [R_mag(k+1), lat_r(k+1), long_r(k+1), V_vec(k+1,:),  V_mag(k+1), V_i_vec(k+1,:),
    V_i_mag(k+1), Accel(k+1,:), Accel_mag(k+1), alt(k+1), range(k+1), d_Vr(k+1)]= ...
                pva(R_mag(k), lat_r(k-1:k), long_r(k), V_i_vec(k,:), Accel(k,:), Force_vec(k+1,:),
    M_Vehicle(k+1), d_t(k+1), range(k));
        end

        % Gravity at given altitude
        grav(k+1) = u_E/(R_mag(k+1))^2; %m/s
        %Magnitude of the Ground Speed
        V_h(k+1) = norm(V_vec(k+1,1:2));

        Vel_dir(k+1,:) = norm_vector_matrix(V_vec(k+1,:));

    end

    % Error in altitude
    Error_Alt = (alt(end) - h_o)/1000; %km
    % Range and Altitude in km
```

```
alt_k = alt/1000; %km
range_k = range/1000; %km

% Final Altitude
alt_end = alt_k(end);

%Circular Orbit Velocity
Vc_end = sqrt(u_E/(R_E + alt(end))); %m/s

% Maximum dynamic pressure actually experienced
[q_max, i_q_max] = max(q);

% Difference between what the velocities ought to be and actual
Error_Vel_Hor = Vc - V_h(end);
Error_Vel_Hor2 = Vc_end - V_h(end);
Error_Vel_Vert = -V_vec(end,3);

% Latitude and Longitude in degrees
long_d = long_r*rad2deg; %degrees
lat_d = lat_r*rad2deg; %degrees

fprintf('\nFinal Altitude:\t %6.1f km\n', alt_end)
fprintf('Error Altitude:\t %6.1f km\n', Error_Alt)
fprintf('Final Velocity:\t %6.1f m/s\n', V_mag(end))
fprintf('Final Tangential Velocity:\t %6.1f m/s\n', V_h(end))
fprintf('Error in Tangential Velocity:\t %6.1f m/s\n', Error_Vel_Hor)
fprintf('Error in Actual Tangential Velocity:\t %6.1f m/s\n', Error_Vel_Hor2)
fprintf('Error in Vertical Velocity:\t %6.1f m/s\n', Error_Vel_Vert)
fprintf('Mass of 1st Stage:\t %6.1f kg\n', M_S1(1))
fprintf('Mass of 2nd Stage:\t %6.1f kg\n', M_S2(1))
fprintf('Mass of Entire Vehicle:\t %6.1f kg\n', M_Vehicle(1))

figure(1)
plot(t(1:k+1), Thrust,'LineWidth',2)
xlabel('Time [s]', 'Fontsize',15); ylabel('Thrust [N]', 'Fontsize',15);
title('ELV Thrust Profile', 'Fontsize',15)
print -dmeta Traj_2_22_fig_thrustprofile

figure(2)
plot(t(1:(length(t))), Drag)
xlabel('Time [s]'); ylabel('Drag [N]');

figure(2)
plot(t(1:k+1), Accel_mag./grav,'LineWidth',2)
xlabel('Time [s]', 'Fontsize',15); ylabel('Acceleration [g]', 'Fontsize',15);
title('ELV Acceleration Profile', 'Fontsize',15)
print -dmeta Traj_3_22_fig_accelprofile

figure(3)
plot(t(1:k+1), Accel(:,1)'./grav, 'b')
hold on
plot(t(1:k+1), Accel(:,2)'./grav, 'r')
plot(t(1:k+1), Accel(:,3)'./grav, 'g')
xlabel('Time [s]'); ylabel('Acceleration [g]');
legend('Latitudal', 'Longitudal', 'Vertical');
hold off

figure(4)
plot(t(1:k+1), V_mag)
xlabel('Time [s]'); ylabel('Magnitude of Total Velocity [m/s]');

figure(5)
plot(t(1:k+1), V_h)
xlabel('Time [s]'); ylabel('Velocity Tangent to the Earth [m/s]');

figure(6)
plot(t(1:k+1), alt_k,'b:','LineWidth',3)
hold on
plot(t(1:k+1),h_o/1000,'r--')
plot(t(1:k+1),0,'g','LineWidth',2)
xlabel('Time [s]', 'Fontsize',15); ylabel('Altitude [km]', 'Fontsize',15);
```

```
axis([0 t(end) -50 250])
hold off
title('ELV Altitude Profile', 'Fontsize',15)
print -dmeta Traj_3_22_fig_altprofile

figure(7)
plot(t(1:k+1),M_Vehicle./1000)
hold on
plot(t(1:k+1),zeros(1,length(t))+120,'-r')
xlabel('Time [s]'); ylabel('Vehicle Mass [mT]');
hold off

figure(8)
plot(t(1:k+1),gam*rad2deg)
xlabel('Time [s]'); ylabel('gamma [deg]');
hold off

figure(9)
plot(t(1:k+1),d_gam*rad2deg)
xlabel('Time [s]'); ylabel('d_gamma [deg]');
hold off

figure(10)
plot(range_k(1:SRB_end), alt_k(1:SRB_end), 'b')
hold on
plot(range_k(SRB_end:MECO), alt_k(SRB_end:MECO), 'r')
plot(range_k(MECO:S2_ign1), alt_k(MECO:S2_ign1), 'y')
plot(range_k(S2_ign1:SECO1), alt_k(S2_ign1:SECO1), 'g')
plot(range_k(SECO1:len_t), alt_k(SECO1:len_t), 'm')
title('ELV Trajectory Shape')
xlabel('Range (km)')
ylabel('Altitude (km)')
legend('SRB', 'MECO', 'Coast 1', 'SECO1', 'Coast 2')
hold off


figure(11)
plot(t(1:k+1),Mach)
xlabel('Time [s]')
ylabel('Mach Number')

figure(12)
plot(t(1:k+1),Drag/1000)
xlabel('Time [s]')
ylabel('Drag [kN]')

figure(13)
plot(t(1:k+1),q)
xlabel('Time [s]')
ylabel('Dynamic Pressure [N/m^2]')

figure(14)
plot(long_d(1:SRB_end), lat_d(1:SRB_end), 'b')
hold on
plot(long_d(SRB_end:MECO), lat_d(SRB_end:MECO), 'r')
plot(long_d(MECO:S2_ign1), lat_d(MECO:S2_ign1), 'y')
plot(long_d(S2_ign1:SECO1), lat_d(S2_ign1:SECO1), 'g')
plot(long_d(SECO1:len_t), lat_d(SECO1:len_t), 'm')
xlabel(' Longitude [deg]'); ylabel('Latitude [deg]');
legend('SRB', 'MECO', 'Coast 1', 'SECO1', 'Coast 2')
axis([-120 220 -70 70])
hold off

figure(15)
plot(alt_k, V_mag,'LineWidth',2)
xlabel('Altitude (km)', 'Fontsize',15); ylabel('Magnitude of Total Velocity [m/s]', 'Fontsize',15);
title('ELV Altitude vs Velocity', 'Fontsize',15)
print -dmeta Traj_2_22_fig_altvsvel

aoa = compare_T_V(Thrust_dir, V_vec);
```

```
figure(6)
```

RD180_properties

```
% AAE 450
% Caley Burke
% original: 2-12-05

function RD180_properties

% This function assigns the properties of the RD180 Engine listed below into
% the workspace

% Vacuum Thrust of RD-180 engines
Tvac_RD180_full = 4152e3; %N
% RD_180 Nozzle Exit Area
Ae_RD180 = 2*pi*(1.5/2)^2; %m^2
% RD_180 Max Mass Flow Rate
Mdot_RD180 = 1252.19; %kg/s
% RD-180 Oxidizer to Fuel Ratio
O2F_RD180 = 2.72;
% Vacuum Isp
Isp_vac_RD180 = 338; %s
% Sea Level Isp
Isp_sl_RD180 = 311; %s

assignin('base','Tvac_RD180_full',Tvac_RD180_full)
assignin('base','Ae_RD180',Ae_RD180)
assignin('base','Mdot_RD180',Mdot_RD180)
assignin('base','O2F_RD180',O2F_RD180)
assignin('base','Isp_vac_RD180',Isp_vac_RD180)
assignin('base','Isp_sl_RD180',Isp_sl_RD180)
```

SSME_properties

```
% AAE 450
% Caley Burke
% original: 2-12-05

function SSME_properties

% This function assigns the properties of the SSME Engine listed below into
% the workspace

% Vacuum Thrust of SSME engines
Tvac_SSME_full = 2278e3; %N
% SSME Nozzle Exit Area
Ae_SSME = pi*(1.63/2)^2; %m^2
% SSME Max Mass Flow Rate
Mdot_SSME = 512.7661437; %kg/s
% SSME Oxidizer to Fuel Ratio
O2F_SSME = 6;
% SSME Vacuum Isp
Isp_vac_SSME = 453; %s
% SSME Sea Level Isp
Isp_sl_SSME = 363; %s

assignin('base','Tvac_SSME_full',Tvac_SSME_full)
assignin('base','Ae_SSME',Ae_SSME)
assignin('base','Mdot_SSME',Mdot_SSME)
assignin('base','O2F_SSME',O2F_SSME)
assignin('base','Isp_vac_SSME',Isp_vac_SSME)
assignin('base','Isp_sl_SSME',Isp_sl_SSME)
```

thrust_alt

```
% AAE 450
% Thurst Altitude
% Caley Burke
% Original: 1-24-2005
% This code finds the actual thrust of the vehicle from the vacuum thrust based on the
% current altitude

function Thrust = thrust_alt(Altitude, AE, T_vac)
    % Finds the atmospheric pressure at the given altitude
    Pa = atmosMetric_rev(Altitude); %N/m^2
    % Finds the actual thrust at the given pressure
    Thrust = T_vac - Pa*AE;
    return
```

atmosMetric

```
% Phil Spindler

function [densM,tK,pressureM]= atmosMetric(altMeter)
%
alt=altMeter*3.281;
%% convert altitude to feet
%         results = atmos(altitude,output)
%
%         Units are Calculated in English and converted to metric at the end
%
%       density      =  slugs / ft^3
%
%               temperature  =  degrees Rankine
%
%       pressure     =  density * R * Temp = lb / ft^2
%
%
%               R = 1716
%

    n = length(alt);
    t = zeros(1,n);
        dens = zeros(size(t));

        r = 1716.0;
        gc = 32.174;

    for i = 1:n

        if alt(i) < 35000

                a       = -0.003566;
                t(i)    = a * alt(i) + 518.69;
                dens(i) = (6.38e-15) .* t(i) .^ (-(1 + (gc / (a * r))));

            elseif alt(i) < 65578

            t(i)    = 390.0;
            dens(i) = (0.004) .* exp(-(gc * alt(i) ./ (r * t(i))));

            elseif alt(i) < 104924

                a       = 0.0005208;
                t(i)    = a * alt(i) + 357.6;
                dens(i) = 0.004491 .* exp(-alt(i) ./ 20369.0);

            elseif alt(i) < 154107

                a       = 0.001629;
                t(i)    = a * alt(i) + 245.26;
                dens(i) = (1.572e+28) .* t(i) .^ (-(1.0 + (gc / (a * r))));

            elseif alt(i) < 170502

                t(i)    = 487.8;
                dens(i) = (0.001075) .* exp(-(gc * alt(i) ./ (r * t(i))));

            elseif alt(i) < 235700

                a       = -0.001591;
                t(i)    = a * alt(i) + 749.89;
                dens(i) = (1.825e-35) .* t(i) .^ (-(1 + (gc / (a * r))));

            elseif alt(i) < 278705

                a       = -0.001;
                t(i)    = a * alt(i) + 611.7;
                dens(i) = 0.02542 .* exp(-alt(i) ./ 19311.0);

            elseif alt(i) < 295099

                t(i)    = 321;
                dens(i) = 0.18485 .* exp(-(gc * alt(i)) ./ (r * t(i)));

            elseif alt(i) < 327888

                a       = 0.000884;
                t(i)    = a * alt(i) + 60.13;
                dens(i) = (3.09e+47) .* t(i) .^ (-(1.0 + (gc / (a * r))));

            elseif alt(i) < 360777

                a       = 0.0025;
                t(i)    = a * alt(i) - 469.72;
                dens(i) = (4.2335e12) .* t(i) .^ (-(1.0 + (gc / (a * r))));

            elseif alt < 400000

                a       = 0.00659;
                t(i)    = a * alt(i) - 1944.86;
                dens(i) = 2.2788 .* t(i) .^ (-(1.0 + (gc / (a * r))));

            else

                dens(i) = 0;
                t(i)    = 0;

            end

    end

        pressure = r.*dens.*t;


densM=dens*515.4;
% density in kg/m^3
tK=t*273/492;
%
pressureM=pressure*47.88;
% pressure in n/m^2
```

atmosMetric_rev

This code is the exact same as atmosMetric except the first active line of code is changed to the following:

```
function pressureM = atmosMetric(altMeter)
```

SRB_Thrust_Profile_M_dot_2

```
% AAE 450
% Caley Burke
% Data Source: Archambeau

% Averaged the Thrust and Mass Flow rate between every 2 seconds to come up
% with 1 second change in time
%Burn Time: 111.6s
%Time (s)   Fvac (N)    Mdot (kg/s)
function [t, Fvac, Mdot] = SRB_Thrust_Profile_M_dot;
SRB_Func = [1   14135003.84 5374.298833
    2   14066968.29 5344.769968
    3   14142152.14 5369.241278
    4   14217335.98 5393.712588
    5   14368606.67 5446.69218
    6   14519877.36 5499.671772
    7   14595058.98 5526.592481
    8   14670240.6  5553.51319
    9   14708350.75 5566.485933
    10  14746460.89 5579.458675
    11  14744739.43 5577.485549
    12  14743017.96 5575.512422
    13  14746492.03 5575.489742
    14  14749966.09 5575.467062
    15  14761644.89 5578.506131
    16  14773323.7  5581.5452
    17  14793207.25 5587.668698
    18  14813090.81 5593.792195
    19  14814616.55 5593.021088
    20  14816142.29 5592.249981
    21  14784477.62 5579.073122
    22  14752812.95 5565.896263
    23  14491026.19 5469.326441
    24  14229239.43 5372.756619
    25  13996290.49 5286.868898
    26  13763341.55 5200.981177
    27  13574694.68 5131.58154
    28  13386047.81 5062.181902
    29  13216548.31 4999.903666
    30  13047048.81 4937.625429
    31  12889330.43 4879.679
    32  12731612.04 4821.732571
    33  12584442.62 4767.687037
    34  12437273.19 4713.641502
    35  12298960.18 4662.861833
    36  12160647.16 4612.082164
    37  12024422.59 4562.028243
    38  11888198.02 4511.974321
    39  11754958.2  4462.963662
    40  11621718.38 4413.953004
    41  11506375.98 4371.632833
    42  11391033.59 4329.312662
    43  11298897.56 4295.724145
    44  11206761.54 4262.135627
    45  11129584.89 4234.126297
    46  11052408.24 4206.116966
    47  10952269.87 4169.398661
    48  10852131.49 4132.680356
    49  10723193.1  4085.053154
    50  10594254.72 4037.425952
    51  10551903.2  4022.480083
    52  10509551.67 4007.534213
    53  10528396.57 4015.449401
    54  10547241.46 4023.364588
    55  10603511.47 4045.477218
    56  10659781.48 4067.589847
    57  10717081.25 4090.11071
    58  10774381.02 4112.631572
    59  10821623.36 4131.342259
    60  10868865.7  4150.052945
    61  10941436.22 4178.40247
    62  11014006.74 4206.751995
```

```
63  11063144.02 4226.211109
64  11112281.31 4245.670223
65  11151052.01 4261.228443
66  11189822.71 4276.786662
67  11226803.01 4291.664493
68  11263783.3  4306.542323
69  11311643.94 4325.570525
70  11359504.59 4344.598726
71  11380927.23 4353.602535
72  11402349.86 4362.606344
73  11425009.11 4372.063746
74  11447668.35 4381.521147
75  11455459.41 4385.376682
76  11463250.47 4389.232218
77  11478472.29 4395.877347
78  11493694.1  4402.522475
79  11374897.66 4358.229177
80  11256101.22 4313.935879
81  11161138.35 4277.988682
82  11066175.49 4242.041484
83  10938442.57 4193.597816
84  10810709.65 4145.154147
85  10699608.63 4103.038093
86  10588507.61 4060.922039
87  10419299.47 3996.647996
88  10250091.33 3932.373953
89  10072229.18 3864.811365
90  9894367.02  3797.248777
91  9845049.583 3778.583449
92  9795732.145 3759.918122
93  9708257.86  3726.737838
94  9620783.574 3693.557554
95  9502583.195 3648.651906
96  9384382.816 3603.746259
97  9265971.146 3558.749893
98  9147559.477 3513.753527
99  9046642.664 3475.424969
100 8945725.852 3437.096411
101 8808235.758 3385.114722
102 8670745.664 3333.133033
103 8541849.535 3284.37185
104 8412953.406 3235.610667
105 8251578.584 3174.443732
106 8090203.763 3113.276797
107 7951319.151 3060.637399
108 7812434.54  3007.998001
109 7713010.106 2970.395191
110 7613585.672 2932.792381
111 7157202.543 2765.462144
112 6700819.414 2598.131908
113 5756715.432 2239.725873
114 4812611.45  1881.319838
115 3897514.768 1548.677852
116 2982418.087 1216.035865
117 2429712.491 1000.670194
118 1877006.896 785.3045221
118.2   1790407.576 749.6975187
118.4   1704948.779 714.5441077
118.6   1621161.625 680.0257261
118.8   1543178.956 647.8660249
119 1469773.951 617.6114118
119.2   1399411.976 588.536139
119.4   1327275.604 558.6897591
119.6   1254668.611 528.616583
119.8   1183162.553 498.95164
120 1112660.013 469.6949302
120.2   1050980.522 440.8464536
120.4   990737.8069 412.9958802
120.6   932985.6511 386.5514433
120.8   877387.7691 361.2863466
121 806007.5955 329.9431118
```

```
    121.2   730359.3529 297.2390997
    121.4   655978.8535 265.4876317
    121.6   585270.8063 235.5505333
    121.8   519834.3469 208.0628339
    122 461990.5576 47.8539982
    122.2   409579.8267 162.2046422
    122.4   361620.4316 142.4280136];

t = SRB_Func(:,1);
Fvac = SRB_Func(:,2);
Mdot = SRB_Func(:,3);
```

dragRocket

```
% Phil Spindler

function Cd=dragRocket(M)
% calculate the drag of a rocket based on frontal area
% assumes conical top structure

    if M <= .8
         Cd = .4;
    elseif M <= 1.5
         Cd = .8570.*M - .2857;
    elseif  M > 1.5
        Cd = .55 + .45.*exp( - .9.*(M - 1.5));
    end
```

pva

```
% AAE 450
% Caley Burke
% Finds new position, velocity, acceleration
% Given: previous position, velocity, current thrust, drag, vehicle mass

function  [rho, phi, theta, Vel, Vel_mag, Vel_i, Vel_mag_i, Accel, Accel_mag, altitude, range, d_Vr] =
...
    pva(rho_prev, phi_prev, theta_prev, Vel_i_prev, Accel_prev, ...
    Force, M_Vehicle, d_t, range_prev)

% rho = distance from the center of the Earth
% phi = latitude
% theta = longitude

% Cartesian to Spherical conversion
% rho = [x^2 + y^2 + z^2]^0.5
% phi = asin(z/r)
% theta = atan(y/x)

%gravitational parameter
u_E = 398600.485; % km^3/s^2
u_E = u_E*1000^3; %m^3/s^2

% radius of Earth
R_E = 6378.14; %km
R_E = R_E*1000; %m

% gravity  at given altitude
grav = u_E/rho_prev^2; %m/s

% velocity of the Earth on the Surface at the Equator
% http://imagine.gsfc.nasa.gov/docs/ask_astro/answers/970401c.html
V_E = R_E*2*pi/(23*60*60 + 56*60 + 4.09053); %rad*m/s

% Centripetal Acceleration
A_cent = norm(Vel_i_prev(1:2))^2/rho_prev; %m/s^2
% Total Acceleration
Accel = Force/M_Vehicle + [0, 0, A_cent - grav]; %m/s^2
```

```
if (Accel(3) < 0 & rho_prev < (500 + R_E));
    Accel(3) = 0;
end

% Acceleration magnitude
Accel_mag = norm(Accel);

% Averages current and previous accelerations
Accel_avg = mean([Accel; Accel_prev]);

% Change in the Velocity of the Earth on the Surface given the latitude
Vr_old = V_E*cos(phi_prev(1));
Vr_new = V_E*cos(phi_prev(2));
d_Vr = Vr_new - Vr_old;

% Finds the current velocity
Vel_i = Accel_avg*d_t + Vel_i_prev + [0, d_Vr, 0] ;
% Velocity magnitude
Vel_mag_i = norm(Vel_i);

% Averages current and previous velocities
Vel_i_avg = mean([Vel_i;Vel_i_prev]);

% Spherical Velocities
vel_phi = Vel_i_avg(1)/(2*pi*rho_prev); %rad/s
vel_theta = Vel_i_avg(2)*d_t/(2*pi*rho_prev); %rad/s
vel_rho = Vel_i_avg(3)*d_t; %m/s

% Spherical position
phi = phi_prev(2) + vel_phi*d_t;
theta = theta_prev + vel_theta*d_t;
rho = rho_prev + vel_rho*d_t;

% Vehicle Velocity (Earth's Inertial Velocity not included)
Vel = Vel_i - [0, Vr_new, 0];
Vel_mag = norm(Vel);

% New altitude
altitude = rho - R_E; %m

% Total changes in latitude and longitude
d_phi_tot = phi - phi_prev(2); %rad
d_theta_tot = theta - theta_prev; %rad

% Angle change, based on latitude and longitude
d_angle = norm([d_phi_tot, d_theta_tot]); %rad

% Distance covered on the Earth
d_range = d_angle*R_E; %m

% Total Distance covered by the vehicle
range = range_prev + d_range; %m
```

Two_Stage_Mass_SRB_Estimate

```
% AAE 450
% Caley Burke
% Given Isp, delta v percent responsibility, and expected propellant mass fraction, gives inital
estimates
% for masses of a two stage vehicle

% Revised from AAE 439 HW#3 Problem 1
% Fall 2003

function [moi12, m_prop_12, moi22, m_prop_22] = ...
    Two_Stage_Mass_SRB_Estimate(Isp1, zeta1, Isp2, zeta2, mpay, dv_m, per_dv_SRB, per_dv_S1)

% Isp - seconds
% zeta - propellant mass fraction, unitless
% mpay - payload mass, kg
% dv_m - change in velocity, m/s
% per_dv_SRB - percent of dv_m due to the SRBs, unitless
% per_dv_S1 - percent of dv_m due to the 1st stage, unitless

% All returned masses are in kg

%Givens
%Stage 1
MR1 = 1 - zeta1;
%Stage 2
MR2 = 1 - zeta2;

%Gravitational constant
g0= 9.807;%m/s^2

% Divides the delta v between the different stages
dv_m_0 = dv_m*per_dv_SRB;
dv_m_1 = dv_m*per_dv_S1;
dv_m_2 = dv_m*(1 - per_dv_SRB - per_dv_S1);

%For two stage vehicle
%calculates inert & propellant mass of the 2nd stage
moi22 = mpay*(1-exp(dv_m_2/(Isp2*g0)))/(exp(dv_m_2/(Isp2*g0))*MR2-1);
m_prop_22 = moi22*zeta2;

%Payload of first stage
mpay12 = mpay + moi22;
%calculates inert & propellant mass of the first stage
moi12 = mpay12*(1-exp(dv_m_1/(Isp1*g0)))/(exp(dv_m_1/(Isp1*g0))*MR1-1);
m_prop_12 = moi12*zeta1;

%gross overall mass
mgross2 = mpay12 + moi12;%kg

%gross overall weight
wgross2 = mgross2*g0; %F=mg, N
```

prop_size

```
% AAE 450
% Caley Burke
% Original: 2-7-2005

% Calculates the volume and height of the tanks for propellant based on
% given information

function [Mass_Ox, Vol_Ox, Tank_h_Ox, Mass_Fuel, Vol_Fuel, Tank_h_Fuel] = ...
    prop_size(Tank_Area, Mass_prop, O2F_Ratio, Ox, Fuel)

% Oxidizer mass
Mass_Ox = Mass_prop*O2F_Ratio/(O2F_Ratio + 1); %kg
% Fuel mass
Mass_Fuel = Mass_prop/(O2F_Ratio + 1); %kg

% Determines the density of Oxidizer
if Ox == 'LOX' | Ox == 'Lox' | Ox == 'LOx' | Ox == 'lox'
    % Oxidizer is Liquid Oxygen
    rho_Ox = 1.14e3 ; %kg/m^3 source = astronautrix
    %    rho_Ox = 1.116 e3 ; %kg/m^3 source: Shuttle Reference Manual
else
    fprintf('Oxidizer type error')
    return
end

% Determines the density of Fuel
if Fuel == 'RP1' | Fuel == 'rp1' | Fuel == 'Rp1'
    % Fuel is RP-1 or Kerosene
    rho_Fuel = 0.81e3 ; %kg/m^3
elseif Fuel == 'LH2' | Fuel == 'Lh2' | Fuel == 'lh2'
```

```
    % Fuel is Liquid Hydrogen
    rho_Fuel =  0.071e3 ; %kg/m^3
else
    fprintf('Fuel type error')
    return
end

Vol_Ox = Mass_Ox/rho_Ox; %m^3
Vol_Fuel = Mass_Fuel/rho_Fuel; %m^3

Tank_h_Ox = Vol_Ox/(Tank_Area - 0.25)+1; %m^2
Tank_h_Fuel = Vol_Fuel/(Tank_Area - 0.25+1); %m^2
```

References:

[1] Humble, Ronald W., Gary N. Henry, and Wiley J. Larson. Space Propulsion Analysis and Design. 1995: McGraw-Hill, New York.

[2] Sutton, George P. and Oscar Biblarz. Rocket propulsion elements. 2001: John Wiley & Sons, New York.

[3] Isakowitz, Steven J. International Reference Guide to Space Launch Systems. 1991: AIAA, Washington, D.C.

[4] Shuttle Reference Manual. 1998: NASA. 5 Feb. 2005. http://spaceflight.nasa.gov/shuttle/reference/shutref/verboseindex.html

[5]Wade, Mark. Encyclopedia Astronautica. Dec. 2003. Space Wire. 24 Jan. 2005. <http://www.astronautix.com/engines/index.htm>

Resources:

[1] Delta IV Payload Planners Guide Oct. 2000. Boeing Company. 31 Jan. 2005. <http://www.boeing.com/defense-space/space/delta/guides.htm>

7 Cunha, Chris

7.1 Battery Code

Author: Christopher Cunha

The battery code compares numerous chemical reaction used in common batteries today. The code also explores various battery system configurations in order to achieve an optimum solution. A optimum solution being one that conserves the most mass.

```
function optimize
% Christopher Cunha
% Function Utilizes Battery function to help determine optimum volume/mass
%configurtion for a desired power
clear all
clc
%Li/SO2 lithium - sulfer dioxide
V_LiSO2=3.9;
Ahkg_LiSO2=379;
%d_LiSO2

%Li/MnO2 Lithium - Maganese dioxide
V_LiMno2=3.5;
Ahkg_LiMno2=286;
%d_LiSO2

%Li/SOCl2
V_LiSOCl2=3.605;
Ahkg_LiSOCl2=346;
%d_SOCl2

h=30; %Operation Duration, days times hours

for i = 1:20
    Kw(i)=i/2;
    %case 1
    [Wh(i), M1_LiSO2(i), M2_LiSO2(i), M4_LiSO2(i),
M2_ALT_LiSO2(i)]=Battery(Kw(i),h,V_LiSO2,Ahkg_LiSO2);
    %case 2
    [Wh(i), M1_LiMno2(i), M2_LiMno2(i), M4_LiMno2(i),
M2_ALT_LiMno2(i)]=Battery(Kw(i),h,V_LiMno2,Ahkg_LiMno2);
    %case 3
    [Wh(i), M1_LiSOCl2(i), M2_LiSOCl2(i), M4_LiSOCl2(i),
M2_ALT_LiSOCl2(i)]=Battery(Kw(i),h,V_LiSOCl2,Ahkg_LiSOCl2);

end

% %case1
figure;
hold on;
plot(Kw,M1_LiSO2,'bd');
plot(Kw,M2_LiSO2,'gd');
plot(Kw,M4_LiSO2,'rd');
plot(Kw,M2_ALT_LiSO2,'k+');
legend('1 Bat.','2 Bat. Series','4 Bat. Series', 'Fuel Cell Cycle ')
ylabel('Mass (kg)');
xlabel(Kw);
title('Power Config. LiSO2')
%
%
% %case 2
```

```
figure;
hold on;
plot(Kw,M1_LiMno2,'bd');
plot(Kw,M2_LiMno2,'gd');
plot(Kw,M4_LiMno2,'rd');
plot(Kw,M2_ALT_LiMno2,'k+');
legend('1 Bat.','2 Bat. Series','4 Bat. Series', 'Fuel Cell Cycle ')
ylabel('Mass (kg)');
xlabel(Kw);
title('Power Config. LiMno2')
%
% % case 3
figure;
hold on;
plot(Kw,M1_LiSOCl2,'bd');
plot(Kw,M2_LiSOCl2,'gd');
plot(Kw,M4_LiSOCl2,'rd');
plot(Kw,M2_ALT_LiSOCl2,'k+');
legend('1 Bat.','2 Bat. Series','4 Bat. Series', 'Fuel Cell Cycle ')
ylabel('Mass (kg)');
xlabel(Kw);
title('Power Config. LiSOCl2')
%This section allows for user to find exact value of mass for desired power
%requirment

M1_batt_LiSO2=M1_LiSO2
M4_batt_LiSO2=M4_LiSO2
M2_batt_LiSO2=M2_LiSO2(6)

M1_batt_LiMno2=M1_LiMno2(12)
M4_batt_LiMno2=M4_LiMno2(4)
M2_batt_LiMno2=M2_LiMno2(6)

M1_batt_LiSOCl2=M1_LiSOCl2(12)
M4_batt_LiSOCl2=M4_LiSOCl2(4)
M2_batt_LiSOCl2=M2_LiSOCl2(6)

Wh(10)

function [Wh,m1,m2,m4,M_Alt]= Battery(kw,h,V_Battery,Ah_kg,Ah_L)
%Battery Optimization Code, Calculates Watthour, Battery Masses, Volumes, for Variety of
%Batter/Fuel Cell Configurations

%clc

%System Requirments
% kw = kilowatts
% h = hours of operation duration
Wh=kw*1000*h; %watt Hour

%Single Battery
Ah=Wh/(V_Battery);
m1=Ah/Ah_kg;

%Series 4 Battery
Ah=Wh/(3*V_Battery);
m4=Ah/Ah_kg;

%Series 2 Battery
Ah=Wh/(2*V_Battery);
m2=Ah/Ah_kg;
% %Alternating Series .5 operation
%  %V_Fuel_Cell=1;
%  Wh=kw*1000*h/2;
%  %Ah=Wh./(2*V_Battery+V_Fuel_Cell);
%  Ah=Wh./(2*V_Battery);

 M_Alt=Ah./Ah_kg;
% Vol_Battery=(Ah./Ah_L)
% r=linspace(.01,1,100);
% h=.374./(2*pi.*r);
% figure
% plot(r,h)

return
```

8 Davis, Joseph T.

8.1.1 CTV Main Engines Design Process

Author(s): Joe Davis

8.1.1.1 Summary

For the design of the main engines of the CTV, nuclear thermal propulsion was settled on at a very early stage. These engines have a demonstrable feasibility, and I_{sp} values exceeding 1000 seconds. We had initially decided to work with a NERVA (nuclear engine for rocket vehicle application) design. NERVA was a long-term program performed at the Los Alamos National Laboratories in the 1960's and 70's. A graphite matrix core in which uranium fuel rods were imbedded, along with many fuel ports, would have hydrogen fuel run through. The heat flux from the controlled nuclear reaction heats the hydrogen fuel to temperatures approaching 2500 K, after which time the fuel is exhausted from a standard bell nozzle.

NERVA engines employ regenerative cooling for the nozzle and reactor vessel walls. The NERVA program developed reactors with powers ranging from 300 MW to 200,000 MW, thrust levels as high as 900,000 N, and I_{sp} as high as 850. Much analysis was done to develop a NERVA-derivative reactor for this mission (see Code 1). The equations used for analysis are the same as those found in the main body of the paper, in the Main Nuclear Engine section of the CTV technical report. The differences between the NERVA and the engine described in the report are that NERVA has a much higher density (2700 kg/m^3), and NERVA has a different shape trend (NERVA tends to be longer and skinnier).

A problem was observed with the NERVA rockets, however. These reactors tend to be very large in terms of heat production, and require more than just radiators and regenerative flow to cool them. Once the engine is cut off, a small amount of liquid hydrogen still needs to be run through the engines to cool them to a manageable temperature. There is a noticeable I_{sp} drop during this cycle (25% at the least). This causes not only a great deal of fuel to be required simply for the cooling, but it induces gravity losses from the drastically increased cooling times (on the order of 45,000 seconds). Code 2 and the following figures show the effect that the cooling cycle has on the behavior of the NERVA engines.

The inherent problem with the NERVA engines was that they heat up the entire core to the operational temperature. All that needs to be heated to this temperature is the fuel itself. As such, a more thermally efficient engine is desirable, one that can be cooled in a traditional sense (heated fluids, heat exchangers, radiators, etc.). One such design into which a great amount of research has gone is the particle bed reactor. Refer again to the main section of the paper for a detailed description of this engine and the analysis used to design it for our pruposes. Code 3 was used to determine power levels and core sizes.

The particle bed reactor is desirable for several reasons. It is well-developed (theoretically), its unique use of fuel pellets allows for a temperature gradient highly conducive to thermal control, and the reactor cores themselves can be smaller than the minimum NERVA engines. The PBR has a smaller mass density and a higher power density, making a much smaller engine with comparable performance. It also has a higher thrust to weight ratio, a noticeably higher I_{sp}, due to its higher operational fuel temperature (~3200 K). Many particle bed reactor concepts have been proposed, including the MITEE engines proposed by the NASA-endorsed Plus Ultra Technologies.

While the NERVA and PBR reactors are the most feasible near-term propulsive options, there are many more possible theoretical nuclear engines. Even discounting such ideas as Medusa or Orion engines (using pulsed nuclear explosions to propel a craft), there are radical concepts such as gas core nuclear rockets, nuclear lightbulb engines (a solid-core gas-core hybrid), and more advanced, higher-power-density solid core designs. For feasibility and economic issues, however, the oft-researched particle bad was chosen.

8.1.1.2 Major Equations Used:

$$\gamma = \frac{c_p}{c_p - \frac{R_u}{\mathrm{M}}}$$

$$c_{pH_2} = \frac{1}{\mathrm{M}}\left[56.505 - 702.74\left(\frac{T}{100}\right)^{-0.75} + \frac{116500}{T} - 560.7\left(\frac{T}{100}\right)^{1.5}\right]$$

$$c^* = \frac{\sqrt{\gamma R_u T}}{\gamma\sqrt{\left[2/(\gamma+1)\right]^{\left(\frac{\gamma+1}{\gamma-1}\right)}}}$$

$$Isp = \frac{C_F c^*}{g_0}$$

$$\dot{m}_{prop} = \frac{F}{I_{sp} g_0}$$

$$P_c = \frac{F}{C_F A_t}$$

$$Pcore = \dot{m}_{prop}\left(h_v + \int_{T_1}^{T_2} c_p dt\right) = \dot{m}_{prop}\left(0.018061T_2 - 5.715417\right)$$

$$V_{core} = P_{core}\left(6.4*10^{-19} t_b + \frac{1}{P_D}\right)$$

$$m_{core} = \rho_{core} V_{core}$$

$$R_{core} = 4.905*10^{-11} P_{core}{}^4 - 2.881*10^{-7} P_{core}{}^3 + 6.2522*10^{-4} P_{core}{}^2 - 0.5992 P_{core} + 252.28$$

$$Hcore = -6.502*10^{-6} P_{core}{}^2 + 0.05009 P_{core} + 18.335$$

8.1.1.3 Variable Index:

γ	Isentropic parameter
c_p	Specific heat
R_u	Universal gas constant
M	Molar mass
T	Temperature
c*	Characteristic Velocity
C_F	Thrust coefficient
I_{sp}	Specific Impulse
g_0	Terrestrial gravity constant
$\dot{m}_{prop}$	Propellant mass flow rate
F	Thrust
P_c	Chamber pressure
A_t	Throat area
P_{core}	Core power
h_v	Heat of vaporization
V_{core}	Core volume
t_b	Engine burn time
P_D	Core power density
ρ_{core}	Core density
m_{core}	Core mass
R_{core}	Core radius
H_{core}	Core height

8.1.1.4 Code 1: Thermochemical and Sizing Analysis of a NERVA-Derivative Engine

```
% Nuclear engine Isp and sizing code for NERVA style engine
clc
clear
Pa = 0; Pe = 0;
MolMass = 2.018;
Pcmin = 100000; Pcmax = 25e6;
Tcmin = 300; Tcmax = 3000;
MassFullCTV = 171340+150000; % early estimates
MassEmptyCTV = 150000; % early estimates
go = 9.8066;
x = 5; %number of engines
lambda = 0.97;
F = MassFullCTV*2/x; %minimum acceleration 2 m/s^2
R = 8314.3;

Tc = 3000;
Pc = 5e6;
Tt = Tc/100;
Cpp = 1000*(56.505 - 702.74*Tt^(-0.75) + 1165/Tt - 560.7*Tt^(-1.5))/MolMass;
gammag = Cpp/(Cpp-R/MolMass); %  gamma  = 1.28888
cstar = sqrt(gammag*R*Tc/MolMass)/(gammag*((2/(gammag+1))^((gammag+1)/(2*gammag-2))));

Cf = 1.83566;
Isp = cstar*Cf/go
eps = 100;

De = 2.5;  % choose nozzle exit diameter (for payload capabilities)
Ae = pi*(De/2)^2; % calc nozzle area
At = Ae/100; % throat area from nozzle area and expansion ratio

Pc = F/(Cf*At) % F = Cf*Pc*At (sutton equation 3-31)

mdot = F/(Isp*go) %per engine (Sutton eqn 2-5)

% Sizing of the nuclear thermal chamber is done empirically, based on data in
%   "Space Propulsion Analysis and Design" by Humble, Henry, and Larson

% Mach_Exit = solve('100 = ((2/(1.28888+1)*(1+(1.28888-1)*(M^2)/2))^((1.28888+1)/(2*(1.28888-1))))');
% Mach_Exit = double(Mach_Exit)

Mx = 1;
LHS = eps;
RHS = sqrt(((2/(gammag+1))*(1+(gammag-1)/2*Mx^2))^((gammag+1)/(gammag-1)))/Mx;
while abs((RHS-LHS)/RHS) > 0.001
    Mx = Mx + 0.001;
    LHS = eps;
    RHS = sqrt(((1+(gammag-1)/2*Mx^2)/(1+(gammag-1)/2))^((gammag+1)/(gammag-1)))/Mx;
end
Mach_Exit = Mx
Pratio = (1+(gammag-1)*Mach_Exit^2/2)^(gammag/(1-gammag));
Pe = Pc*Pratio
Power = (.018061*Tc-5.715417)*mdot %MW
PowerWatts = Power*10^6; %W
DeltaV = 3.72*1000*2+100;
tburn = 1900;
Vcore = PowerWatts*(6.4e-19*tburn + 1/1570000000)
Mreactor = 2300*Vcore
Hreactor = 0.7;
Rreactor = sqrt(Vcore/(pi*Hreactor));
Deltav = 5000:50:20000;
for k = 1:301
    mratio(k) = exp(-Deltav(k)/(Isp*go));
    Mass_Prop(k) = MassEmptyCTV/mratio(k) - MassEmptyCTV;
end

% plot(Deltav,Mass_Prop)
% title('Propellant Mass vs. Delta V')
% xlabel('Delta V (m/s)')
% ylabel('Propellant Mass (kg)')

go = 9.8066;
c = Isp*go;
InertMass = 150000;
dve2m = 956; % earth to mars del v
dvTCM1 = 100; % earth to mars TCM
dvatm = 4519; % mars insertion
dvm2e = 886; % mars to earth del v
dvTCM2 = 100; % mars to earth TCM
dvate = 830; %earth insertion

dvtotal = dve2m+dvTCM1+dvatm+dvm2e+dvTCM2+dvate;
MassPropTotal = (exp(dvtotal/c)-1)*InertMass;
MassTotal = InertMass+MassPropTotal;
MassProp1 = MassTotal*(1-1/exp(dve2m/c));
MassProp2 = (MassTotal - MassProp1)*(1-1/exp(dvTCM1/c));
MassProp3 = (MassTotal - MassProp1 - MassProp2)*(1-1/exp(dvatm/c));
MassProp4 = (MassTotal - MassProp1 - MassProp2 - MassProp3)*(1-1/exp(dvm2e/c));
MassProp5 = (MassTotal - MassProp1 - MassProp2 - MassProp3 - MassProp4)*(1-1/exp(dvTCM2/c));
MassProp6 = (MassTotal - MassProp1 - MassProp2 - MassProp3 - MassProp4 - MassProp5)*(1-
1/exp(dvate/c));
%
% k = 1;
% t(k) = k-1;
% M(k) = MassTotal;
% a(k) = F*x/M(k);
% while M(k) >= MassTotal - MassProp1
%     k = k+1;
%     t(k) = k-1;
%     M(k) = M(k-1) - x*mdot;
%     a(k) = F*x/M(k);
%     if a(k)>4
%         x = x-1;
%     end
%     if x == 0
%         x = 1;
%     end
% end
% endtime(1) = t(k);
% endmass(1) = M(k);
% endaccel(1) = a(k);
%
% while M(k) >= MassTotal - MassProp1 - MassProp2
%     k = k+1;
%     t(k) = k-1;
%     M(k) = M(k-1) - x*mdot;
%     a(k) = F*x/M(k);
%     if a(k)>4
%         x = x-1;
%     end
%     if x == 0
%         x = 1;
%     end
% end
% endtime(2) = t(k);
% endmass(2) = M(k);
% endaccel(2) = a(k);
%
% while M(k) >= MassTotal - MassProp1 - MassProp2 - MassProp3
%     k = k+1;
%     t(k) = k-1;
%     M(k) = M(k-1) - x*mdot;
%     a(k) = F*x/M(k);
%     if a(k)>4
%         x = x-1;
%     end
%     if x == 0
%         x = 1;
%     end
% end
% endtime(3) = t(k);
% endmass(3) = M(k);
% endaccel(3) = a(k);
%
% while M(k) >= MassTotal - MassProp1 - MassProp2 - MassProp3 - MassProp4
%     k = k+1;
%     t(k) = k-1;
%     M(k) = M(k-1) - x*mdot;
%     a(k) = F*x/M(k);
%     if a(k)>4
%         x = x-1;
%     end
%     if x == 0
%         x = 1;
%     end
% end
% endtime(4) = t(k);
% endmass(4) = M(k);
% endaccel(4) = a(k);
%
% while M(k) >= MassTotal - MassProp1 - MassProp2 - MassProp3 - MassProp4 - MassProp5
%     k = k+1;
%     t(k) = k-1;
%     M(k) = M(k-1) - x*mdot;
%     a(k) = F*x/M(k);
%     if a(k)>4
%         x = x-1;
%     end
%     if x == 0
%         x = 1;
%     end
% end
% endtime(5) = t(k);
% endmass(5) = M(k);
% endaccel(5) = a(k);
%
% while M(k) >= MassTotal - MassProp1 - MassProp2 - MassProp3 - MassProp4 - MassProp5 - MassProp6
%     k = k+1;
%     t(k) = k-1;
%     M(k) = M(k-1) - x*mdot;
%     a(k) = F*x/M(k);
%     if a(k)>4
%         x = x-1;
%     end
%     if x == 0
%         x = 1;
%     end
% end
% endtime(6) = t(k);
% endmass(6) = M(k);
% endaccel(6) = a(k);
%
% figure(1)
%
plot(t,M,endtime(1),endmass(1),'o',endtime(2),endmass(2),'o',endtime(3),endmass(3),'o',endtime(4),endm
ass(4),'o',endtime(5),endmass(5),'o',endtime(6),endmass(6),'o')
% title('Mass versus Engine Burn Time')
% xlabel('Time [sec]')
% ylabel('Vehicle Mass [kg]')
%
% figure(2)
%
plot(t,a,endtime(1),endaccel(1),'o',endtime(2),endaccel(2),'o',endtime(3),endaccel(3),'o',endtime(4),e
ndaccel(4),'o',endtime(5),endaccel(5),'o',endtime(6),endaccel(6),'o')
% title('Acceleration versus Engine Burn Time')
% xlabel('Time [sec]')
% ylabel('Vehicle Acceleration [m/s^2]')
```

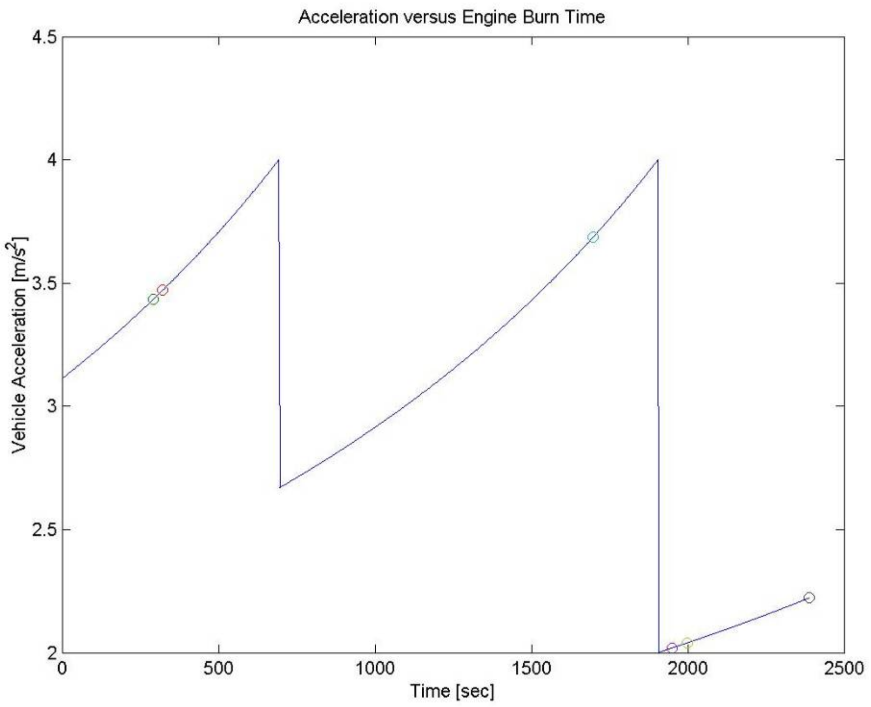


Figure 1 – Acceleration versus burn time for a NERVA-derivative rocket engine (2 < acceleration < 4 m/s^2); without cooling

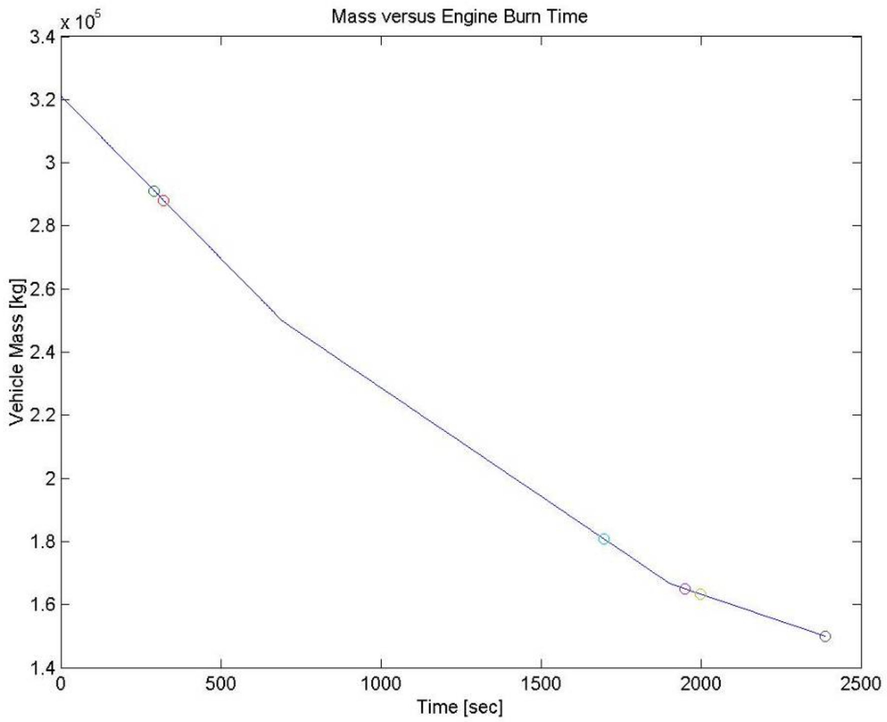


Figure 2 – Total CTV mass (from early estimates) versus main engine burn time; without cooling

8.1.1.5 Code 2: NERVA-Derivative Engine Analysis with Cooling Fuel Bleed

```
% Nuclear engine Isp and sizing code for NERVA style engine
clc
clear
MolMass = 2.018;
Pcmin = 100000; Pcmax = 25e6;
Tcmin = 300; Tcmax = 3000;
MassFullCTV = 171340+150000;
MassEmptyCTV = 150000;
go = 9.8066;
x = 3; %number of engines
lambda = 0.97;
F = 30000;
R = 8314.3;

Tc = 3000;
Pc = 5e6;
Tt = Tc/100;
Cpp = 1000*(56.505 - 702.74*Tt^(-0.75) + 1165/Tt - 560.7*Tt^(-1.5))/MolMass;
gammag = Cpp/(Cpp-R/MolMass); %  gamma  = 1.28888
cstar = sqrt(gammag*R*Tc/MolMass)/(gammag*((2/(gammag+1))^((gammag+1)/(2*gammag-2))));

Cf = 1.83566;
Isp = cstar*Cf/go
eps = 100;

De = 2.5;  % choose nozzle exit diameter (for payload capabilities)
Ae = pi*(De/2)^2; % calc nozzle area
At = Ae/100; % throat area from nozzle area and expansion ratio
Pc = F/(Cf*At) % F = Cf*Pc*At (sutton equation 3-31)
mdot = F/(Isp*go) %per engine (Sutton eqn 2-5)
% Sizing of the nuclear thermal chamber is done empirically, based on data in
%   "Space Propulsion Analysis and Design" by Humble, Henry, and Larson
% Mach_Exit = solve('100 = ((2/(1.28888+1)*(1+(1.28888-1)*(M^2)/2))^((1.28888+1)/(2*(1.28888-1))))');
% Mach_Exit = double(Mach_Exit)
Mx = 1;
LHS = eps;
RHS = sqrt(((2/(gammag+1))*(1+(gammag-1)/2*Mx^2))^((gammag+1)/(gammag-1)))/Mx;
while abs((RHS-LHS)/RHS) > 0.001
    Mx = Mx + 0.001;
    LHS = eps;
    RHS = sqrt(((1+(gammag-1)/2*Mx^2)/(1+(gammag-1)/2))^((gammag+1)/(gammag-1)))/Mx;
end
Mach_Exit = Mx

Pratio = (1+(gammag-1)*Mach_Exit^2/2)^(gammag/(1-gammag));
Pe = Pc*Pratio

% Power = (.018061*Tc-5.715417)*mdot %MW
% PowerWatts = Power*10^6; %W
%
% DeltaV = 3.72*1000*2+100;
% tburn = 22500;
% Vcore = PowerWatts*(6.4e-19*tburn + 1/1570000000)
% Mreactor = 2300*Vcore
%
% Hreactor = 0.7
% Rreactor = sqrt(Vcore/(pi*Hreactor));
%
Rcore = 0.4
Hcore = 1.3
Vcore = pi*Rcore^2*Hcore
Mcore = 2300*Vcore
Mshielding = Rcore^2 * pi * 3500
Power = 2000000000*Vcore
c = Isp*go;
InertMass = 150000;
dve2m = 956; % earth to mars del v
dvatm = 4519; % mars insertion
dvm2e = 886; % mars to earth del v
```

```
    dvate = 830; %earth insertion
    dvtotal = dve2m+dvatm+dvm2e+dvate;
    mcooling = 20873;

    % MassTotal = InertMass+MassPropTotal;
    %
    % MassProp1 = MassTotal*(1-1/exp(dve2m/c));
    % MassProp2 = (MassTotal - MassProp1)*(1-1/exp(dvatm/c));
    % MassProp3 = (MassTotal - MassProp1 - MassProp2)*(1-1/exp(dvm2e/c));
    % MassProp4 = (MassTotal - MassProp1 - MassProp2 - MassProp3)*(1-1/exp(dvate/c));

    MassProp4 = (1-1/exp(dvate/c))*(InertMass+1*mcooling);
    MassProp3 = (1-1/exp(dvm2e/c))*(MassProp4+InertMass+2*mcooling);
    MassProp2 = (1-1/exp(dvatm/c))*(MassProp4+MassProp3+InertMass+3*mcooling);
    MassProp1 = (1-1/exp(dve2m/c))*(MassProp4+MassProp3+MassProp2+InertMass+4*mcooling);
    MassTotal = MassProp4+MassProp3+MassProp2+MassProp1+InertMass+4*mcooling;
    k = 1;
    t(k) = k-1;
    M(k) = MassTotal;
    a(k) = F*x/M(k);
    while M(k) >= MassTotal - MassProp1
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot;
        a(k) = F*x/M(k);
        if a(k)>4
            x = x-1;
        end
        if x == 0
            x = 1;
        end
        dvsofar = Isp*go*log(MassTotal/(M(k)));
        if dve2m - dvsofar < 750*go*log(M(k)/(M(k)-mcooling)) break
        end
    end
    endtime(1) = t(k);
    endmass(1) = M(k);
    endaccel(1) = a(k);
    for ct = 1:45000
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot/20;
        a(k) = (750*x*mdot/20)/M(k);
    end
    ccc = 1
    while M(k) >= MassTotal - MassProp1 - MassProp2 - mcooling
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot;
        a(k) = F*x/M(k);
        if a(k)>4
            x = x-1;
        end
        if x == 0
            x = 1;
        end
        dvsofar = Isp*go*log((MassTotal-MassProp1-mcooling)/(M(k)));
        if dvatm - dvsofar < 750*go*log(M(k)/(M(k)-mcooling)) break
        end
    end
    endtime(2) = t(k);
    endmass(2) = M(k);
    endaccel(2) = a(k);
    for ct = 1:45000
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot/20;
        a(k) = (750*x*mdot/20)/M(k);
    end
    ccc = 2
    while M(k) >= MassTotal - MassProp1 - MassProp2 - MassProp3 - mcooling*2
```

```
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot;
        a(k) = F*x/M(k);
        if a(k)>4
            x = x-1;
        end
        if x == 0
            x = 1;
        end
        dvsofar = Isp*go*log((MassTotal-MassProp1-MassProp2-mcooling*2)/(M(k)));
        if dvm2e - dvsofar < 750*go*log(M(k)/(M(k)-mcooling)) break
        end
    end
    endtime(3) = t(k);
    endmass(3) = M(k);
    endaccel(3) = a(k);
    for ct = 1:45000
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot/20;
        a(k) = (750*x*mdot/20)/M(k);
    end
    ccc = 3
    while M(k) >= MassTotal - MassProp1 - MassProp2 - MassProp3 - MassProp4 - mcooling*3
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot;
        a(k) = F*x/M(k);
        if a(k)>4
            x = x-1;
        end
        if x == 0
            x = 1;
        end
        dvsofar = Isp*go*log((MassTotal-MassProp1-MassProp2-MassProp3-mcooling*3)/(M(k)));
        if dvate - dvsofar < 750*go*log(M(k)/(M(k)-mcooling)) break
        end
    end
    endtime(4) = t(k);
    endmass(4) = M(k);
    endaccel(4) = a(k);
    for ct = 1:45000
        k = k+1;
        t(k) = k-1;
        M(k) = M(k-1) - x*mdot/20;
        a(k) = (750*x*mdot/20)/M(k);
    end
    ccc = 4
    %
    % figure(1)
    %
    plot(t,M,endtime(1),endmass(1),'o',endtime(2),endmass(2),'o',endtime(3),endmass(3),'o',endtime(4),endm
    ass(4),'o',endtime(5),endmass(5),'o',endtime(6),endmass(6),'o')
    % title('Mass versus Engine Burn Time')
    % xlabel('Time [sec]')
    % ylabel('Vehicle Mass [kg]')
    %
    % figure(2)
    %
    plot(t,a,endtime(1),endaccel(1),'o',endtime(2),endaccel(2),'o',endtime(3),endaccel(3),'o',endtime(4),e
    ndaccel(4),'o',endtime(5),endaccel(5),'o',endtime(6),endaccel(6),'o')
    % title('Acceleration versus Engine Burn Time')
    % xlabel('Time [sec]')
    % ylabel('Vehicle Acceleration [m/s^2]')
```

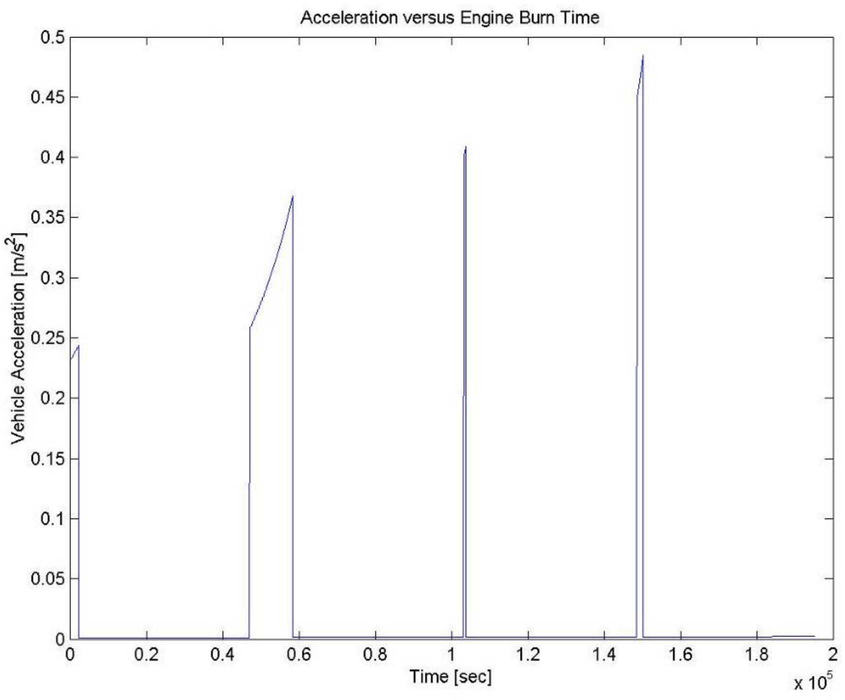


Figure 3 – CTV acceleration versus time (note the acceleration during the cooling phases)

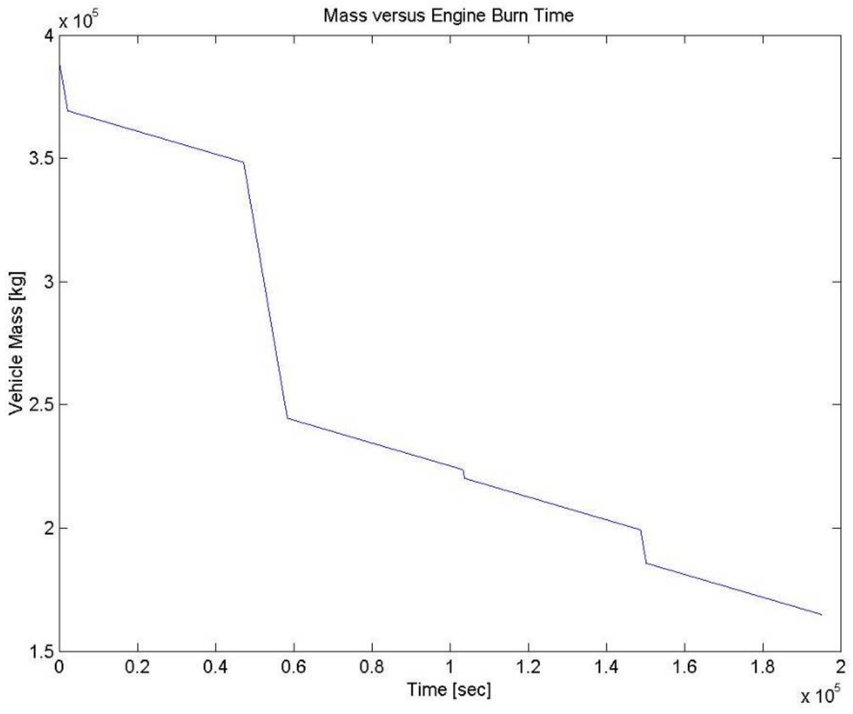


Figure 4 – Total CTV mass versus burn time (note the required time and shallow mass drops for the cooling cycles)

8.1.1.6 Code 3: Sizing Analysis of the Particle Bed Reactor

```
% Nuclear engine Isp and sizing code for NERVA style engine
clc
clear
MolMass = 2.018;
go = 9.8066;
%number of engines
x = 5;
lambda = 0.97;
F = 200000;
R = 8314.3;

Tc = 3196;
Tt = Tc/100;
Cpp = 1000*(56.505 - 702.74*Tt^(-0.75) + 1165/Tt - 560.7*Tt^(-1.5))/MolMass;
gammag = Cpp/(Cpp-R/MolMass); %  gamma  = 1.28888
cstar = sqrt(gammag*R*Tc/MolMass)/(gammag*((2/(gammag+1))^((gammag+1)/(2*gammag-2))));

Cf = 1.83566;
Isp = cstar*Cf/go % [sec]
eps = 100;

De = 1;  % choose nozzle exit diameter (for payload capabilities)
Ae = pi*(De/2)^2; % calc nozzle area
At = Ae/100; % throat area from nozzle area and expansion ratio
Pc = F/(Cf*At) % F = Cf*Pc*At (sutton equation 3-31)
mdot = F/(Isp*go) %per engine (Sutton eqn 2-5)

% Sizing of the nuclear thermal chamber is done empirically, based on data in
%   "Space Propulsion Analysis and Design" by Humble, Henry, and Larson
% Mach_Exit = solve('100 = ((2/(1.28888+1)*(1+(1.28888-1)*(M^2)/2))^((1.28888+1)/(2*(1.28888-1))))');
% Mach_Exit = double(Mach_Exit)

Mx = 1;
LHS = eps;
RHS = sqrt(((2/(gammag+1))*(1+(gammag-1)/2*Mx^2))^((gammag+1)/(gammag-1)))/Mx;
while abs((RHS-LHS)/RHS) > 0.001
    Mx = Mx + 0.001;
    LHS = eps;
    RHS = sqrt(((1+(gammag-1)/2*Mx^2)/(1+(gammag-1)/2))^((gammag+1)/(gammag-1)))/Mx;
end
Mach_Exit = Mx
Pratio = (1+(gammag-1)*Mach_Exit^2/2)^(gammag/(1-gammag));
Pe = Pc*Pratio
Power = (.018061*Tc-5.715417)*mdot %MW
PowerWatts = Power*10^6; %W
tburn = 50*3600;

if Power < 600
    Rcore = 9.0958e-10*Power^4 - 1.3261e-6*Power^3 + 7.1665e-4*Power^2 - 0.1735*Power + 47.625
    Hcore = -0.000283*Power^2 + 0.5203*Power + 26.06
    Num_Elements = 7;
else
    Num_Elements = 37;
    Rcore = 4.905e-11*Power^4 - 2.881e-7*Power^3 + 6.2522e-4*Power^2 - 0.5992*Power + 252.28 % core
radius [m]
    Hcore = -6.502e-6*Power^2 + 0.05009*Power + 18.335 % core "height" [m]
end
Vcore = pi*Rcore^2*Hcore/1000000 %Core volume [m^3]
Mcore = 1600*Vcore % core mass
Lnozzle = 0.45/tan(15*pi/180) % nozzle length [m]
NozzleStrength = 310000000; %Pa - material property
NozzleDensity = 8500; %kg/m^3 - material property
rt = sqrt(At/pi);
NozzleThicknessThroat = 3*Pc*rt/NozzleStrength;

f1 = NozzleThicknessThroat/Lnozzle;
f2 = 0.45/Lnozzle;

% ALL MASSES ARE IN KILOGRAMS
```

```
Mnozzle = 2*pi*NozzleDensity*Lnozzle*((f1*f2*Lnozzle^2)/3 +
0.5*(f1*rt+f2*NozzleThicknessThroat)*Lnozzle + rt*NozzleThicknessThroat) % mass of nozzle
Mvessel = 2*Pc*Vcore/(9.81*2500) %mass of core container
Msystem = (Mnozzle+Mvessel)/0.4; % mas of support system (includes structure, cooling, feed)
Mfeedsystem = Msystem * 0.249 % mass of feed system (turbopumps, etc.)
Mcooling = Msystem * 0.351 % mass of cooling system
Msupport = Msystem - Mfeedsystem - Mcooling % mass of structural support for engines (gimbals, etc.)
Mtotal = (Mcore+Msystem)*1.1 %mass of everything except rad shielding
```

9 Brendan Eash

9.1 Communications Link Budget Mass vs. Power Trade Study

Author: Brendan Eash

Contributors: Jeri Lynn Metzger, Zade Shaw, Justin McCurdy

The communications link budget was defined using the theory equations found in the CTV Antenna Hardware and Design section of the report.

We decided that having a link budget margin of 2dB with a minimum required signal to noise ratio (SNR) of 1dB would be enough to give us significant leeway in our design. To this end, we set the SNR_{avail} to 3dB to determine the power required for various sizes of primary communication dishes. In the initial link budget code (other versions were modified by Jeri Metzger and Zade Shaw which contain more detailed information for specific link budgets), we input several various sizes of transmitting dishes at various power levels to find what we considered the optimum solution. Figure 9-1 below is an example of this type of output.

9.1.1 Communication Codes

The following codes were used to develop the link budget for the CTV.

9.1.1.1 link_budget.m

This code shows parametric studies of the power required for a specific transmission data rate for various dish sizes. There are several fixed inputs, two of which are swept. First, we define several sizes of dish diameters to do a parametric study of. Then the available signal to noise ratio is found for a given transmission power and compared to the required signal to noise ratio. Figure 9-1 shows an example output file of this study.

9.1.1.2 Code

```
clear all; close all; clc; format long; format compact;

Rd = 10*log10(100e6); %bps - data rate
f = 31.977; %Ka-band, GHz
% f = 8.145; %X-band, GHz
am = 2.27936636e8; %semi-major axis Mars, km
ae = 1.49597807e8; %semi-major axis Earth, km
S = am + ae %Earth-Mars Distance, km
S = 3.985538969629583e+008 %Earth-Mars Distance, worst case based on exclusion angles, km
%S will need to be changed, for the time being its set to the
%approx. opposition value.  This value is assumed to be worst case.
p_min = 1; %power value in kW
p_max = 100; %power value in kW
POWER = [p_min*1e3:1e3:p_max*1e3]; %Transmission Power, W
etaR = .55; %receiving antenna efficency
```

```
etaT = .55; %transmitting antenna efficency
Dr = 20; %diameter of receiving antenna, m
Dt = [0:5:40]; %diameter of transmitting antenna, m
T = 150; %noise equivalent temperature of receiving system (K)

for i = 1:length(Dt)
    Dt(1) = 1;
    for j = 1:length(POWER)
           GT = 20.4 + 20*log10(Dt(i)) + 20 * log10(f) + 10*log10(etaT); %gain of transmission antenna
           EIRP = 10*log10(POWER(j)) + GT; %effective isotropic radiated power
           L_space = 92.45 + 20*log10(f) + 20*log10(S); %space loss
           L_other = 3; %losses due to ineffeciences in system
           L_atm = 6; %atmospheric loss
           k = -228.6; %Boltzmann's constant, dB
           GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f); %gain of recieving antenna
           T_dB = 10 * log10(T);
           G_T = GR - T_dB;
           SNR_avial(i,j) = EIRP + G_T - L_space - L_other - k - Rd - L_atm;
    end
end


figure; hold on;
for i = 1:length(Dt)
    POWER_plot(i,:) = POWER/1e3;
end
plot(POWER_plot',SNR_avial');
plot([0 p_max],[3 3],'r-.');
plot([0 p_max],[1 1],'r--');

xlabel('Power (kW)');
ylabel('S/N Ratio (dB)');
legend('1m Dia.','5m Dia.','10m Dia.','15m Dia.','20m Dia.',...
    '25m Dia.','30m Dia.','35m Dia.','40m Dia.','Ideal S/N','Min. S/N',-1);
ttl = sprintf('Transmitting Antenna Power Required for %im Recieving Antenna',Dr);
title(ttl);

%Created by:  Brendan Eash,  Jan. 23, 2005
%Modified by:  Brendan Eash, Jan. 24, 2005
%Modified by:  Brendan Eash, Jan. 26, 2005
%Fixed by:  Brendan Eash, Feb. 7, 2005
```

9.1.1.3 Code Output and Trade Study Results

This code outputs Figure 9-1 below. Based on the results of this analysis, we decided that the primary communications dish should be on the order of 5-10 meters in diameter. This size range gives us a reasonable power draw on the without compromising too much on mass and pointing requirements.

The primary communications dish is designed to be universal in the sense that every platform which may need to be in contact with Earth can use this dish for that purpose. This dish can also be used to transmit in another band, primarily the X-band, for satellite to CTV communications. This capability must exist since the only transponders for deep space use are capable of receiving in the X-band.

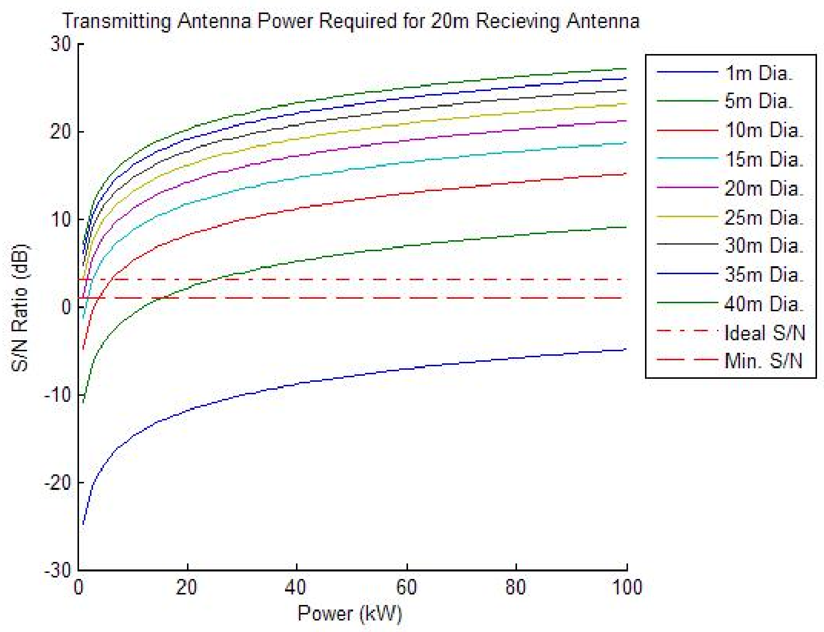


Figure 9-1: Transmitting Dish Size vs. Power Trade Study

9.2 Communications Coverage Analysis

Author: Brendan Eash

Contributor: Jeri Lynn Metzger

During the course of investigating the various issues involved with communications, we found that the Sun happens to get in the way of communication line of sight (LOS). To find the length of time that there would be a communications blackout, we needed to find the angles between the Earth, Sun and the Sun and Mars. Figure 9-2 and Figure 9-3 show the geometry of the two positions where this blackout occurs.

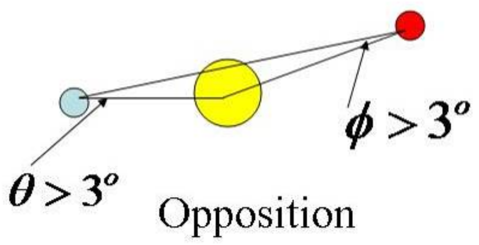


Figure 9-2: Opposition Geometry (Brendan Eash)

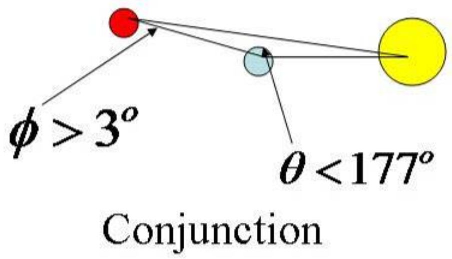


Figure 9-3: Conjunction Geometry (Brendan Eash)

In Figure 9-3 and Figure 9-3, the large ball represents the sun, the outermost circle is Mars and the remaining circle is Earth. The drawings are not made to scale.

Once we had these angles over the duration of a mission, we determined when the communications blackouts occurred using the guideline that communications was lost within three degrees of the Sun. Our analysis shows that while there is no communications blackout during 'mission critical times,' i.e. Mars Arrival, Mars landing, etc, there is an approximate 30 day blackout period which occurs while the astronauts are on the surface of Mars (during a nominal mission).

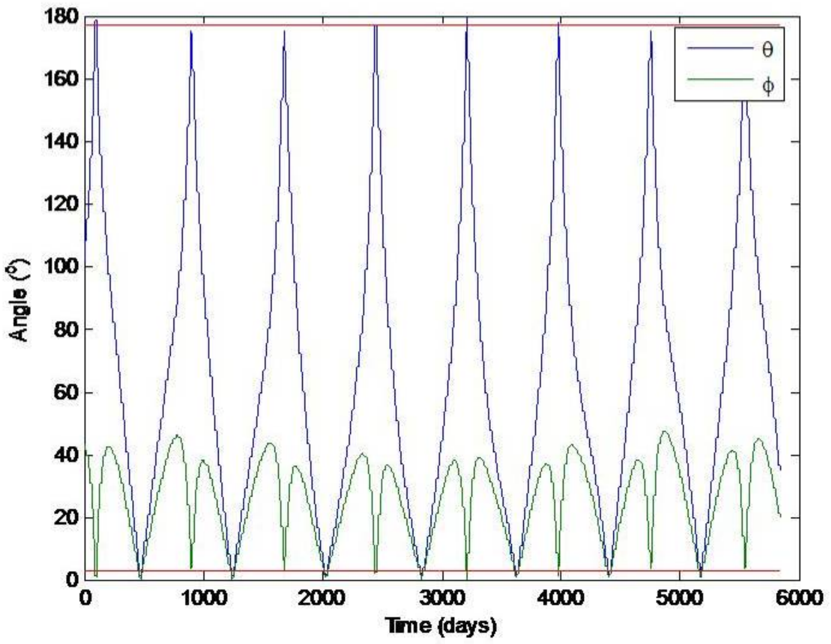


Figure 9-4: Communications Blackout Times

Figure 9-4 shows the blackout periods for the entirety of the program (2014 – 2030). Opposition occurs when the 'peaks' have the same apparent magnitude while conjunction occurs when the 'peaks' have discordant magnitudes.

The result of this study is the creation of the Mars Orbiting Relay Satellites (MORS) and the Heliocentric Relay Satellite (HRS). These satellites give continuous communications coverage as well as provide navigation and tracking for the Mars Lander Vehicle (MLV).

9.2.1 Communications Coverage Codes

These codes were used to find the length of time of the blackout at Mars.

9.2.1.1 coverage.m

This code produces the coverage analysis for the blackout periods between Earth and Mars.

9.2.1.1.1 Code

```
function [time,phi0,phi1] = coverage(xyz0,xyz1);

% Coverage Tool

rng = comm_range(xyz0,xyz1);
r0 = comm_range(xyz0,[0 0 0]);
r1 = comm_range(xyz1,[0 0 0]);

phi0 = rad2deg(acos((r0.^2+rng.^2-r1.^2)/2./r0./rng));
phi1 = rad2deg(acos((r1.^2+rng.^2-r0.^2)/2./r1./rng));
time = 1:length(rng);
```

9.2.1.1.2 Code Output

This code outputs a time vector and two vectors of angle based on two input xyz triple position vectors.

9.2.1.2 comm_range.m

This code finds the line of sight distance for communication (no bending effects are taken into account) between two locations in space defined by a xyz Cartesian triple.

9.2.1.2.1 Code

```
function r = comm_range(xyz,xyz1)

%comm_range finds the LOS distance for communication between two locations
%given as xyz and xyz1 sets of cartesian triples.  these vectors are
%expected in column format (Nx3).

x0 = xyz(:,1); y0 = xyz(:,2); z0 = xyz(:,3);
x1 = xyz1(:,1); y1 = xyz1(:,2); z1 = xyz1(:,3);

r = sqrt((x0-x1).^2+(y0-y1).^2+(z0-z1).^2);
```

9.2.1.2.2 Code Output

This code finds the range distance measurement based on an input xyz position triple. This triple can be a matrix with 3 columns, one for each x, y, and z. The number of rows is only constrained by the limits of Matlab.

9.2.1.3 Planetary_orbit_propagator.m

This code creates a file which contains the xyz position history for Earth and Mars starting in 2014 and ending in 2030.

9.2.1.3.1 Code

```
% 3-D Planetary Orbit Propegator

clear; close all; clc; format long; format compact; warning off;

mus = 1.3271244e11;
[ae,ie,ee,om_earth,Om_earth,am,im,em,om_mars,Om_mars] = elems;
t_day = 24*3600; %conversion from days to seconds

% 2014 start date Earth and Mars positions
ts_i_earth = atan(.9675/-.1756) + om_earth;
E_i_earth = 2*atan(sqrt((1-ee)/(1+ee))*tan(ts_i_earth/2));
ts_i_mars = atan(.6966/-1.5125) + om_mars;
E_i_mars = 2*atan(sqrt((1-em)/(1+em))*tan(ts_i_mars/2));

for i = 1:365*16 %# of days
    TOFoffset_earth = 1.975192706237976e+006; %sec
    TOFoffset_mars = -6.748694712149173e+006; %sec
    try
        E_earth(i) = fsolve('kepler',E_earth(i-1),[],i*t_day+TOFoffset_earth,mus,ae,ee);
        ts_earth = 2*atan(sqrt((1+ee)/(1-ee))*tan(E_earth(i)/2));
        P_earth = ae*(1-ee^2);
        r_earth = P_earth ./ (1+ee*cos(ts_earth));
        trans_earth = xyz2rthh(ie,Om_earth,om_earth+ts_earth,'r')';
        xyz_earth(i,:) = [r_earth 0 0]*trans_earth;

        E_mars(i) = fsolve('kepler',E_mars(i-1),[],i*t_day+TOFoffset_mars,mus,am,em);
        ts_mars = 2*atan(sqrt((1+em)/(1-em))*tan(E_mars(i)/2));
        P_mars = am*(1-em^2);
        r_mars = P_mars ./ (1+em*cos(ts_mars));
        trans_mars = xyz2rthh(im,Om_mars,ts_mars+om_mars,'r')';
        xyz_mars(i,:) = [r_mars 0 0]*trans_mars;
    catch
        E_earth(i) = fsolve('kepler',E_i_earth,[],i*t_day+TOFoffset_earth,mus,ae,ee);
        ts_earth = 2*atan(sqrt((1+ee)/(1-ee))*tan(E_earth(i)/2));
        P_earth = ae*(1-ee^2);
        r_earth = P_earth ./ (1+ee*cos(ts_earth));
        trans_earth = xyz2rthh(ie,Om_earth,om_earth+ts_earth,'r')';
        xyz_earth(i,:) = [r_earth 0 0]*trans_earth;

        E_mars(i) = fsolve('kepler',E_i_mars,[],i*t_day+TOFoffset_mars,mus,am,em);
        ts_mars = 2*atan(sqrt((1+em)/(1-em))*tan(E_mars(i)/2));
        P_mars = am*(1-em^2);
        r_mars = P_mars ./ (1+em*cos(ts_mars));
        trans_mars = xyz2rthh(im,Om_mars,ts_mars+om_mars,'r')';
        xyz_mars(i,:) = [r_mars 0 0]*trans_mars;
    end
end

cd SatData2_6_05
save PlanetaryData xyz_earth xyz_mars
cd ..
```

9.2.1.3.2 Code Output

This code finds the xyz position history of Earth and Mars starting in 2014 and ending in 2030. This output position history is saved to a file for later use. The step size of the code is set at one day intervals.

9.2.1.4 propagator_3d.m

This code outputs the xyz position of a particle in orbit of a body.

9.2.1.4.1 Code

```
function [xyz] = propagator_3D(period,e,i,ts_i,RAAN,omega,mus,P_time);

% general 3D propagator
% [r,ts] = propagator_3D(period,e,i,ts_i,mus,P_time)
% period - period in days
% e - eccentricity of orbit
% i - inclination of orbit, deg
% ts_i - initial orbit location, deg
% RAAN - Right ascention of ascending node, deg
% omega - argument of periapsis, deg
% mus - Gravitational parameter of body which you are orbiting
% P_time - propagation time

t_day = 24*3600; %conversion from days to seconds

a = ((period*t_day/2/pi)^2*mus)^(1/3);
P = a*(1-e^2);

inc = deg2rad(i);
RAAN = deg2rad(RAAN);
omega = deg2rad(omega);

ts_i = deg2rad(ts_i);
E_i = 2*atan(sqrt((1-e)/(1+e))*tan(ts_i/2));
TOF_i = (E_i-e*sin(E_i))/sqrt(mus/a^3);

for i = 1:P_time
    try
        E(i) = fsolve('kepler',E(i-1),[],i*t_day+TOF_i,mus,a,e);
        ts = 2*atan(sqrt((1+e)/(1+e))*tan(E(i)/2));
        r = P/(1+e*cos(ts));
        transformation = xyz2rthh(inc,RAAN,omega+ts,'r')';
        xyz(i,:) = [r 0 0]*transformation;
    catch
        E(i) = fsolve('kepler',E_i,[],i*t_day+TOF_i,mus,a,e);
        ts = 2*atan(sqrt((1+e)/(1+e))*tan(E(i)/2));
        r = P/(1+e*cos(ts));
        transformation = xyz2rthh(inc,RAAN,omega+ts,'r')';
        xyz(i,:) = [r 0 0]*transformation;
    end
end
```

9.2.1.4.2 Code Output

This code outputs the xyz positions of a particle in orbit of a body (two-body problem). Inputs into this code are the period, inclination, eccentricity, initial true anomaly, right ascension of ascending node, argument of periapsis, the gravitational parameter of the body around which the spacecraft is orbiting and the length of time the propagation should occur for in days. The particle could be a satellite or a planet (modeled as a particle).

9.2.1.5 range_finder.m

This code is used to find the power required for HRS transmission to both the Earth and Mars simultaneously.

9.2.1.5.1 Code

```
% range finder

close all; clear all; clc; format long; format compact;
addpath(pwd);

cd C:\MATLAB7\work\aae450
cd SatData2_6_05
load PlanetaryData

EMR = comm_range(xyz_earth,xyz_mars);
power_EM = LBPower(EMR);

direc = dir;

for a = 3:length(direc);
    if direc(a).name(1:3) == 'HSD'
        fid = fopen(direc(a).name,'r');
        xyz = fread(fid,[365*16,3],'double');
        MSR = comm_range(xyz,xyz_mars);
        ESR = comm_range(xyz,xyz_earth);
        fclose(fid);
        power_MS(a-2,:) = LBPower(MSR);
        power_ES(a-2,:) = LBPower(ESR);
    end
end

cd ..

power_total = power_MS + power_ES;
figure;
contourf(power_total); colorbar;
title('Total Power Required for Satellite');

figure;
contourf(power_ES); colorbar;
title('Power Required for Satellite to Earth');

figure;
contourf(power_MS); colorbar;
title('Power Required for Satellite to Mars');

figure;
[time,phi0,phi1] = coverage(xyz_earth,xyz_mars);
plot(time,phi0,time,phi1);
hold on;
plot([0,5840],[3,3],'r',[0,5840],[177,177],'r');

[i0h j0h] = find(phi0<=3);
[i0l j0l] = find(phi0>=177);
[i1 j1] = find(phi1<=1);

max_dist = 0;
for a = 1:length(i0h)
    if EMR(i0h(a)) > max_dist
        max_dist = EMR(i0h(a));
    end
    try
        if EMR(i0l(a)) > max_dist
            max_dist = EMR(i0l(a));
        end
    end
    try
        if EMR(i1(a)) > max_dist
            max_dist = EMR(i1(a));
        end
    end
end
max_dist
```

9.2.1.5.2 Code Output

This code finds the maximum range between two points. It was specifically created to use the output of the Planetary_orbit_propagator.m and the Helio_sat_orbit_propagator.m to find the range distance between Earth, Mars, and a Heliocentric Satellite in a given orbit. From these range numbers, the power requirements were derived for the satellite to transmit to both Earth and Mars from a given location. Figure 9-5 shows a sample output from this code. The x-axis is the day of the orbit while the y-axis is a reference orbit number. The patterning of these orbits follows the naming convention given by their names. All the names of the satellites begin with 'HSD_' followed by the period of the orbit in days (integer values only), the eccentricity of the orbit (out to two decimal places, the number is 100x the eccentricity), and the inclination of the orbit.

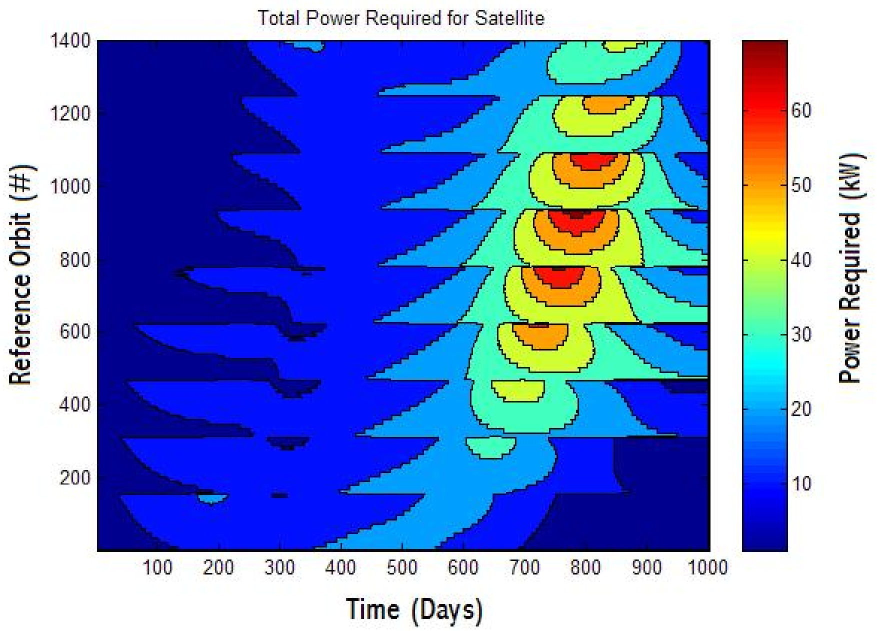


Figure 9-5: Power Required for Simultaneous Earth and Mars Transmission

From Figure 9-5 it is possible to find some various trends to narrow the search down to a few specific orbits. We can see that the increasing eccentricity and increasing inclination increase the power, as well as the increase of the period. All increases in along the y-axis. The final orbit of the heliocentric satellite was determined by the occultation of Mars by the Sun during opposition of the Earth and Mars.

9.2.1.6 Helio_sat_orbit_propagator.m

This code produces a number of file which contain the XYZ positions of a heliocentric satellite based on input Keplarian orbital elements.

9.2.1.6.1 Code

```
% Heliocentric Satellite Orbit Generator

mus = 1.3271244e11;
P_time = 365*16; %propagation time, days
ts_i = 0; %Initial true anomaly, deg
RAAN = 0; %initial RAAN, deg
omega = 0; %initial arg. of periapsis, deg
cd 'C:\MATLAB7\work\aae450\'
pathname = 'C:\MATLAB7\work\aae450\';
dname = 'SatData2_6_05';
mkdir([pathname dname]);

for P = 400:25:475
    for e = 5:2.5:50
        for i = 0:1
            xyz = propagator_3D(P,e/100,i,ts_i,RAAN,omega,mus,P_time);
            cd([pathname dname]);
            fname = sprintf('HSD_%i_%i_%i.dat',P,e,i);
            fid = fopen(fname,'w');
            fwrite(fid,xyz,'double');
            fclose(fid);
            cd ..
        end
    end
end
```

9.2.1.6.2 Code Output

This code outputs a number of files which contain the XYZ position information of a satellite in a heliocentric orbit. These files are used by range_finder.m to show the power requirements necessary for each heliocentric satellite orbit.

9.2.1.7 kepler.m

This code is used with the Matlab fsolve function to solve Kepler's Equation relating Time of Flight (TOF) to Eccentric Anomaly (E).

9.2.1.7.1 Code

```
function F = kepler(E,TOF,mue,at,et);

% kepler(E,TOF,mu,at,et)
% E   - what you are solving for
% TOF - Time Of Flight
% mu  - gravitational parameter
% at  - semi-major axis
% et  - eccentricity

F = E - et * sin(E) - sqrt(mue / at ^ 3) * TOF;
```

9.2.1.8 elems.m

Note: This code is modified from a code received from Eric Gustafson to find the orbital elements of Earth and Mars in the Mean Ecliptic of J2000 frame.

9.2.1.8.1 Code

```
function [a_earth,i_earth,e_earth,omega_earth,Omega_earth,a_mars,i_mars,e_mars,omega_mars,Omega_mars]
= elems;

%find oribital elements from chebytop:
clear; clc; close all
```

```
format compact
km_per_AU = 1.49597807e8; %km/AU
sec_per_year = 365*24*3600; %seconds/year

%DATA FROM CHEBYTOP - system/pldata.f
%EQUATION is from system/plelem.f
% Orbit Elements for Earth
% earth_elem = [a a1 a2; e e1 e2; ...]
% a (AU), e, i (deg), Omega (deg), omega (deg)
earth_elem = [1.00000023d0,.01670911d0,.0d0,174.876384d0,102.940753d0, 357.525443d0;
               .0d0,-.42052d-4,.01305635d0,-.24161358d0, .32343001d0,.9856002586d0;
               .0d0,-.126d-6,-.91465d-5,.9822d-5, .150003d-3,-.155d-3]';
% Orbit Elements for Mars
mars_elem = [1.52369150d0,.09340489d0,1.849709d0,49.559177d0,336.059380d0,19.388107d0;
              .0d0,.9191d-4,-.82027565d-2,-.29546843d0,.4441893d0,.5240207766d0;
              .0d0,-.77d-7,-.199914d-4,-.6487619d-3,-.18d-3,.1825972d-3]';

%define elements at launch date:
epoch = 2451545.0d0; %julian date of J2000
tau = 2457410.502; %departure date (from chebytop output)
time = tau - epoch;
t = time / 36525; %in centuries
for i=1:5
    earth_orb(i) = earth_elem(i,1) + t*(earth_elem(i,2) + t*earth_elem(i,3));
    mars_orb(i) = mars_elem(i,1) + t*(mars_elem(i,2) + t*mars_elem(i,3));
    %convert to km and radians
    if (i == 1)
        earth_orb(i) = earth_orb(i)*km_per_AU;
        mars_orb(i) = mars_orb(i)*km_per_AU;
    end
    if (i==3 | i==4 | i==5)
        earth_orb(i) = earth_orb(i)*pi/180;
        mars_orb(i) = mars_orb(i)*pi/180;
    end
end

%---------------------------------------------------
%---------------------------------------------------
%old plotting program now:
a_mars = mars_orb(1); %km
e_mars = mars_orb(2);
i_mars = mars_orb(3); %rad
Omega_mars = mars_orb(4); %rad
omega_mars = mars_orb(5); %rad
%
p_mars = a_mars*(1-e_mars^2); %km

%Earth orbital elements
a_earth = earth_orb(1); %km
e_earth = earth_orb(2);
i_earth = earth_orb(3); %rad
Omega_earth = earth_orb(4); %rad
omega_earth = earth_orb(5); %rad
%
p_earth = a_earth*(1-e_earth^2); %km
```

9.2.1.9 xyz2rthh.m

This code converts from the xyz inertial frame to the $\hat{r}$, $\hat{\theta}$, $\hat{h}$ orbit fixed frame. The transformation of this matrix is used to convert in the other direction.

9.2.1.9.1 Code

```
function transformation = xyz2rthh(inclination,omega,theta,units);

% Transformation Matrix for xyz coords to rthh coords
% inclination
% omega
% theta
% units - either 'r' for radians for 'd' for degrees

switch units
case 'r'
  i = inclination;
  o = omega;
  th = theta;
case 'd'
  i = inclination / 180 * pi;
  o = omega / 180 * pi;
  th = theta / 180 * pi;
otherwise
  fprintf('Please either specify ''r'' for radians or ''d'' for degrees for units\n');
  return
end

transformation = [cos(o)*cos(th)-sin(o)*cos(i)*sin(th),-cos(o)*sin(th)-
sin(o)*cos(i)*cos(th),sin(o)*sin(i);
                  sin(o)*cos(th)+cos(o)*cos(i)*sin(th),-sin(o)*sin(th)+cos(o)*cos(i)*cos(th),-
cos(o)*sin(i);
                  sin(i)*sin(th),sin(i)*cos(th),cos(i)];
return
```

9.2.1.10 LBPower.m

This code takes a given range measurement and finds the power required to transmit over that range in the X-band. This functionality was used by the range_finder.m file to find an initial guess for the power required for the HRS satellite.

9.2.1.10.1 Code

```
function [power] = LBPower(range);

% Link Budget Power required based on range for X-link

f_dl = 8.145; %X-band D/L, GHz
f_ul = 7.1623215; %X-band U/L, GHz
eta = .55; %antenna effeciency
D = 20; %m

Gt = 20.4+10*log10(eta)+20*log10(f_ul)+20*log10(D);
Gr = 20.4+10*log10(eta)+20*log10(f_dl)+20*log10(D);
T_ul = 10*log10(400);
T_dl = 10*log10(150);
L_other = 3;
SNR = 3;
Rd = 10*log10(100e6);
k = -228.6;

L_space_dl = 92.45+20*log10(f_dl)+20*log10(range);
L_space_ul = 92.45+20*log10(f_ul)+20*log10(range);

EIRP_dl = SNR+k+Rd+L_other-Gr+T_dl+L_space_dl;
power_dl = 10.^((EIRP_dl - Gt)/10)/1e3;

EIRP_ul = SNR+k+Rd+L_other-Gr+T_ul+L_space_ul;
power_ul = 10.^((EIRP_ul - Gt)/10)/1e3;

power = power_ul+power_dl;
```

9.3 Communications Dish Antenna Gain Pattern Analysis

Author: Brendan Eash

Contributor: Wayne Davis of Ball Aerospace & Technologies Corp.

During the course of the communications analysis, we found in necessary to look at the gain pattern of the antenna to help us with the pointing requirements of the CTV. Since it was necessary to have continuous communications coverage over the course of the mission and due to the fact that we could not perform a spacewalk to fix any broken parts, having a passive pointing system became very important. This problem was solved by the pointing the angular momentum vector of the CTV at Earth at all times. By keeping the communications dish at the center of rotation, we get a free look at Earth at all times. A stability analysis was performed with nutation perturbations being the most important piece of information to the communications group. It was determined that if the variability of the nutation angle was approximately the same magnitude as the communications antenna pattern half power beam-width (HPBW), communications could be assured.

Wayne Davis of Ball Aerospace & Technologies Corp. sent us a spreadsheet which would calculate the antenna gain pattern based on the diameter of the dish, the efficiency of the dish, and the bandwidth of the carrier signal. Figure 9-6 shows a sample output of this spreadsheet.

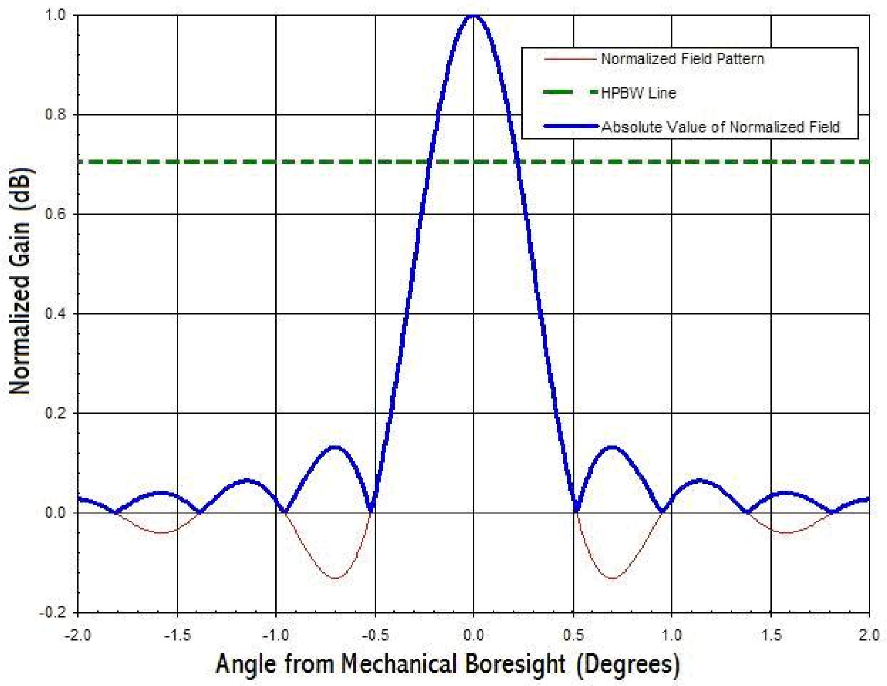


Figure 9-6: Antenna Gain Pattern for CTV Main Communications Dish (Wayne Davis)

9.4 Rigid Body Dynamics Inertia Matrix and Spin-Up Fuel Costs

Author: Brendan Eash

Contributors: Pete Browning, Zade Shaw, Ray Wright

As part of the communications analysis, we find it necessary to know where the main communications dish is pointed at all times. To do this analysis, it was necessary to define the inertia characteristics of the Crew Transport Vehicle (CTV). To find the inertia matrix of the CTV, we make a few simplifying assumptions:

1. Fuel sloshing is not a consideration.
2. The CTV is a rigid body. All parts of the CTV have a fixed volume and are uniformly dense within a given part (i.e. each 'section' of the CTV has constant volume and constant mass).
3. Since the largest masses changes will be due to fuel burns and the departure of the Mars Lander Vehicle (MLV), all inertia characteristics will remain constant between major burns and the departure of the MLV. This means we are neglecting the fuel lost to keep the CTV orientated in the correct direction. This fuel loss is roughly 1.3kg per day for the length of the mission. This also means that the inertia characteristics for the mission are discontinuous over the course of the mission.
4. For all major burns, the gravity torque during the burn was considered to be negligible compared to the length of time of the burn. Any appreciable gravity induced torque was assumed taken care of by vectoring of the main engines during the burn.

The inertia matrix of the CTV is used in the analysis of the spin up and spin down fuel cost of the CTV as well as in the angular momentum vector point analysis.

There are total of six different operating inertia configurations of the CTV. The operating inertia configurations are as follows:

1. Initial build before leaving Earth. Maximum mass and overall size. No spinning during this portion of the mission. See Figure 9-7 for the CTV's physical configuration.
2. Post Earth Departure burn. Two large external tanks drop off after the first major burn. See Figure 9-8 for the CTV's physical configuration.

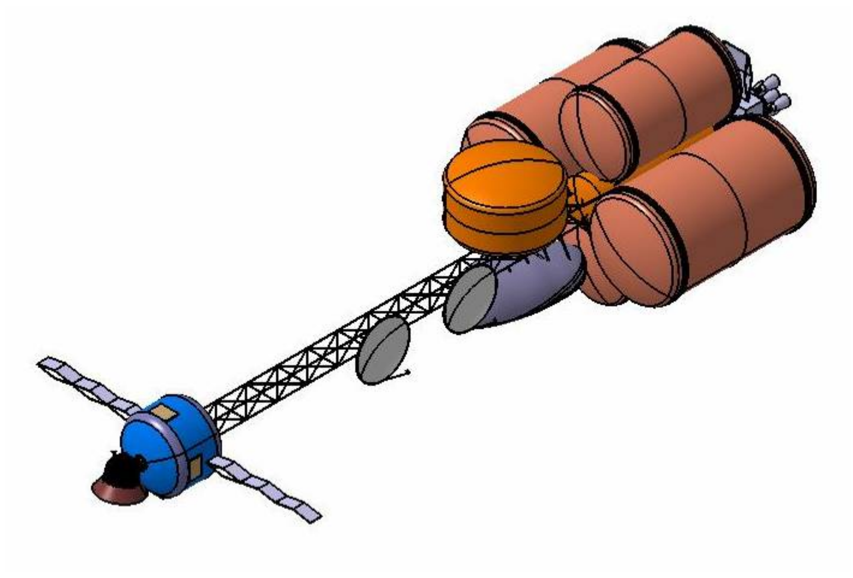


Figure 9-7: CTV Phase #1, Pre-Earth Departure

3. Post Mars Arrival, pre MLV separation. See Figure 9-8 for the CTV's physical configuration. This configuration is the same as the Post Earth Departure configure, only fuel is lost from the two large tanks between these configurations.

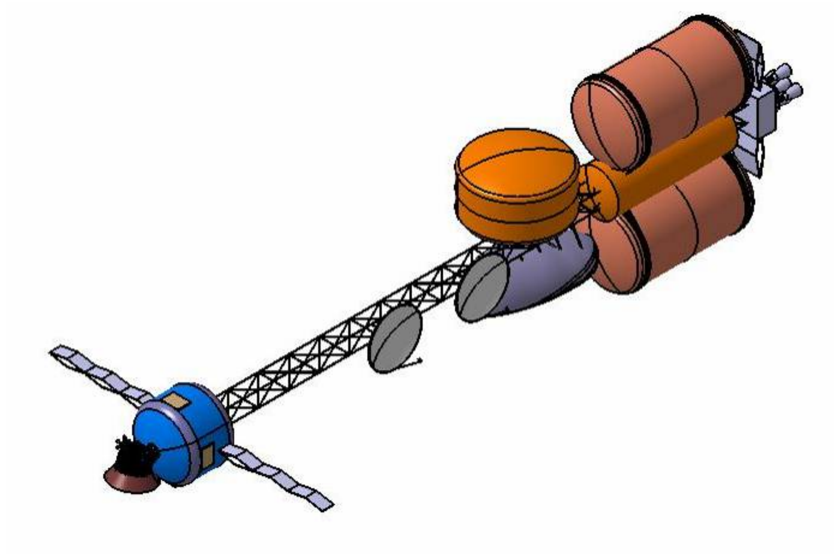


Figure 9-8: CTV Phase #2 – 3, Post Earth Departure, Post Mars Arrival

4. Post Mars Arrival, post MLV separation. Counter balance tank loads its fuel into another tank, is dropped with the MLV. See Figure 9-9 for the CTV's physical configuration.

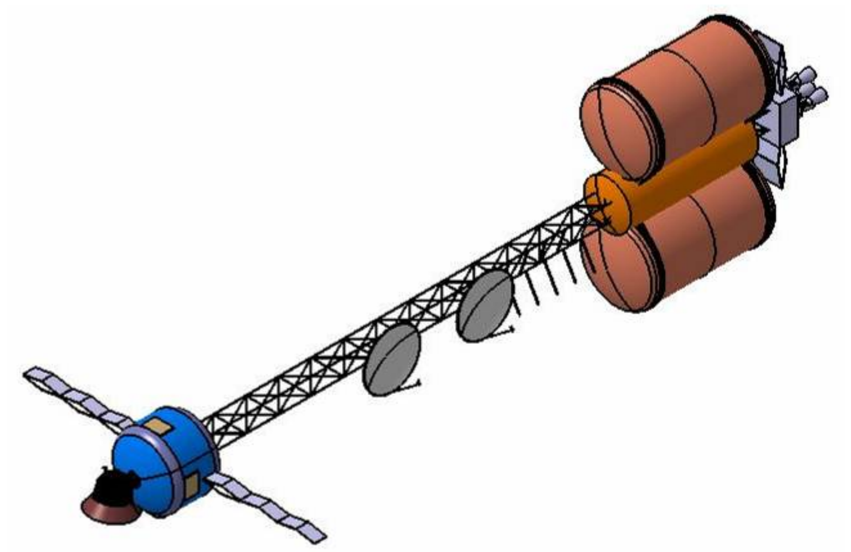


Figure 9-9: CTV Phase #4 - 5, Post ARV Seperation, Post Mars Departure

5. Post Mars Departure. MLV ascent stage is jettisoned before the departure burn. See Figure 9-9 for the CTV's physical configuration. This configuration is the same as the Post MLV departure and counter balance tank drop. The only change between the two is the loss of fuel for the Mars departure burn from the two large fuel tanks. Note that at this point, the large external fuel tanks should be empty and only carried to help balance out the CTV for spin stability and artificial gravity for the return to Earth.

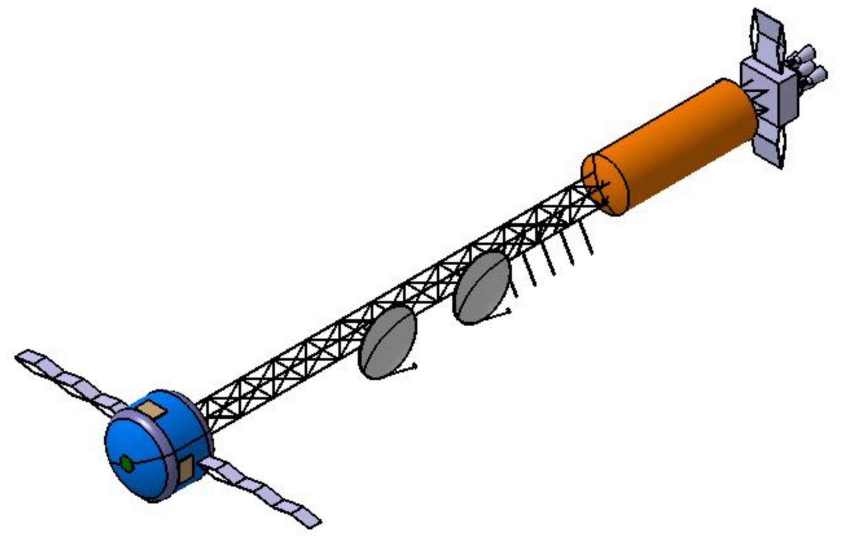


Figure 9-10: CTV Phase #6, Post Earth Arrival

6. Post Earth Arrival. The Ascent and Recovery Vehicle (AVR) has detached and has performed the hyperbolic re-entry maneuver. See Figure 9-10 for the CTV's physical configuration.

It is necessary for the CTV to stop spinning before each major maneuver (or course correction maneuver). To this end, we find that six total spin up/down cycles will cover all of the major maneuvers as they currently stand. Since the thrusters for the spin-up/down maneuvers are Xenon Ion Propulsion System (XIPS) thrusters. These thrusters are very efficient which means that the amount of fuel necessary for all the maneuvers is relatively small. The fuel required for all the spin-up/down maneuvers is approximately 650 kilograms.

9.4.1 Rigid Body Dynamics Inertia Matrix Codes

The following codes were used for the rigid body inertia matrix calculations of the CTV.

9.4.1.1 CTV_imatrix_dragster.m

This code produces the inertia matrix used for the CTV. This code has two inputs, a flag determining which portion of the mission the code should produce an inertia matrix for and also a flag which determines if the user wishes to see a 2D projection of the center of mass of each of the sections of the CTV. This portion of the code was added for visualization in debugging the code.

9.4.1.1.1 Code

```
function [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(flag,plot_flag);

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(flag,plot_flag)
%
% This function calculates the CM, dry and wet masses and the inertia
% matrix associated with the CTV in the dragster configuration.
%
% 'flag' is the only input corresponding to the mass properties of
% various segments of the mission (occurring after mission milestones).
%
% flag = 0 >> initial launch mass from either LPPO or LEO
% flag = 1 >> mass after Earth departure burn
% flag = 2 >> post Mars arrival, pre MLV seperation
% flag = 3 >> post MLV seperation
% flag = 4 >> post Mars departure
% flag = 5 >> Earth arrival, pre-capture burn, post ARV seperation
% flag = 6 >> Post Earth capture burn
%
% 'plot_flag' determines if the user wishes to output a 2D graphic showing
% the locations of each of the major mass' CM as well as the CM of the CTV.
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m_comm = 1078.7; %kg
m_dc = 206.65; %kg
m_hf = 39984.7; %kg
m_reactors = 2763; %kg
m_engines = 11000; %kg
m_nukeshld = 19000+8200; %kg
m_radshld = 16000; %kg, includes rad shld and structure for cc
m_truss_str = 17000; %kg
m_thermal = 9549.38; %kg
m_arv = 7500; %kg
m_mlv = 36000; %kg
m_fuel_burn1 = 200000; %kg
```

```
    m_fuel_burn2 = 13358; %kg
    m_fuel_burn3 = 10617; %kg
    m_fuel_burn4 = 41308; %kg
    m_fuel_burn5 = 15774; %kg
    m_fuel_burn6 = 36894; %kg

    m_fuel_tank0 = m_fuel_burn6; %kg, this tank is the small, final tank
    m_fuel_tank1 = m_fuel_burn1 / 2; %kg, first tank to be dropped
    m_fuel_tank2 = m_fuel_burn1 / 2; %kg, second tank to be dropped
    m_fuel_tank3 = (m_fuel_burn2 + m_fuel_burn3 + m_fuel_burn4 + ...
        m_fuel_burn5) / 2; %kg, third tank to be kept
    m_fuel_tank4 = (m_fuel_burn2 + m_fuel_burn3 + m_fuel_burn4 + ...
        m_fuel_burn5) / 2; %kg, fourth tank to be kept
    m_fuel_tank5 = m_mlv / 1.15; %kg, this tank is the counter mass to the MLV
    m_fuel_total = m_fuel_tank0 + m_fuel_tank1 + m_fuel_tank2 + ...
        m_fuel_tank3 + m_fuel_tank4 + m_fuel_tank5; %kg
    m_tank_str0 = .14 * m_fuel_tank0; %kg
    m_tank_str1 = .11 * m_fuel_tank1; %kg
    m_tank_str2 = .11 * m_fuel_tank2; %kg
    m_tank_str3 = .14 * m_fuel_tank3; %kg
    m_tank_str4 = .14 * m_fuel_tank4; %kg
    m_tank_str5 = .14 * m_fuel_tank5; %kg
    d_tank0 = 5; %m
    d_tank1 = 10; %m
    d_tank2 = 10; %m
    d_tank3 = 8; %m
    d_tank4 = 8; %m
    d_tank5 = 10; %m
    d_nuke = 5; %m, should be the same size as d_tank0

    m_dry = m_comm + m_dc + m_hf + m_reactors + m_engines + m_radshld + ...
        m_truss_str + m_thermal + m_tank_str0 + m_tank_str1 + m_tank_str2 + ...
        m_tank_str3 + m_tank_str4 + m_tank_str5 + m_radshld + m_nukeshld;

    switch flag
        case 0 % initial mass in LEO
        case 1 % Post-Earth Departure
            m_tank_str1 = 0;
            m_fuel_tank1 = 0;
            m_tank_str2 = 0;
            m_fuel_tank2 = 0;
        case 2 % Post Mars insertion
            m_tank_str1 = 0;
            m_fuel_tank1 = 0;
            m_tank_str2 = 0;
            m_fuel_tank2 = 0;
            m_fuel_tank3 = (m_fuel_burn3 + m_fuel_burn4 + m_fuel_burn5) / 2;
            m_fuel_tank4 = (m_fuel_burn3 + m_fuel_burn4 + m_fuel_burn5) / 2;
        case 3 % Apotwist #1 (includes lander)
            m_tank_str1 = 0;
            m_fuel_tank1 = 0;
            m_tank_str2 = 0;
            m_fuel_tank2 = 0;
            m_fuel_tank3 = (m_fuel_burn4 + m_fuel_burn5) / 2;
            m_fuel_tank4 = (m_fuel_burn4 + m_fuel_burn5) / 2;
        case 4 % Apotwist#2 (no lander)
            m_tank_str1 = 0;
            m_fuel_tank1 = 0;
            m_tank_str2 = 0;
            m_fuel_tank2 = 0;
            m_fuel_tank3 = m_fuel_burn5 / 2;
            m_fuel_tank4 = m_fuel_burn5 / 2;
            m_mlv = 0;
        case 5 % Post Mars Departure
            m_tank_str1 = 0;
            m_fuel_tank1 = 0;
            m_tank_str2 = 0;
            m_fuel_tank2 = 0;
            m_fuel_tank3 = 0;
            m_fuel_tank4 = 0;
            m_mlv = 0;
            m_tank_str5 = 0;
            m_fuel_tank5 = 0;
        case 6 % Earth Arrival (post-capture burn)
            m_tank_str1 = 0;
            m_fuel_tank1 = 0;
            m_tank_str2 = 0;
            m_fuel_tank2 = 0;
            m_fuel_tank3 = 0;
            m_fuel_tank4 = 0;
            m_mlv = 0;
            m_tank_str5 = 0;
            m_fuel_tank5 = 0;
            m_avr = 0;
            m_fuel_tank0 = 0;
            m_tank_str3 = 0;
            m_tank_str4 = 0;
        otherwise
            m_tank_str1 = 0;
            m_fuel_tank1 = 0;
            m_tank_str2 = 0;
            m_fuel_tank2 = 0;
    end

    % inertia information for the crew compartment, modeled as a solid cylinder
    m_cc = m_hf + m_dc + m_radshld;
    r_cc = 7.3 / 2; %m
    L_cc = 5; %m
    I = cylinder_inertia(L_cc,r_cc,m_cc);
    C = [0,0,1;1,0,0;0,1,0];
    I_cc_initial = C' * I * C;

    % truss inertia information, modeled as a solid cylinder
    m_truss = m_truss_str + m_comm;
    r_truss = .5; %m
    L_truss = 60; %m
    I = cylinder_inertia(L_truss,r_truss,m_truss);
    C = [0,0,1;1,0,0;0,1,0];
    I_truss_initial = C' * I * C;

    % small tank between power and crew, modeled as a solid cylinder
    r_tank0 = d_tank0 / 2; %m
    L_tank0 = 12.14; %m
    m_tank0 = m_fuel_tank0 + m_tank_str0;
    I = cylinder_inertia(L_tank0,r_tank0,m_tank0);
    C = [0,0,1;1,0,0;0,1,0];
    I_tank0_initial = C' * I * C;

    % large tank, dropped after first burn, modeled as a solid cylinder
    r_tank1 = d_tank1 / 2; %m
    L_tank1 = 18.77; %m
    m_tank1 = m_fuel_tank1 + m_tank_str1;
    I = cylinder_inertia(L_tank1,r_tank1,m_tank1);
    C = [0,0,1;1,0,0;0,1,0];
    I_tank1_initial = C' * I * C;

    % large tank, dropped after first burn, modeled as a solid cylinder
    r_tank2 = d_tank2 / 2; %m
    L_tank2 = 18.77; %m
    m_tank2 = m_fuel_tank2 + m_tank_str2;
    I = cylinder_inertia(L_tank2,r_tank2,m_tank2);
    C = [0,0,1;1,0,0;0,1,0];
    I_tank2_initial = C' * I * C;

    % large tank, kept for entire mission duration, modeled as a solid cylinder
    r_tank3 = d_tank3 / 2; %m
    L_tank3 = 15.79; %m
    m_tank3 = m_fuel_tank3 + m_tank_str3;
    I = cylinder_inertia(L_tank3,r_tank3,m_tank3);
    C = [0,0,1;1,0,0;0,1,0];
    I_tank3_initial = C' * I * C;
    % large tank, kept for entire mission duration, modeled as a solid cylinder
    r_tank4 = d_tank4 / 2; %m
    L_tank4 = 15.79; %m
    m_tank4 = m_fuel_tank4 + m_tank_str4;
    I = cylinder_inertia(L_tank4,r_tank4,m_tank4);
    C = [0,0,1;1,0,0;0,1,0];
    I_tank4_initial = C' * I * C;

    % tank to balance out MLV, dropped when MLV goes, modeled as a solid
    % cylinder
    r_tank5 = d_tank5 / 2; %m
    L_tank5 = 5.22; %m
    m_tank5 = m_fuel_tank5 + m_tank_str5;
    I = cylinder_inertia(L_tank5,r_tank5,m_tank5);
    C = [0,0,1;1,0,0;0,1,0];
    I_tank5_initial = C' * I * C;

    % power reactors and thermal control systems, modeled as a solid cylinder
    r_nuke = d_nuke / 2; %m
    L_nuke = 5; %m
    m_nuke = m_reactors + m_engines + m_thermal + m_nukeshld;
    I = cylinder_inertia(L_nuke,r_nuke,m_nuke);
    C = [0,0,1;1,0,0;0,1,0];
    I_nuke_initial = C' * I * C;

    % ARV inertia characteristics, modeled as a cone
    r_arv = 4 / 2; %m
    h_arv = 4; %m
    I = cone_inertia(r_arv,h_arv,m_arv);
    C = [0,1,0;0,0,1;1,0,0];
    I_arv_initial = C' * I * C;

    % MLV inertia characteristics, modeled as a solid cylinder
    L_mlv = 16; %m
    r_mlv = 8 / 2; %m
    I = cylinder_inertia(L_mlv,r_mlv,m_mlv);
    C = eye(3);
    I_mlv_initial = C' * I * C;

    % locations of each large mass section
    rho_cc = [(L_cc / 2 + L_truss + L_tank0 / 2), 0, 0]; %crew compartment center
    rho_truss = [(L_tank0 + L_truss / 2), 0, 0]; %truss center
    rho_tank0 = [(L_tank0 / 2), 0, 0]; %central tank in rear of CTV
    rho_tank1 = [0, (r_tank0 + r_tank1), 0]; %dropped after Earth departure
    rho_tank2 = [0,-(r_tank0 + r_tank2), 0]; %dropped after Earth departure
    rho_tank3 = [0, 0, (r_tank0 + r_tank3)]; %tank is kept until last burn
    rho_tank4 = [0, 0,-(r_tank0 + r_tank4)]; %tank is kept until last burn
    rho_tank5 = [(L_tank0 + L_truss / 4), 0, (r_truss + r_tank5)];%balances MLV
    rho_nuke = [-(L_nuke / 2),0,0]; %nuclear power reactor and engines
    rho_arv = [(L_tank0 + L_truss + L_cc + h_arv / 4), 0, 0]; %ARV
    rho_mlv = [(L_tank0 + L_truss / 4), 0, -(r_truss + r_mlv)]; %MLV

    m_dry1 = m_comm + m_dc + m_hf + m_reactors + m_engines + m_radshld + ...
        m_truss_str + m_thermal + m_tank_str0 + m_tank_str1 + m_tank_str2 + ...
        m_tank_str3 + m_tank_str4 + m_tank_str5 + m_radshld;

    m_fuel_total = m_fuel_tank0 + m_fuel_tank1 + m_fuel_tank2 + ...
        m_fuel_tank3 + m_fuel_tank4 + m_fuel_tank5;

    m_wet = m_dry1 + m_fuel_total + m_arv + m_mlv;

    CTV_CM = (rho_cc * m_cc + rho_truss * m_truss + rho_tank0 * m_tank0 + ...
        rho_tank1 * m_tank1 + rho_tank2 * m_tank2 + rho_tank3 * m_tank3 + ...
        rho_tank4 * m_tank4 + rho_tank5 * m_tank5 + rho_nuke * m_nuke + ...
        rho_arv * m_arv + rho_mlv * m_mlv) / m_wet;

    % plot out locations of each mass' CM.  Plot the overall CM as well.
    if plot_flag == 1
        xoffset = .5;
        yoffset = 1;
        figure; hold on;
        plot(rho_cc(1),rho_cc(2),'r*');
        text(rho_cc(1)+xoffset,rho_cc(2)+yoffset,'Crew Compartment CM');
        plot(rho_truss(1),rho_truss(2),'r*');
        text(rho_truss(1)+xoffset,rho_truss(2)-yoffset,'Truss CM');
        plot(rho_tank0(1),rho_tank0(2),'r*');
        text(rho_tank0(1)+xoffset,rho_tank0(2)+yoffset,'Tank_0');
        plot(rho_tank1(1),rho_tank1(2),'r*');
        text(rho_tank1(1)+xoffset,rho_tank1(2)+yoffset,'Tank_1');
        plot(rho_tank2(1),rho_tank2(2),'r*');
        text(rho_tank2(1)+xoffset,rho_tank2(2)-yoffset,'Tank_2');
        plot(rho_tank3(1),rho_tank3(2),'r*');
        text(rho_tank3(1)+xoffset,rho_tank3(2)+yoffset,'Tank_3');
        plot(rho_tank4(1),rho_tank4(2),'r*');
        text(rho_tank4(1)+xoffset,rho_tank4(2)+yoffset,'Tank_4');
        plot(rho_tank5(1),rho_tank5(2),'r*');
        text(rho_tank5(1)+xoffset,rho_tank5(2)+yoffset,'Tank_5');
        plot(rho_nuke(1),rho_nuke(2),'r*');
        text(rho_nuke(1)+xoffset,rho_nuke(2)+yoffset,'Nuke');
        plot(rho_arv(1),rho_arv(2),'r*');
        text(rho_arv(1)+xoffset,rho_arv(2)-yoffset,'ARV');
        plot(rho_mlv(1),rho_mlv(2),'r*');
        text(rho_mlv(1)+xoffset,rho_mlv(2)+yoffset,'MLV');
        plot(CTV_CM(1),CTV_CM(2),'bo');
        text(CTV_CM(1)+xoffset,CTV_CM(2)+yoffset,'CTV CM');
    end

    % move the inertia of each section to the CM location of the CTV
    I_cc = parallel_axis_theorem(I_cc_initial,m_cc,rho_cc-CTV_CM);
    I_truss = parallel_axis_theorem(I_truss_initial,m_truss,rho_truss-CTV_CM);
    I_tank0 = parallel_axis_theorem(I_tank0_initial,m_tank0,rho_tank0-CTV_CM);
    I_tank1 = parallel_axis_theorem(I_tank1_initial,m_tank1,rho_tank1-CTV_CM);
    I_tank2 = parallel_axis_theorem(I_tank2_initial,m_tank2,rho_tank2-CTV_CM);
    I_tank3 = parallel_axis_theorem(I_tank3_initial,m_tank3,rho_tank3-CTV_CM);
    I_tank4 = parallel_axis_theorem(I_tank4_initial,m_tank4,rho_tank4-CTV_CM);
    I_tank5 = parallel_axis_theorem(I_tank5_initial,m_tank5,rho_tank5-CTV_CM);
    I_arv = parallel_axis_theorem(I_arv_initial,m_arv,rho_arv-CTV_CM);
    I_mlv = parallel_axis_theorem(I_mlv_initial,m_mlv,rho_mlv-CTV_CM);
    I_nuke = parallel_axis_theorem(I_nuke_initial,m_nuke,rho_nuke-CTV_CM);

    % Calculate the total inertia of the CTV
    I_CTV = I_cc + I_truss + I_tank0 + I_tank1 + I_tank2 + I_tank3 + ...
        I_tank4 + I_tank5 + I_arv + I_mlv + I_nuke;
```

9.4.1.1.2 Code Output

This code outputs not only the CTV’s inertia matrix, but also its dry mass, wet mass and the location of the center of mass from the center of the primary nuclear propulsion system.

9.4.1.2 cone_inertia.m

This code generates the inertia matrix of a right cone of uniform density.

9.4.1.2.1 Code

```
function [I] = cone_inertia(r,h,m);

% [I] = cone_inertia(r,h,m);
% This function determines the inertia matrix associated with a right
% cone where the z-axis is along the principal axis of the cone.
% r - radius of the cylinder
% h - length of the cylinder
% m - mass of the cylinder

I(1,1) = 3 / 20 * m * r ^ 2 + 3 / 80 * m * h ^ 2;
I(2,2) = 3 / 20 * m * r ^ 2 + 3 / 80 * m * h ^ 2;
I(3,3) = 3 / 10 * m * r ^ 2;
```

9.4.1.2.2 Code Output

The inputs into this code are the radius of the right circular cone, the height of the cone and the mass of the cone.

9.4.1.3 cylinder_inertia.m

This code generates the inertia matrix of a right circular cylinder of uniform density.

9.4.1.3.1 Code

```
function [I] = cylinder_inertia(L,r,m);

% [I] = cylinder_inertia(L,r,m);
% This function determines the inertia matrix associated with a right
% cylinder where the y-axis is along the principal axis of the cylinder.
% L - length of the cylinder
% r - radius of the cylinder
% m - mass of the cylinder

I(1,1) = m / 12 * (3 * r ^ 2 + L ^ 2);
I(2,2) = m * r ^ 2 / 2;
I(3,3) = m / 12 * (3 * r ^ 2 + L ^ 2);
```

9.4.1.3.2 Code Output

The inputs into this code are the radius of the right circular cylinder, the length of the cylinder and the mass of the cylinder.

9.4.1.4 parallel_axis_theorem.m

This code uses the parallel axis theorem to find the inertia matrix at a point which is not the center of mass of an object.

9.4.1.4.1 Code

```
function inertia_matrix = parallel_axis_theorem(I,M,rho);

% inertia_matrix = parallel_axis_theorem(I,M,rho);
% This function uses the parallel axis theorem to shift an inertia matrix
% such that the inertia of several objects can be summed up.
% I - initial inertia matrix
% M - total mass of the object
% rho - 3 element vector which is from the CM of the object of interest to
% the point q.  The 3 element vector MUST be in the same vector basis as
% the inertia matrix.

for i = 1:3
    for j = 1:3
        if i == j
            switch i
                case 1
                    I_final(i,j) = I(i,j) + M * (rho(2) ^ 2 + rho(3) ^ 2);
                case 2
                    I_final(i,j) = I(i,j) + M * (rho(1) ^ 2 + rho(3) ^ 2);
                case 3
                    I_final(i,j) = I(i,j) + M * (rho(2) ^ 2 + rho(1) ^ 2);
            end
        else
            I_final(i,j) = I(i,j) - M * rho(i) * rho(j);
        end
    end
end

inertia_matrix = I_final;
```

9.4.1.4.2 Code Output

This code outputs the inertia matrix of a known object at a new location. The inputs into this code include the inertia matrix at the mass center and a vector which is the difference between the position of the mass center and the new position of interest.

9.4.2 Spin Fuel Cost and Rates Codes

These codes find the cost of spin up and spin down for artificial gravity as well as the cost to 'slew' the CTV into position to perform a burn.

9.4.2.1 spin_rate_cost.m

This code determines the amount of fuel that is required for the spin up and spin down cycles over the duration of the mission.

9.4.2.1.1 Code

```
% spin_rate_cost.m
% created by:  Brendan Eash, March 7th, 2005

clear all; close all; clc; format long; format compact;

tic;

g0 = 9.81;
g1 = .38 * g0;
re = [52.5 0 0]; %m
Isp_ranges = [300 450 3800];
max_num_engines = 10;
row = 0;
time_step = 1; %(hrs)
mprop_reqd = zeros(72/time_step,3);
F_per_engine = zeros(72/time_step,max_num_engines,1);
t_start = 1;

Isp = Isp_ranges(1);
for dt = t_start:time_step:72
    row = row + 1;
    for n = 1:max_num_engines
        for i = 1:4
            [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(i,0);
            [L I] = eig(I_CTV);
            r = re - CTV_CM;
            omega0 = sqrt(g0 / r(1));
            omega1 = sqrt(g1 / r(1));
            if i == 1 | i == 4
                omega = omega0;
                num_burn = 4;
            else %spin so astronauts have .38g at Mars
                omega = omega1;
                num_burn = 2;
            end
            M = omega * I(3,3) / dt / 3600;
            F = M / r(1) / n;
            if i == 1
                if F_per_engine((dt/time_step),n,i) < F;
                    F_per_engine((dt/time_step),n,i) = F;
                end
            end
```

```
                mdot = F / Isp / g0;
                mprop_phase = num_burn * mdot * dt * 3600;
                if n == 1;
                    mprop_reqd((dt/time_step),1) = mprop_reqd((dt/time_step),1) + mprop_phase * n;
                end
            end
        end
    end

    F_per_engine(1,:,1)

    figure;
    contourf(F_per_engine(:,:,1)); colorbar;
    xlabel('Number of Engines');
    ylabel('Time');
    title('Thrust per engine vs. # of engines');
    axis([1 10 t_start 72]);

    Isp = Isp_ranges(2);
    for dt = time_step:time_step:72
        row = row + 1;
        for n = 1:max_num_engines
            for i = 1:4
                [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(i,0);
                [L I] = eig(I_CTV);
                r = re - CTV_CM;
                omega0 = sqrt(g0 / r(1));
                omega1 = sqrt(g1 / r(1));
                if i == 1 | i == 4
                    omega = omega0;
                    num_burn = 4;
                else
                    omega = omega1;
                    num_burn = 2;
                end
                M = omega * I(3,3) / dt / 3600;
                F = M / r(1) / n;
                mdot = F / Isp / g0;
                mprop_phase = num_burn * mdot * dt * 3600;
                if n == 1;
                    mprop_reqd((dt/time_step),2) = mprop_reqd((dt/time_step),2) + mprop_phase * n;
                end
            end
        end
    end

    Isp = Isp_ranges(3);
    for dt = time_step:time_step:72
        row = row + 1;
        for n = 1:max_num_engines
            for i = 1:4
                [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(i,0);
                [L I] = eig(I_CTV);
                r = re - CTV_CM;
                omega0 = sqrt(g0 / r(1));
                omega1 = sqrt(g1 / r(1));
                if i == 1 | i == 4
                    omega = omega0;
                    num_burn = 4;
                else
                    omega = omega1;
                    num_burn = 2;
                end
                M = omega * I(3,3) / dt / 3600;
                F = M / r(1) / n;
                mdot = F / Isp / g0;
                mprop_phase = num_burn * mdot * dt * 3600;
                if n == 1;
                    mprop_reqd((dt/time_step),3) = mprop_reqd((dt/time_step),3) + mprop_phase * n;
                end
            end
        end
    end

    mean(mprop_reqd)

    % XIPS 10 engines
    num_engines = 10;
    F_per_engineXIPS = .167;
    [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(1,0);
    [L I] = eig(I_CTV);
    r = re - CTV_CM;
    omega0 = sqrt(g0 / r(1));
    omega1 = sqrt(.38 * g0 / r(1));
    omega2 = sqrt(.5 * g0 / r(1));
    TIME_sec0 = omega0 * I(3,3) / r(1) / num_engines / F_per_engineXIPS;
    TIME_sec1 = omega1 * I(3,3) / r(1) / num_engines / F_per_engineXIPS;
    TIME_sec2 = omega2 * I(3,3) / r(1) / num_engines / F_per_engineXIPS;
    TIME0 = TIME_sec0 / 3600
    TIME1 = TIME_sec1 / 3600
    TIME2 = TIME_sec2 / 3600
    toc;

    TIME=86400*3;
    omega = TIME / I(3,3) * r(1) * num_engines * F_per_engineXIPS;
    a = r(1)*omega^2;
    num_g = a/g0
```

9.4.2.1.2 Code Output

This code outputs the spin up and spin down cost associated with the 1.00g artificial gravity required for the mission. It is important to note that the spin up (and spin down) time for the full 1.00g is approximately two weeks.

9.4.2.2 spin_rates.m

This code outputs the angular velocity necessary to achieve either one 'g' or 0.38 'g' for each leg of the CTV's mission. It also displays the inertia dyadic as well as the moment arm during each stage of the mission.

9.4.2.2.1 Code

```
% spin_rates.m
% created by:  Brendan Eash, March 7th, 2005

clear; clc; close all;

g0 = 9.81; %m/sec^2
g1 = .38 * g0;
re = [62.5 0 0]; %m

maxspin = 6 * 2 * pi / 60;

help CTV_imatrix_dragster

for i = 0:6;
    [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(i,0);
    [L I] = eig(I_CTV);
    r = re - CTV_CM;
    omega0 = sqrt(g0 / r(1));
    omega1 = sqrt(g1 / r(1));
    fprintf('==============================\n');
    fprintf('=   CTV at Mission Phase #%i   =\n',i);
    fprintf('==============================\n');
    fprintf('For 1.00g of gravity:\n');
    fprintf('---------------------\n');
    fprintf('omega = %f rad/sec\n',omega0);
    fprintf('Maximum omega:  %frad/sec\n',maxspin);
    fprintf('moment arm = (%f b1 + %f b2 + %f b3)m\n',r(1),r(2),r(3));
    fprintf('\n');
    fprintf('For 0.38g of gravity:\n');
    fprintf('---------------------\n');
    fprintf('omega = %f rad/sec\n',omega1);
    fprintf('Maximum omega:  %frad/sec\n',maxspin);
    fprintf('Moment Arm = (%f b1 + %f b2 + %f b3)m\n',r(1),r(2),r(3));
    fprintf('\n');
    fprintf('Inertia dyadic:\n')
    fprintf('---------------\n');
    fprintf('[%f b1b1 + %f b2b2 + %f b3b3]kg*m^2\n',I(1,1),I(2,2),I(3,3));
    fprintf('\n');
end
```

9.4.2.2.2 Code Output

```
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  [I_CTV,CTV_CM,m_dry,m_wet] = CTV_imatrix_dragster(flag,plot_flag)

  This function calculates the CM, dry and wet masses and the inertia
  matrix associated with the CTV in the dragster configuration.

  'flag' is the only input corresponding to the mass properties of
  various segments of the mission (occurring after mission milestones).

  flag = 0 >> initial launch mass from either LPPO or LEO
  flag = 1 >> mass after Earth departure burn
  flag = 2 >> post Mars arrival, pre MLV seperation
  flag = 3 >> post MLV seperation
  flag = 4 >> post Mars departure mass
  flag = 5 >> Earth arrival, pre-capture burn, post ARV seperation
  flag = 6 >> Post Earth capture burn

  'plot_flag' determines if the user wishes to output a 2D graphic showing
  the locations of each of the major mass' CM as well as the CM of the CTV.

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

===============================
=   CTV at Mission Phase #0   =
===============================
For 1.00g of gravity:
---------------------
omega = 0.446290 rad/sec
Maximum omega:  0.628319rad/sec
moment arm = (49.253304 b1 + 0.000000 b2 + -0.062411 b3)m

For 0.38g of gravity:
---------------------
omega = 0.275111 rad/sec
Maximum omega:  0.628319rad/sec
Moment Arm = (49.253304 b1 + 0.000000 b2 + -0.062411 b3)m

Inertia dyadic:
---------------
[23744459.679102 b1b1 + 325359678.529405 b2b2 + 332761396.177012 b3b3]kg*m^2

===============================
=   CTV at Mission Phase #1   =
===============================
For 1.00g of gravity:
---------------------
omega = 0.493585 rad/sec
Maximum omega:  0.628319rad/sec
moment arm = (40.266552 b1 + 0.000000 b2 + -0.104752 b3)m

For 0.38g of gravity:
```

```
---------------------
omega = 0.304266 rad/sec
Maximum omega:  0.628319rad/sec
Moment Arm = (40.266552 b1 + 0.000000 b2 + -0.104752 b3)m

Inertia dyadic:
---------------
[8481194.447409 b1b1 + 250555757.716925 b2b2 + 255640774.837625 b3b3]kg*m^2



===============================
=   CTV at Mission Phase #2   =
===============================
For 1.00g of gravity:
---------------------
omega = 0.499489 rad/sec
Maximum omega:  0.628319rad/sec
moment arm = (39.320335 b1 + 0.000000 b2 + -0.109210 b3)m

For 0.38g of gravity:
---------------------
omega = 0.307906 rad/sec
Maximum omega:  0.628319rad/sec
Moment Arm = (39.320335 b1 + 0.000000 b2 + -0.109210 b3)m

Inertia dyadic:
---------------
[7809854.619521 b1b1 + 243821785.666187 b2b2 + 248342326.958999 b3b3]kg*m^2

===============================
=   CTV at Mission Phase #3   =
===============================
For 1.00g of gravity:
---------------------
omega = 0.504725 rad/sec
Maximum omega:  0.628319rad/sec
moment arm = (38.508822 b1 + 0.000000 b2 + -0.113033 b3)m

For 0.38g of gravity:
---------------------
omega = 0.311133 rad/sec
Maximum omega:  0.628319rad/sec
Moment Arm = (38.508822 b1 + 0.000000 b2 + -0.113033 b3)m

Inertia dyadic:
---------------
[7276260.054434 b1b1 + 238083237.602126 b2b2 + 242155120.329852 b3b3]kg*m^2

===============================
=   CTV at Mission Phase #4   =
===============================
For 1.00g of gravity:
---------------------
omega = 0.532283 rad/sec
Maximum omega:  0.628319rad/sec
moment arm = (34.624475 b1 + 0.000000 b2 + -0.868678 b3)m

For 0.38g of gravity:
---------------------
omega = 0.328122 rad/sec
Maximum omega:  0.628319rad/sec
Moment Arm = (34.624475 b1 + 0.000000 b2 + -0.868678 b3)m

Inertia dyadic:
---------------
[3401218.790790 b1b1 + 210862150.779974 b2b2 + 212925456.244056 b3b3]kg*m^2

===============================
=   CTV at Mission Phase #5   =
===============================
For 1.00g of gravity:
---------------------
omega = 0.554079 rad/sec
Maximum omega:  0.628319rad/sec
moment arm = (31.954074 b1 + 0.000000 b2 + 0.000000 b3)m

For 0.38g of gravity:
---------------------
omega = 0.341557 rad/sec
Maximum omega:  0.628319rad/sec
Moment Arm = (31.954074 b1 + 0.000000 b2 + 0.000000 b3)m
Inertia dyadic:
---------------
[1245086.525188 b1b1 + 198393755.631610 b2b2 + 198873207.786610 b3b3]kg*m^2

===============================
=   CTV at Mission Phase #6   =
===============================
For 1.00g of gravity:
---------------------
omega = 0.666921 rad/sec
Maximum omega:  0.628319rad/sec
moment arm = (22.055673 b1 + 0.000000 b2 + 0.000000 b3)m


For 0.38g of gravity:
---------------------
omega = 0.411118 rad/sec
Maximum omega:  0.628319rad/sec
Moment Arm = (22.055673 b1 + 0.000000 b2 + 0.000000 b3)m

Inertia dyadic:
---------------
[559556.780187 b1b1 + 160412021.826400 b2b2 + 160412021.826400 b3b3]kg*m^2
```

10 Evans, Meredith G.

10.1 MHV Capture Maneuver Appendix

Author: Meredith Evans

Contributor(s): Eric Gustafson, Kim Mrozek

10.1.1 MHV Aero-Braking Analysis

After the transfer is complete the MHV burns into a loose Martian orbit with a period of 7 days and a periapsis of 500 kilometers. The MHV undergoes a maneuver at apoapsis to initiate the aero-braking process. Since Mars' atmosphere is very shallow we assumed that the atmosphere starts at 100 km. From that we decided to let the radius of periapsis be 80 km. A ballute is also being used to help slow the vehicle down by subjecting the vehicle to larger drag coefficient. The script *aerobrakingtrial.m* is used with the function file *MHV.m* to analyze the trajectory. A function file *rhomars.m* for Mars' atmosphere was also used. The results for each pass are found in Table 3 below. Figure 11 is theta start with respect to time and is used to show that the trajectory makes sense. Figure 12 shows the position with respect to time again to show that the trajectory makes sense. The aero-braking trajectory can be seen in Figure 13. The velocity profile can be seen in Figure 14.

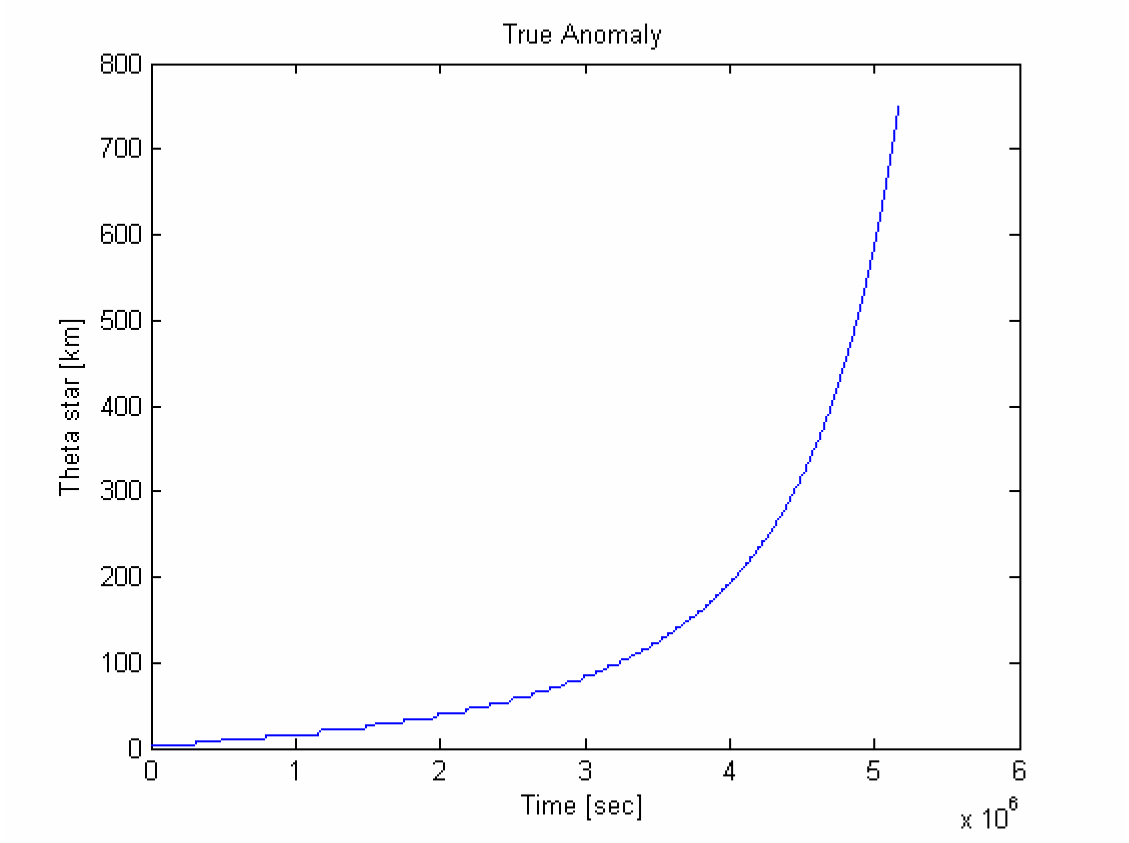


Figure 11 True Anomaly verses Time

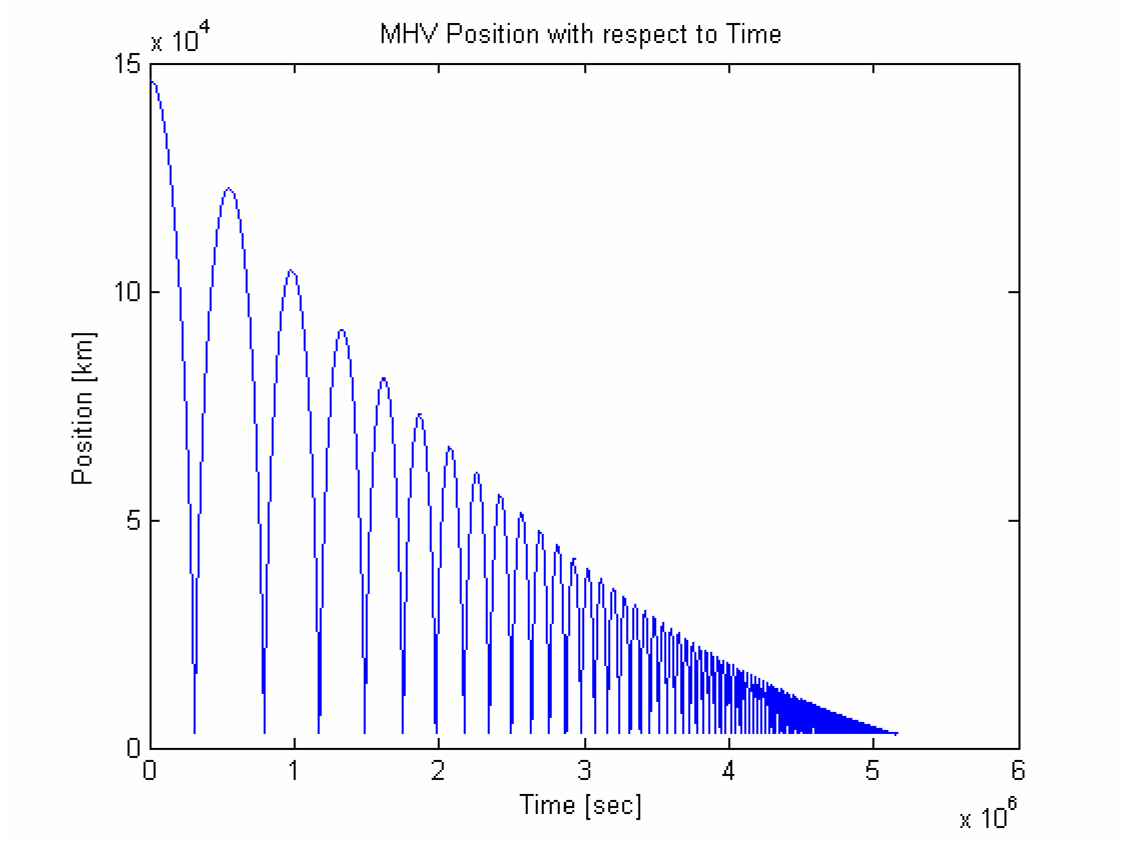


Figure 12 MHV Position with respect to Time

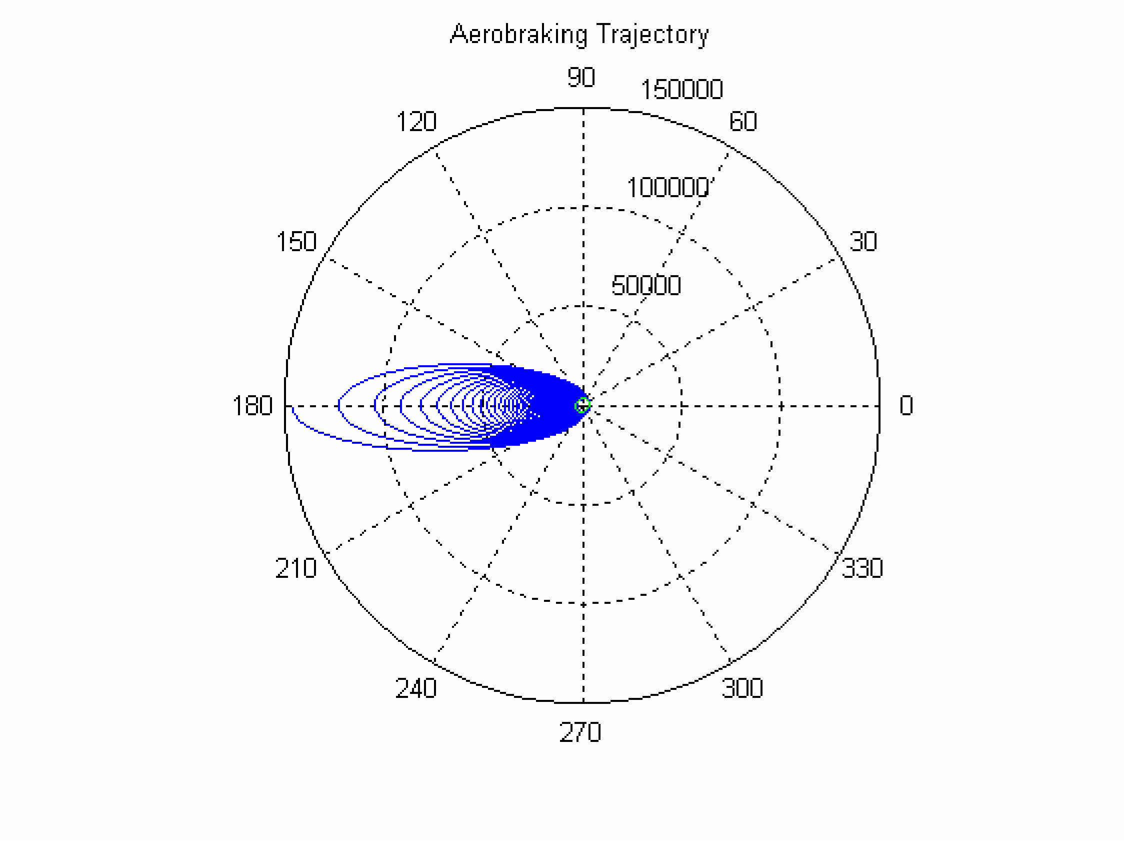


Figure 13 Aero-braking Trajectory

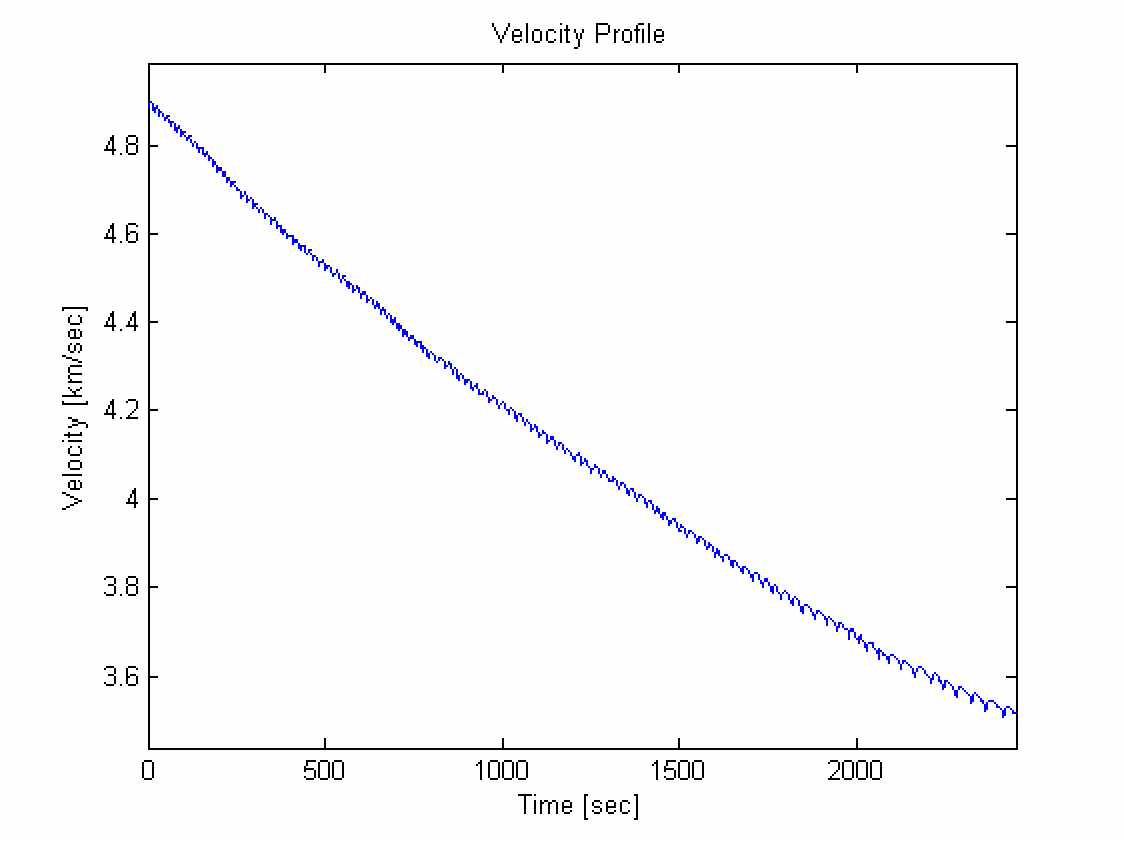


Figure 14 Velocity Profile

Table 3 Aerobraking Information for each Pass through Mars' Atmopshere

Pass	Location	Time [sec]	Velocity [km/sec]	Density [kg/m^3]	Height [km]	deltav_eq [km/s]
1	enter	311531.6885	4.8927	1.68E-05	97.78467	
1	pariapsis	311636.7337	4.90006	3.77E-05	80.00405	
1	exit	311750.4691	4.87835	1.63E-05	99.34317	0.01436
2	enter	791917.29	4.87986	1.62E-05	99.9664	
2	pariapsis	792035.8766	4.88803	3.75E-05	80.07737	
2	exit	792143.9623	4.86666	1.72E-05	96.78561	0.0132
3	enter	1173116.833	4.87008	1.67E-05	97.98245	
3	pariapsis	1173221.065	4.87778	3.79E-05	79.90883	
3	exit	1173335.973	4.8563	1.68E-05	97.77393	0.01378
4	enter	1487711.432	4.85811	1.65E-05	98.60144	
4	pariapsis	1487826.784	4.86555	3.78E-05	79.96811	
4	exit	1487935.09	4.84448	1.75E-05	96.24226	0.01363
5	enter	1751639.814	4.84715	1.67E-05	98.0638	
5	pariapsis	1751746.719	4.85494	3.81E-05	79.81659	
5	exit	1751861.147	4.83379	1.67E-05	98.17507	0.01335
6	enter	1978219.726	4.83445	1.62E-05	99.85504	
6	pariapsis	1978339.664	4.8428	3.80E-05	79.86443	

6	exit	1978448.331	4.82191	1.77E-05	95.85901	0.01254
7	enter	2174358.083	4.82309	1.63E-05	99.62643	
7	pariapsis	2174467.08	4.83222	3.83E-05	79.73457	
7	exit	2174589.049	4.8101	1.64E-05	99.02641	0.01299
8	enter	2347212.722	4.81351	1.71E-05	97.03382	
8	pariapsis	2347313.508	4.82103	3.83E-05	79.70439	
8	exit	2347435.362	4.79937	1.66E-05	98.31761	0.01414
9	enter	2501031.705	4.80153	1.67E-05	98.03777	
9	pariapsis	2501145.019	4.80913	3.85E-05	79.65715	
9	exit	2501269.65	4.78481	1.63E-05	99.38546	0.01672
10	enter	2638828.551	4.79044	1.69E-05	97.40855	
10	pariapsis	2638927.197	4.79853	3.83E-05	79.70477	
10	exit	2639056.038	4.77599	1.76E-05	96.06117	0.01445
11	enter	2763021.007	4.77816	1.67E-05	98.1584	
11	pariapsis	2763123.456	4.78674	3.85E-05	79.62643	
11	exit	2763252.801	4.76398	1.75E-05	96.24197	0.01418
12	enter	2875611.11	4.76905	1.86E-05	94.22322	
12	pariapsis	2875707.236	4.77482	3.89E-05	79.47936	
12	exit	2875830.772	4.75261	1.71E-05	96.94777	0.01644
13	enter	2978455.869	4.75784	1.92E-05	93.43048	
13	pariapsis	2978561.464	4.76228	3.88E-05	79.5201	
13	exit	2978679.338	4.7401	1.67E-05	97.94095	0.01775
14	enter	3073280.441	4.7439	1.69E-05	97.64676	
14	pariapsis	3073383.712	4.75228	3.89E-05	79.4605	
14	exit	3073514.036	4.72957	1.76E-05	95.91272	0.01433
15	enter	3160622.436	4.73494	1.93E-05	93.23775	
15	pariapsis	3160729.057	4.73931	3.90E-05	79.42049	
15	exit	3160847.696	4.7171	1.68E-05	97.76304	0.01784
16	enter	3241783.476	4.71984	1.64E-05	99.17982	
16	pariapsis	3241900.05	4.72913	3.94E-05	79.26303	
16	exit	3242032.216	4.7047	1.64E-05	98.99041	0.01515
17	enter	3317346.764	4.71064	1.78E-05	95.54998	
17	pariapsis	3317446.855	4.71769	3.93E-05	79.27219	
17	exit	3317577.568	4.69497	1.77E-05	95.75343	0.01567
18	enter	3387701.029	4.69945	1.86E-05	94.35637	
18	pariapsis	3387800.447	4.70564	3.96E-05	79.17194	
18	exit	3387938.95	4.68039	1.62E-05	99.7971	0.01906
19	enter	3453472.357	4.68372	1.62E-05	99.91664	
19	pariapsis	3453602.852	4.6928	3.96E-05	79.17598	
19	exit	3453731.435	4.66785	1.63E-05	99.46391	0.01586
20	enter	3515147.437	4.67177	1.62E-05	99.8146	
20	pariapsis	3515276.018	4.68103	3.98E-05	79.06779	
20	exit	3515403.383	4.65718	1.68E-05	97.8651	0.0146
21	enter	3573158.353	4.66218	1.74E-05	96.39472	

21	pariapsis	3573271.861	4.66957	4.01E-05	78.96667	
21	exit	3573408.115	4.64481	1.65E-05	98.5498	0.01737
22	enter	3627921.451	4.65007	1.70E-05	97.33519	
22	pariapsis	3628041.689	4.65818	4.02E-05	78.93184	
22	exit	3628179.828	4.63231	1.62E-05	99.82851	0.01775
23	enter	3679807.074	4.63756	1.65E-05	98.76129	
23	pariapsis	3679937.93	4.64607	4.01E-05	78.95886	
23	exit	3680068.518	4.6211	1.64E-05	99.00754	0.01646
24	enter	3728918.64	4.62619	1.68E-05	97.84984	
24	pariapsis	3729040.504	4.63488	4.05E-05	78.80139	
24	exit	3729178.573	4.60979	1.65E-05	98.63545	0.01639
25	enter	3775638.239	4.61825	1.99E-05	92.52853	
25	pariapsis	3775750.903	4.62247	4.03E-05	78.866	
25	exit	3775873.394	4.60003	1.65E-05	98.8405	0.01822
26	enter	3820024.752	4.60554	1.90E-05	93.65626	
26	pariapsis	3820127.813	4.61163	4.07E-05	78.71341	
26	exit	3820270.312	4.5866	1.66E-05	98.50364	0.01893
27	enter	3862240.069	4.59076	1.69E-05	97.4376	
27	pariapsis	3862352.709	4.59975	4.07E-05	78.72674	
27	exit	3862503.204	4.57363	1.66E-05	98.4429	0.01713
28	enter	3902484.032	4.57935	1.74E-05	96.33541	
28	pariapsis	3902594.721	4.58765	4.09E-05	78.64556	
28	exit	3902740.987	4.563	1.65E-05	98.64521	0.01636
29	enter	3940947.818	4.56782	1.76E-05	95.90796	
29	pariapsis	3941075.642	4.57474	4.10E-05	78.60608	
29	exit	3941202.558	4.55211	1.66E-05	98.24424	0.01571
30	enter	3977793.932	4.5545	1.67E-05	98.2195	
30	pariapsis	3977919.582	4.56396	4.13E-05	78.45157	
30	exit	3978058.641	4.54032	1.75E-05	96.25823	0.01418
31	enter	4013145.494	4.54359	1.70E-05	97.17419	
31	pariapsis	4013260.173	4.55267	4.12E-05	78.51611	
31	exit	4013415.879	4.52597	1.66E-05	98.40169	0.01762
32	enter	4046988.602	4.52915	1.63E-05	99.77362	
32	pariapsis	4047116.77	4.54007	4.15E-05	78.37063	
32	exit	4047262.89	4.51569	1.64E-05	98.90234	0.01346
33	enter	4079495.208	4.51731	1.63E-05	99.50409	
33	pariapsis	4079622.66	4.5283	4.17E-05	78.32487	
33	exit	4079769.612	4.50406	1.66E-05	98.34486	0.01325
34	enter	4110757.791	4.5051	1.62E-05	99.78486	
34	pariapsis	4110901.596	4.51546	4.18E-05	78.27412	
34	exit	4111038.543	4.491	1.64E-05	99.1805	0.0141
35	enter	4140832.176	4.4928	1.62E-05	99.93874	
35	pariapsis	4140972.541	4.50357	4.21E-05	78.15134	
35	exit	4141117.288	4.47853	1.62E-05	99.86668	0.01426

36	enter	4169824.046	4.48448	1.81E-05	95.08751	
36	pariapsis	4169948.382	4.49185	4.23E-05	78.09373	
36	exit	4170092.301	4.46753	1.68E-05	97.6861	0.01696
37	enter	4197787.943	4.47012	1.66E-05	98.50615	
37	pariapsis	4197933.736	4.47932	4.21E-05	78.15015	
37	exit	4198075.637	4.45369	1.64E-05	99.27979	0.01644
38	enter	4224767.071	4.45856	1.70E-05	97.37806	
38	pariapsis	4224903.308	4.46782	4.25E-05	77.98762	
38	exit	4225048.289	4.44298	1.72E-05	96.86638	0.01558
39	enter	4250831.758	4.44513	1.64E-05	99.13901	
39	pariapsis	4250963.452	4.45675	4.25E-05	77.99671	
39	exit	4251118.843	4.43189	1.66E-05	98.41321	0.01324
40	enter	4276077.6	4.43571	1.74E-05	96.33825	
40	pariapsis	4276208.403	4.44429	4.28E-05	77.86434	
40	exit	4276365.84	4.41713	1.63E-05	99.67414	0.01858
41	enter	4300460.103	4.42217	1.69E-05	97.48741	
41	pariapsis	4300593.863	4.43176	4.30E-05	77.80386	
41	exit	4300754.59	4.40454	1.64E-05	98.93648	0.01763
42	enter	4324054.522	4.41234	1.89E-05	93.88311	
42	pariapsis	4324181.759	4.4191	4.32E-05	77.73745	
42	exit	4324331.472	4.39431	1.70E-05	97.34717	0.01803
43	enter	4346906.475	4.3978	1.71E-05	97.10383	
43	pariapsis	4347041.596	4.40721	4.33E-05	77.67825	
43	exit	4347203.925	4.37998	1.65E-05	98.66082	0.01782
44	enter	4369042.879	4.38468	1.69E-05	97.4989	
44	pariapsis	4369179.506	4.39451	4.35E-05	77.61542	
44	exit	4369341.531	4.36789	1.68E-05	97.83187	0.01679
45	enter	4390506.091	4.37488	1.90E-05	93.67944	
45	pariapsis	4390636.077	4.38167	4.37E-05	77.53885	
45	exit	4390788.283	4.35694	1.72E-05	96.82775	0.01795
46	enter	4411325.646	4.36019	1.72E-05	96.89174	
46	pariapsis	4411470.865	4.36931	4.38E-05	77.48976	
46	exit	4411627.211	4.34338	1.63E-05	99.65537	0.0168
47	enter	4431552.788	4.34678	1.67E-05	98.04088	
47	pariapsis	4431703.33	4.35723	4.40E-05	77.43133	
47	exit	4431860.281	4.33142	1.68E-05	97.67718	0.01536
48	enter	4451239.585	4.33794	1.89E-05	93.86832	
48	pariapsis	4451373.499	4.34503	4.42E-05	77.35862	
48	exit	4451541.871	4.31713	1.62E-05	99.9952	0.02081
49	enter	4470351.783	4.32133	1.64E-05	99.12675	
49	pariapsis	4470505.571	4.33279	4.44E-05	77.29576	
49	exit	4470662.582	4.30794	1.69E-05	97.59061	0.01339
50	enter	4488957.057	4.30843	1.63E-05	99.43041	
50	pariapsis	4489111.768	4.32041	4.45E-05	77.23346	

50	exit	4489276.851	4.29433	1.72E-05	96.89438	0.01409
51	enter	4507072.736	4.29664	1.65E-05	98.58449	
51	pariapsis	4507227.27	4.30798	4.47E-05	77.17154	
51	exit	4507393.479	4.28165	1.71E-05	96.97986	0.01499
52	enter	4524733.068	4.2881	1.89E-05	93.77341	
52	pariapsis	4524871.446	4.29531	4.49E-05	77.10774	
52	exit	4525037.625	4.2691	1.72E-05	96.78821	0.019
53	enter	4541898.159	4.27164	1.66E-05	98.31766	
53	pariapsis	4542046.742	4.28315	4.49E-05	77.0993	
53	exit	4542225.081	4.25542	1.70E-05	97.29629	0.01622
54	enter	4558623.223	4.25852	1.67E-05	98.03757	
54	pariapsis	4558772.382	4.2699	4.51E-05	77.02265	
54	exit	4558951.157	4.24241	1.72E-05	96.90219	0.01611
55	enter	4574919.137	4.24733	1.79E-05	95.36888	
55	pariapsis	4575064.252	4.25654	4.55E-05	76.90085	
55	exit	4575247.257	4.22818	1.65E-05	98.61715	0.01915
56	enter	4590833.472	4.23891	2.18E-05	90.37676	
56	pariapsis	4590979.02	4.2428	4.50E-05	77.06907	
56	exit	4591142.047	4.21599	1.64E-05	98.98643	0.02292
57	enter	4606334.474	4.22294	1.84E-05	94.66326	
57	pariapsis	4606480.868	4.23166	4.58E-05	76.77855	
57	exit	4606665.528	4.20334	1.66E-05	98.35754	0.0196
58	enter	4621474.733	4.20701	1.65E-05	98.56722	
58	pariapsis	4621649.95	4.21799	4.59E-05	76.76097	
58	exit	4621815.508	4.19221	1.73E-05	96.61433	0.0148
59	enter	4636267.034	4.19455	1.69E-05	97.48809	
59	pariapsis	4636434.651	4.20559	4.62E-05	76.64589	
59	exit	4636607.392	4.17941	1.76E-05	95.90731	0.01514
60	enter	4650725.864	4.18358	1.80E-05	95.2521	
60	pariapsis	4650872.855	4.19331	4.62E-05	76.64532	
60	exit	4651070.285	4.16378	1.66E-05	98.35074	0.0198
61	enter	4664820.455	4.16736	1.66E-05	98.45185	
61	pariapsis	4664991.911	4.17952	4.66E-05	76.49276	
61	exit	4665180.195	4.15134	1.68E-05	97.71385	0.01602
62	enter	4678631.538	4.15769	1.84E-05	94.6157	
62	pariapsis	4678779.928	4.16705	4.66E-05	76.49683	
62	exit	4678978.575	4.13795	1.69E-05	97.61628	0.01974
63	enter	4692109.483	4.14404	1.84E-05	94.63849	
63	pariapsis	4692279.885	4.15212	4.69E-05	76.40827	
63	exit	4692455.75	4.12552	1.74E-05	96.35324	0.01852
64	enter	4705264.73	4.12664	1.64E-05	99.08273	
64	pariapsis	4705446.075	4.13984	4.73E-05	76.27456	
64	exit	4705646.485	4.1099	1.63E-05	99.64598	0.01674
65	enter	4718170.885	4.11539	1.70E-05	97.17754	

65	pariapsis	4718341.825	4.12688	4.74E-05	76.21958	
65	exit	4718559.952	4.0943	1.69E-05	97.42159	0.0211
66	enter	4730788.28	4.10039	1.64E-05	99.28132	
66	pariapsis	4730966.707	4.11398	4.75E-05	76.18327	
66	exit	4731168.478	4.08549	1.74E-05	96.44721	0.0149
67	enter	4743140.106	4.08689	1.65E-05	98.78624	
67	pariapsis	4743343.164	4.09875	4.75E-05	76.20688	
67	exit	4743538.45	4.06858	1.62E-05	99.90356	0.01831
68	enter	4755227.534	4.0727	1.63E-05	99.4267	
68	pariapsis	4755423.512	4.08611	4.81E-05	75.98257	
68	exit	4755617.774	4.05871	1.76E-05	96.04498	0.014
69	enter	4767095.672	4.06399	1.88E-05	93.95195	
69	pariapsis	4767269.922	4.07269	4.84E-05	75.90872	
69	exit	4767482.323	4.04186	1.63E-05	99.54282	0.02213
70	enter	4778679.253	4.04651	1.66E-05	98.46738	
70	pariapsis	4778863.9	4.06025	4.84E-05	75.89408	
70	exit	4779082.307	4.03047	1.66E-05	98.24346	0.01604
71	enter	4790055.018	4.03332	1.67E-05	98.12096	
71	pariapsis	4790253.38	4.04613	4.88E-05	75.76794	
71	exit	4790467.647	4.01597	1.66E-05	98.52697	0.01735
72	enter	4801211.644	4.0204	1.69E-05	97.55539	
72	pariapsis	4801410.859	4.03261	4.90E-05	75.69601	
72	exit	4801625.139	4.00294	1.68E-05	97.91523	0.01746
73	enter	4812146.354	4.00611	1.66E-05	98.30911	
73	pariapsis	4812344.585	4.01985	4.92E-05	75.63547	
73	exit	4812568.2	3.99023	1.66E-05	98.29657	0.01588
74	enter	4822883.873	3.99272	1.65E-05	98.7573	
74	pariapsis	4823084.473	4.00702	4.93E-05	75.59049	
74	exit	4823317.975	3.97572	1.64E-05	99.20908	0.017
75	enter	4833438.863	3.98244	1.82E-05	94.84698	
75	pariapsis	4833634.262	3.9927	4.97E-05	75.48939	
75	exit	4833867.781	3.96076	1.62E-05	99.90732	0.02168
76	enter	4843772.996	3.96743	1.74E-05	96.39309	
76	pariapsis	4843988.319	3.97853	4.97E-05	75.48139	
76	exit	4844239.037	3.94226	1.62E-05	99.96812	0.02517
77	enter	4853916.212	3.95195	1.66E-05	98.43958	
77	pariapsis	4854139.404	3.9654	5.01E-05	75.36014	
77	exit	4854398.025	3.929	1.63E-05	99.54749	0.02295
78	enter	4863901.605	3.94097	1.79E-05	95.37328	
78	pariapsis	4864107.476	3.95217	5.03E-05	75.26938	
78	exit	4864342.262	3.92234	1.70E-05	97.2088	0.01864
79	enter	4873690.283	3.92644	1.75E-05	96.27277	
79	pariapsis	4873907.636	3.93851	5.05E-05	75.2054	
79	exit	4874144.379	3.90847	1.70E-05	97.39777	0.01798

80	enter	4883295.728	3.90976	1.63E-05	99.58444	
80	pariapsis	4883523.672	3.92539	5.07E-05	75.14731	
80	exit	4883803.242	3.88874	1.66E-05	98.24087	0.02102
81	enter	4892769.351	3.8996	1.81E-05	95.16016	
81	pariapsis	4892993.435	3.91029	5.10E-05	75.06482	
81	exit	4893258.664	3.8757	1.70E-05	97.19811	0.0239
82	enter	4902037.295	3.88239	1.66E-05	98.54046	
82	pariapsis	4902284.369	3.89657	5.12E-05	74.9962	
82	exit	4902551.732	3.86234	1.72E-05	96.72768	0.02005
83	enter	4911173.577	3.86984	1.71E-05	97.0883	
83	pariapsis	4911402.146	3.88407	5.14E-05	74.92662	
83	exit	4911696.531	3.84725	1.68E-05	97.74849	0.02258
84	enter	4920142.911	3.85484	1.67E-05	98.22464	
84	pariapsis	4920397.731	3.86896	5.17E-05	74.83867	
84	exit	4920665.926	3.83657	1.83E-05	94.71641	0.01827
85	enter	4928978.936	3.84235	1.73E-05	96.56534	
85	pariapsis	4929217.941	3.8558	5.20E-05	74.7528	
85	exit	4929509.302	3.82124	1.79E-05	95.47265	0.02111
86	enter	4937652.282	3.82718	1.70E-05	97.29909	
86	pariapsis	4937913.057	3.84072	5.23E-05	74.67309	
86	exit	4938182.778	3.81007	1.96E-05	92.82589	0.01712
87	enter	4946179.732	3.81154	1.64E-05	98.96137	
87	pariapsis	4946454.883	3.82668	5.26E-05	74.58839	
87	exit	4946770.039	3.78924	1.64E-05	98.96163	0.0223
88	enter	4954604.115	3.80124	1.84E-05	94.6704	
88	pariapsis	4954857.632	3.81295	5.28E-05	74.50776	
88	exit	4955183.403	3.77514	1.65E-05	98.75524	0.0261
89	enter	4962843.261	3.78296	1.64E-05	99.13201	
89	pariapsis	4963133.065	3.79871	5.31E-05	74.43589	
89	exit	4963464.091	3.76093	1.65E-05	98.56372	0.02203
90	enter	4971023.426	3.77519	2.03E-05	91.97392	
90	pariapsis	4971273.278	3.78471	5.34E-05	74.35408	
90	exit	4971602.627	3.74896	1.73E-05	96.58844	0.02623
91	enter	4979000.712	3.75657	1.72E-05	96.75791	
91	pariapsis	4979296.235	3.76989	5.36E-05	74.288	
91	exit	4979625.666	3.73499	1.75E-05	96.09625	0.02158
92	enter	4986921.026	3.74693	2.08E-05	91.36678	
92	pariapsis	4987175.792	3.75618	5.39E-05	74.18735	
92	exit	4987534.149	3.71962	1.70E-05	97.34397	0.0273
93	enter	4994657.914	3.72916	1.82E-05	94.94757	
93	pariapsis	4994959.622	3.74104	5.42E-05	74.12049	
93	exit	4995294.323	3.70844	1.90E-05	93.75334	0.02072
94	enter	5002286.699	3.71263	1.72E-05	96.76326	
94	pariapsis	5002604.751	3.72696	5.45E-05	74.02061	

94	exit	5002994.801	3.68854	1.66E-05	98.51333	0.02409
95	enter	5009795.948	3.69577	1.64E-05	99.08008	
95	pariapsis	5010130.738	3.71328	5.47E-05	73.96486	
95	exit	5010535.012	3.67567	1.68E-05	97.85138	0.0201
96	enter	5017255.138	3.68694	1.96E-05	92.86178	
96	pariapsis	5017574.061	3.69747	5.50E-05	73.86723	
96	exit	5017991.183	3.65904	1.65E-05	98.80003	0.02789
97	enter	5024532.786	3.66929	1.76E-05	96.01468	
97	pariapsis	5024887.15	3.68343	5.53E-05	73.78156	
97	exit	5025289.821	3.64939	1.86E-05	94.2616	0.0199
98	enter	5031739.576	3.65595	1.84E-05	94.60388	
98	pariapsis	5032097.747	3.66906	5.56E-05	73.70073	
98	exit	5032513.733	3.63538	1.91E-05	93.52287	0.02057
99	enter	5038807.858	3.63887	1.72E-05	96.78023	
99	pariapsis	5039213.287	3.65344	5.59E-05	73.62419	
99	exit	5039699.15	3.61394	1.63E-05	99.66913	0.02492
100	enter	5045783.971	3.62325	1.69E-05	97.58108	
100	pariapsis	5046204.508	3.63967	5.63E-05	73.52134	
100	exit	5046731.787	3.59908	1.64E-05	99.21372	0.02416
101	enter	5052690.521	3.61091	1.86E-05	94.2749	
101	pariapsis	5053124.185	3.62327	5.66E-05	73.43902	
101	exit	5053673.085	3.58439	1.63E-05	99.3378	0.02651
102	enter	5059433.89	3.59294	1.68E-05	97.68719	
102	pariapsis	5059932.73	3.60884	5.69E-05	73.35362	
102	exit	5060511.272	3.57109	1.68E-05	97.8191	0.02185
103	enter	5066129.863	3.57985	1.79E-05	95.52147	
103	pariapsis	5066640.663	3.59466	5.72E-05	73.2604	
103	exit	5067292.848	3.55541	1.65E-05	98.84509	0.02444
104	enter	5072683.438	3.56359	1.71E-05	97.0003	
104	pariapsis	5073254.227	3.58036	5.76E-05	73.16769	
104	exit	5073990.152	3.54068	1.66E-05	98.40394	0.02291
105	enter	5079114.838	3.54642	1.66E-05	98.49483	
105	pariapsis	5079785.127	3.56509	5.80E-05	73.06706	
105	exit	5080652.247	3.52504	1.63E-05	99.50157	0.02138
106	enter	5085495.329	3.53457	1.80E-05	95.23402	
106	pariapsis	5086225.885	3.55037	5.83E-05	72.97742	
106	exit	5087260.388	3.51267	1.70E-05	97.34368	0.0219
107	enter	5091624.292	3.51788	1.73E-05	96.56527	
107	pariapsis	5092586.223	3.53518	5.86E-05	72.88355	
107	exit	5094194.207	3.49747	1.67E-05	98.04759	0.02041
108	enter	5097246.912	3.50066	1.67E-05	98.18295	
108	pariapsis	5098809.498	3.52279	5.91E-05	72.75254	
108	pariapsis	5102416.748	3.5215	0.00012137	62.13172	
108	pariapsis	5108595.784	3.5215	0.0001214	62.1288	

10.1.2 Matlab script for aero-braking

10.1.2.1 Aerobraking.m

```
%AEROBRAKINGTRIAL.m
%Meredith Evans, Eric Gustafson, Kim Mrozek
%This code calls on MHV.m and integrates position and velocity. It also
%finds the delta_v needed to burn at apoapsis and initiate the aerobraking.
%Inputs that may change are the altitude, initial radius of pariapsis, period of
%orbit, and MHV mass. The code outputs the velocity, density, altitude,
%deltav_eq and trajectory as a function of time.

%%%%%%%%%%%%%%%%%%%%%%
%Aerobraking Analysis
%%%%%%%%%%%%%%%%%%%%%%
clear all; close all; clc; format long
global P e

%%%%%%%%%%%
%Constants
%%%%%%%%%%%
muE = 398600.485043; mu = 1.3271244E+11; muM = 4.28282868534e+04;
radius_mars = 3397;              %km
S = pi*7^2/4;                    %Cross sectional Area
MHV_mass = 57949.6652266306;     %Current Mass

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Initial Orbit (Mar's Parking Orbit MHV)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
period = 7*24*3600;        %sec
rp_inital = 500;           %km
a_initial = (((period/(2*pi))^2)*muM)^(1/3);
e_initial = 1-(rp_inital/a_initial);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Desired areobraking altitude
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
altitude = 80;                                          %km
rp=altitude + radius_mars;                              %km

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Orbit Parameteres before braking
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r_ap = a_initial*(1+e_initial);                          %apoapsis
a_new = 0.5*(r_ap + rp);
e_new = 1 -(rp/a_new);
P_new = a_new * (1-e_new^2);
v_ellipse_1 = sqrt(2*((muM/r_ap)-(muM/(2*a_initial)))); %old ellipse
v_ellipse_2 = sqrt(2*((muM/r_ap)-(muM/(2*a_new))));     %new ellipse
delta_v_initial = abs(v_ellipse_1 - v_ellipse_2);       %delta_v at first apoapsis

%%%%%%%%%%%%%%%%%
%Mass Propellant
%%%%%%%%%%%%%%%%%
g = 9.81;               % [m/s^2]
Isp = 1007.1;           % Assumed for now
mdot = 1.68;            % based on current engine [kg/s]
mass_f = MHV_mass;      % Current MHV mass [kg]

mass_prop = mass_f*exp(delta_v_initial*10^3/(Isp*g)) - mass_f;
time_burn = mass_prop/mdot;

a = a_new;
e = e_new;
P = P_new;

%%%%%%%%%%%%%%
%Initial State
%%%%%%%%%%%%%%
ra = a*(1+e);
va = sqrt((-muM/a)+((2*muM)/ra));
xo = [ra 0 pi va/ra];
ra_limit = radius_mars + 150;           %km
tf = 3600*14.5;
```

```
t=[];
x=[];
count=1;
while ra >= ra_limit
    count=count+1;
    [t_temp, x_temp] = ode45('MHV', [0 tf], xo', odeset('RelTol',1e-6));
    if length(t) >= 1
        t = [t; t_temp + t(end)];
    else
        t=[t;t_temp];
    end
    x = [x; x_temp];
    ra(count)=max(x_temp(:,1));
    xo=x_temp(end,:);
end
t_end = t(end)/(24*3600);      %days

fid=fopen('MHV Aerobraking Output.txt','w');
passnumber=1;
for i=2:length(t)-1
    if x(i,1) <= 3497 && x(i-1,1) > 3497
        v_enter = sqrt(x(i,2)^2 + (x(i,4)*x(i,1))^2);
        h = x(i,1) - radius_mars;
        fprintf(fid,'%5d %10s %15.5f %15.5f %15.5g %15.5f\n',passnumber, 'enter',
t(i), v_enter, rho_mars(h*1000), h);
    elseif x(i,1) < x(i-1,1) && x(i,1) < x(i+1,1)
        v = sqrt(x(i,2)^2 + (x(i,4)*x(i,1))^2);
        h = x(i,1) - radius_mars;
        fprintf(fid,'%5d   %10s   %15.5f   %15.5f   %15.5g   %15.5f\n',passnumber,
'pariapsis', t(i), v, rho_mars(h*1000), h);
    elseif x(i,1) >= 3497 && x(i-1,1) < 3497
        v_exit = sqrt(x(i,2)^2 + (x(i,4)*x(i,1))^2);
        h = x(i-1,1) - radius_mars;
        deltav_eq=v_enter-v_exit;
        fprintf(fid,'%5d %10s %15.5f %15.5f %15.5g %15.5f %15.5f\n',passnumber,
'exit', t(i), v_exit, rho_mars(h*1000), h, deltav_eq);
```

```
            passnumber=passnumber+1;

        end
    end
    fclose(fid)

    %velocity profile
    count=0
    for i=2:length(t)-1
        if x(i,1) <= 3497
            count=count+1;
            v(count) = sqrt(x(i,2)^2 + (x(i,4)*x(i,1))^2);
            h(count) = x(i,1) - radius_mars;
            density(count)=rho_mars(h(count)*1000);
        else
        end
    end



    % %%%%%%%%%%%%%
    % %DeltaV Raise
    % %%%%%%%%%%%%
    % rp_raise = 150+radius_mars;
    % a_raise = 0.5*(rp_raise + ra);
    % e_raise = 1 -(rp_raise./a_raise);
    % v_raise = sqrt(2.*((muM./ra)-(muM./(2.*a_raise))));
    % delta_raise = abs(v_raise-v_exit);



    %%%%%%%
    %Output
    %%%%%%%
    TOF_aero = t_end
    delta_v_start_aero = delta_v_initial
    mass_prop_start_aero = mass_prop
```

```
blargh = input('Pause!');

%%%%%%%%%%%%%%
%Plots
%%%%%%%%%%%%%
thst_2pi = 0:.01:2*pi;
R_surface = radius_mars*ones(size(thst_2pi));
R_upper = (150+radius_mars)*ones(size(thst_2pi));

figure(1)
% subplot(221)
plot(t,x(:,1))
xlabel('Time [sec]')
ylabel('Position [km]')
title('Position vs. Time')

% subplot(222)
figure(2)
plot(t,x(:,3))
xlabel('Time [sec]')
ylabel('Theta_star [km]')
title('Theta_star vs. Time')

% subplot(223)
figure(3)
polar(x(:,3),x(:,1))
hold on
polar(thst_2pi, R_surface, 'r--')
polar(thst_2pi, R_upper, 'g--')
title('Aerobraking Trajectory')

figure(4)
plot(v)
title('velocity profile')
```

10.1.2.2 MHV.m

```
%MHV.m
```

```
%Meredith Evans, Eric Gustafson, Kim Mrozek
%This code integrates the position and velocity in the r_hat and theta_hat
%direction. This code also includes the drag induced by Mars' atmosphere at
%100 km altitude. In order to obtain reasonable results ballutes will be
%used on the MHV. These ballutes increase drag by almost 50% which is a
%feature we are looking for in aerobraking.

function xdot=MHV(t,x)

global P e

%%%%%%%%%%
%Constants
%%%%%%%%%%
muM = 4.28282868534e4;              %km^3/s^2
radius_mars = 3397;                 %km
Cd = 1.0;                           %drag on sphere/cone/biconic section pg. 281
Cd_ploot = 1.45;
S = (pi*7^2/4);                      %Cross sectional Area
S_ploot = 3455.8;%275.67                     %307
MHV_mass = 57949.6652266306;         %Current Mass kg
ploot_mass = 857;%257                     %kg
mass_heat_shield = ((1/8)*MHV_mass);
mass_decent_prop = 50000;            %kg
total_mass = MHV_mass + mass_heat_shield + ploot_mass + mass_decent_prop;

r = P./ (1 + e*cos(x(3)));
h = r - radius_mars;
rho = rho_mars(h*1000);
v_mag=norm([sqrt(muM/P)*(e*sin(x(3))) sqrt(muM/P)*(1+(e*cos(x(3))))]);

%%%%%
%Drag
%%%%%
drag1=(0.5*rho*(sqrt(muM/P)*(e*sin(x(3))))*v_mag*S*Cd)/total_mass;
drag1_ploot=(0.5*rho*(sqrt(muM/P)*(e*sin(x(3))))*v_mag*S_ploot*Cd_ploot)/total_mas
s;
```

```
drag2=(0.5*rho*(sqrt(muM/P)*(1+(e*cos(x(3)))))*v_mag*S*Cd)/total_mass;
drag2_ploot=(0.5*rho*(sqrt(muM/P)*(1+(e*cos(x(3)))))*v_mag*S_ploot*Cd_ploot)/total
_mass;


%%%%%%%%%%%%
%State Space
%%%%%%%%%%%%
xdot(1)=x(2);
xdot(2)=(-muM/x(1)^2)+(x(1)*x(4)^2)-((drag1 + drag1_ploot)/1000);
xdot(3)=x(4);
xdot(4)=(-2*x(2)*x(4))/x(1)-((drag2 + drag2_ploot)/1000);
xdot=xdot';
return
```

10.1.2.3 Rhomars.m

```
%RHOMARS.m
function rho_output = rho_mars(h)


%rho is in kg/m^3
%height is in meters


if h<=100.1e3
    T = -23.4 - .00222*h;
    p = .699 * exp(-.00009*h);
    rho_output = p / (.1921 * (T+ 273.1));
else
    rho_output = 0;
end
```

10.1.3 MHV Aero-braking Raising Periapsis

For a situation when the MHV needs to raise periapsis there is extra 70 km/s ΔV. This cost was found based on the worse case being the MHV having to raise periapsis during the last pass through Mars' atmosphere. Any time before that costs less. A summary of ΔV cost for each pass can be seen in Table 4.

Table 4 ΔV Cost to Raise Periapsis for each Pass

ΔV [km/s]	Mass Propellent [kg]	Burn Time [sec]
0.00	5.41	3.22

0.01	47.37	28.19
0.01	47.31	28.16
0.01	47.04	28.00
0.01	46.46	27.65
0.01	45.16	26.88
0.01	41.37	24.63
0.01	40.80	24.29
0.01	44.11	26.26
0.01	44.86	26.70
0.01	45.08	26.83
0.01	44.29	26.36
0.01	39.28	23.38
0.01	41.68	24.81
0.01	42.68	25.40
0.01	42.30	25.18
0.01	37.81	22.51
0.01	40.10	23.87
0.01	38.24	22.76
0.01	37.11	22.09
0.01	34.75	20.69
0.01	34.12	20.31
0.01	30.55	18.19
0.01	26.91	16.02
0.01	23.15	13.78
0.00	19.26	11.46
0.00	10.48	6.24
0.00	5.20	3.09
0.00	8.94	5.32
0.01	25.77	15.34
0.01	51.61	30.72
0.02	90.53	53.89
0.03	149.02	88.70

10.2 MHV Ballute and Parachute Appendix

See appendix by Jason Friel

10.3 Additional Work

10.3.1 CTV spin-up

Author(s): Meredith Evans

Here the CTV to represents a dumb bell configuration. For purposes of calculating the intertia each compartment is assumed to be a sphere and the booms are assumed to be thin rods. Each side is assumed to be a mass of 100 metric tons, the middle is 200 metric tons and the booms are 25 metric tons. Based on the inertias the force needed to spin the CTV up is calculated using equations 1 -4. The results are seen in Table 5 and the calculations carried out in Matlab are seen below.

$$M_x = I_x\dot{\omega}_x + (I_z - I_y)\omega_y\omega_z \text{ Equation 1}$$

$$M_y = I_y\dot{\omega}_y + (I_x - I_y)\omega_y\omega_x \text{ Equation 2}$$

$$M_z = I_z\dot{\omega}_z + (I_y - I_x)\omega_y\omega_x \text{ Equation 3}$$

$$\dot{\omega}_z = \frac{M_z}{I_z} \text{ Equation 4}$$

$$\omega_z = \frac{M_z}{I_z}t \text{ Equation 5}$$

Table 5 Force Applied During Spin-Up

Time (day)	0.01	0.02	0.04	1	2	3
Force (N)	218	73	36	2	0.8	0.5

10.3.2 Matlab script for CTV spin-up

```
%Meredith Evans
%AAE 450
%dumb bell with equal mass on each side


clear all;close all;clc
M=400000;        % Total Mass   - kg
M1=1/3*M;           % Mass of one side
M2=2/3*M;           % Mass of crew side
R=0.025;         %distance from CM to crew - km
r=0.005;         %diameter of sphere - km
I_sphere_1=[2/5*M1*r^2 0 0; 0 2/5*M1*r^2 0; 0 0 2/5*M1*r^2]; %Inertia kg-m^2
I_sphere_2=[2/5*M2*r^2 0 0; 0 2/5*M2*r^2 0; 0 0 2/5*M2*r^2]; %kg-m^2
I_1=I_sphere_1 + [0 0 0; 0 R^2 0; 0 0 R^2];
I_2=I_sphere_2 + [0 0 0; 0 R^2 0; 0 0 R^2];
```

```
I=I_1+I_2;
I_z=I(3,3);
big_omega=0;
w_z=5.457877*60*2*pi;
time=[600,1800,3600,86400,172800,259200,345600]; %1 day - seconds
M_z=w_z*I_z./time;
F=M_z./(2*r);
time_day=time/3600;
g=1./time_day;            %change in acceleration (g's) per hour
a=F/M;                    %assuming constant acceleration
delta_v=a.*time;          %change in velocity
wz = M_z(2).*900./I_z;    %angular velocity
F1=(wz*I_z./900)./(2*r);  %total force
```

10.4 CTV Free-return Launch Windows

Author(s): Meredith Evans

Contributor(s): Jackie Jaron, Kevin Kloster

Using MIDAS the given launch dates with the corresponding ΔV and $V_{infinity}$ information are found for each launch window. These launch dates can be seen in Table 6 and the file used to run MIDAS can be seen below. In this file Earth is defined as the departure and arrival location while Mars is defined as the fly by body. An initial departure date and time of flight is defined before running the program. The Julian date and time of flight are variables when running MIDAS to allow the program to find free-return trajectories (the script used in MIDAS can be seen below). After running the input file, the data was analyzed to make sure the total time of flight was 3 years and that the ΔV at the Mars fly by was 0 kilometers per second. Note that this information was compiled by both Meredith Evans and Jackie Jaron. Further analysis was done by Kevin Kloster.

Table 6 CTV Free-return Launch Window Dates

Year	Month	Day	Hour	Second	Delta_V	V_infinity	C^3	
Departure	2016	2	21	9	34	3.74		11.69
Mars fly by	2016	7	21	22	22	0.00	7.71	
Arrival	2019	2	20	16	21	0.51	3.42	
Departure	2018	5	5	9	44	3.77		12.40

Mars fly by	2018	9	15	16	8	0.00	6.09	
Arrival	2021	5	3	8	38	0.54	3.52	
Departure	2020	7	18	21	35	3.80		13.1797
Mars fly by	2021	1	27	17	52	0.00	2.86	
Arrival	2023	7	3	16	22	0.63	3.81	
Departure	2022	9	8	16	14	4.03		18.4904
Mars fly by	2023	4	5	23	21	0.00	3.46	
Arrival	2025	8	18	19	27	1.32	5.59	
Departure	2024	10	16	12	52	4.14		21.1816
Mars fly by	2025	6	30	7	52	0.00	2.97	
Arrival	2027	11	3	3	18	0.99	4.81	
Departure	2026	11	10	3	2	3.78		12.6478
Mars fly by	2027	6	27	3	49	0.00	4.36	
Arrival	2029	11	4	20	44	0.58	3.63	

10.4.1 Input file used in MIDAS to find launch windows:

```
HEAD='Mars Free Return'
SHOTA='EARTH'
BULSI='EARTH'
BODY='Mars'
JDL=2026,12,01
nda=1
ndb=2
rn(2)=-1
ALTB=350
re=300
jdate=0.0
tpb=150
adate=1100
varyi='adate','tpb','jdate'
loops=50, $
$END
```

10.5 MHV Trajectory Burn Analysis

Author(s): Meredith Evans

Contributor(s): Eric Gustafson, Kim Mrozek

This work is a continuation of Kim Mrozeks work on the MHV hohmann transfer launch dates. A data sheet including radius and velocity information for Earth and Mars is compiled using information from Satellite Tool Kit (STK). The script *main.m* seen below was then written such that it would call on this data from STK and the function script *orbit3.m* seen below and find ΔV information. In more detail, this script finds the best ΔV and the corresponding time of flight for a range of Julian dates.

As a note, the radius vectors obtained from STK are very close to being 180 degrees apart. Because of this, we were unable to find the cross product of these vectors in Matlab. To solve this problem we assumed the orbits to be coplanar. Therefore, to compensate for this assumption there is an inclination change halfway through the transfer.

Based on the ΔV the mass of propellant and corresponding burn time are found using a mass flow rate of 1.68 kilograms per second squared and an I_{sp} of 1000 seconds. Table 8 - Table 26 summarize the burn analysis for the MHV for each maneuver except for aerobraking and for each launch window. A overall summary is found in Table 26. Addition information such as the semi-major axis, $V_{inifinity}$, and transfer angle is also seen in the tables as extra information to aid fellow students in their analysis.

Note that this work was a joint effort by Meredith Evans, Eric Gustafson, and Kim Mrozek.

Table 26 Summary of ΔV Cost for Optimal Launch Day

	Window 2	Window 3	Window 4	Window 5	Window 6	Window 7
Year	2016	2018	2020	2022	2024	2027
TOF [days]	327	246	212	287	280	261
Total ΔV [km/sec]	5.81	6.08	6.38	6.60	6.43	6.19

Table 8 MHV Launch Dates and Burn Analysis for 2016

Julian date	ΔV 1 [km/s]	Mass Propellant [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant [kg]	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant [kg]	Burn Time [sec]
2457485	3.63	42244.14	19832.93	0.00	0.04	0.02	2.10	18239.32	8563.06
2457486	3.63	42173.72	19799.87	0.00	4.34	2.04	2.10	18222.03	8554.94
2457488	3.63	42265.62	19843.02	0.00	5.78	2.71	2.10	18190.53	8540.16
2457490	3.65	42425.74	19918.19	0.00	5.86	2.75	2.10	18169.44	8530.26
2457492	3.66	42627.96	20013.13	0.00	0.39	0.18	2.10	18158.27	8525.01
2457495	3.69	43003.41	20189.40	0.00	5.70	2.67	2.09	18153.13	8522.60
2457505	3.81	44797.97	21031.91	0.00	10.17	4.78	2.11	18270.86	8577.87
2457515	3.85	46003.74	21598.00	0.18	1735.78	814.92	2.06	17794.57	8354.26
2457525	3.92	47532.84	22315.89	0.34	3332.28	1564.45	2.03	17523.71	8227.09
2457535	3.98	49245.54	23119.97	0.50	4860.88	2282.10	2.02	17479.94	8206.54
2457545	4.05	51099.01	23990.14	0.63	6205.23	2913.25	2.05	17753.78	8335.11

2457555	4.14	53519.99	25126.76	0.74	7447.47	3496.47	2.12	18399.44	8638.23
2457565	4.25	56421.54	26488.99	0.85	8624.35	4048.99	2.23	19496.18	9153.13
2457575	4.36	59730.12	28042.31	0.94	9727.78	4567.03	2.40	21134.21	9922.16

Table 28 MHV Orbit Characteristics for 2016

Julian date	TOF [days]	$V_{infinity}$ Departure [km/s]	$V_{infinity}$ Arrival [km/s]	Transfer Angle [rad]	Semi_major Axis [km]
2457485	326.81	3.02	2.72	3.711	1.8915E+08
2457486	325.77	3.00	2.71	3.693	1.8915E+08
2457488	323.87	3.03	2.71	3.660	1.8916E+08
2457490	321.87	3.08	2.70	3.626	1.8915E+08
2457492	319.81	3.13	2.70	3.591	1.8912E+08
2457495	316.87	3.23	2.70	3.540	1.8908E+08
2457505	306.91	3.64	2.72	3.370	1.8877E+08
2457515	314.50	3.78	2.62	3.370	1.9104E+08
2457525	321.42	3.96	2.57	3.360	1.9322E+08
2457535	328.58	4.15	2.56	3.347	1.9543E+08
2457545	334.65	4.34	2.61	3.322	1.9742E+08
2457555	340.49	4.59	2.75	3.291	1.9928E+08
2457565	346.53	4.86	2.97	3.259	2.0109E+08
2457575	352.52	5.12	3.27	3.224	2.0277E+08

Table 29 MHV Total ΔV (before aero-braking) for 2016

Julian date	Tot. ΔV [km/s]	Total Mass Propellant [kg]
2457485	5.74	60484
2457486	5.73	60400
2457488	5.73	60462
2457490	5.74	60601
2457492	5.76	60787
2457495	5.78	61162
2457505	5.92	63079
2457515	6.09	65534
2457525	6.29	68389
2457535	6.50	71586
2457545	6.73	75058
2457555	7.01	79367
2457565	7.33	8454
2457575	7.69	90592

Table 30 31 MHV Launch Dates and Burn Analysis for 2018

Julian date	ΔV 1 [km/s]	Mass Propellant [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant [kg]	Burn Time [sec]
2458253	3.58	42913.33	20147.10	0.00	3.04	1.43	2.43	21410.44	10051.85
2458254	3.58	42943.48	20161.26	0.00	12.91	6.06	2.42	21366.85	10031.38
2458256	3.59	43028.61	20201.22	0.00	3.04	1.43	2.42	21314.90	10007.00
2458258	3.60	43145.16	20255.94	0.00	0.45	0.21	2.41	21260.99	9981.69
2458260	3.61	43300.29	20328.78	0.00	13.77	6.46	2.41	21212.79	9959.06
2458263	3.63	43585.53	20462.69	0.00	11.31	5.31	2.40	21178.26	9942.85
2458273	3.73	44969.58	21112.48	0.00	7.89	3.70	2.41	21258.02	9980.29
2458283	3.86	47124.32	22124.09	0.00	0.79	0.37	2.46	21682.61	10179.63
2458293	3.77	46651.10	21901.92	0.77	7488.68	3515.81	1.87	16029.42	7525.55
2458303	3.84	47908.16	22492.09	0.84	8248.25	3872.42	1.83	15627.84	7337.02
2458313	3.94	49624.91	23298.08	0.90	8865.99	4162.44	1.83	15622.67	7334.59
2458323	4.08	52322.50	24564.56	0.96	9480.31	4450.85	1.87	16019.15	7520.73
2458333	4.26	56010.00	26295.78	1.01	10143.85	4762.37	1.97	16957.51	7961.27
2458343	4.47	60794.55	28542.04	1.06	10849.35	5093.59	2.14	18609.33	8736.77

Table 32 MHV Orbit Characterisitcs for 2018

Julian date	TOF [days]	$V_{infinity}$ Departure [km/s]	$V_{infinity}$ Arrival [km/s]	Transfer Angle [rad]	Semi_major Axis [km]
2458253	245.78	2.82	3.32	3.073	1.8565E+08
2458254	244.68	2.83	3.31	3.056	1.8566E+08
2458256	242.84	2.86	3.30	3.024	1.8569E+08
2458258	240.81	2.90	3.29	2.990	1.8569E+08
2458260	238.68	2.95	3.28	2.955	1.8567E+08
2458263	235.70	3.03	3.28	2.905	1.8564E+08
2458273	225.73	3.36	3.29	2.738	1.8527E+08
2458283	215.82	3.80	3.36	2.571	1.8448E+08
2458293	295.45	3.51	2.22	3.217	1.9528E+08
2458303	298.42	3.74	2.12	3.159	1.9645E+08
2458313	299.42	4.03	2.11	3.083	1.9719E+08
2458323	300.49	4.42	2.22	3.006	1.9773E+08
2458333	302.63	4.88	2.44	2.936	1.9822E+08
2458343	305.41	5.38	2.79	2.870	1.9861E+08

Table 33 MHV Total ΔV (before aero-braking) for 2018

Julian date	Tot. ΔV [km/s]	Total Mass Propellant [kg]
2458253	6.01	64326.82
2458254	6.01	64323.23
2458256	6.01	64346.55
2458258	6.01	64406.60
2458260	6.02	64526.85
2458263	6.04	64775.10
2458273	6.14	66235.50
2458283	6.32	68807.71
2458293	6.41	70169.20
2458303	6.52	71784.24
2458313	6.67	74113.57
2458323	6.91	77821.96
2458333	7.24	83111.36
2458343	7.67	90253.23

Table 34 MHV Launch Dates and Burn Analysis for 2020

Julian date	ΔV 1 [km/s]	Mass Propellant [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant [kg]	Burn Time [sec]
2459047	3.75	46018.96	21605.15	0.65	6381.71	2996.11	1.90	16295.83	7650.63
2459048	3.75	46004.58	21598.40	0.66	6444.42	3025.55	1.90	16255.27	7631.58
2459050	3.75	46023.55	21607.30	0.66	6457.31	3031.60	1.90	16276.67	7641.63
2459052	3.75	46034.92	21612.64	0.67	6525.45	3063.59	1.89	16251.34	7629.74
2459054	3.75	46073.98	21630.98	0.68	6592.43	3095.04	1.89	16230.50	7619.95
2459057	3.76	46206.53	21693.21	0.68	6673.61	3133.15	1.89	16223.62	7616.72
2459067	3.82	47224.42	22171.09	0.72	7034.88	3302.76	1.89	16205.06	7608.01
2459077	3.94	49347.65	23167.91	0.75	7393.97	3471.35	1.91	16382.31	7691.23
2459087	3.95	50551.34	23733.02	1.08	10625.25	4988.38	1.80	15379.13	7220.25
2459097	4.17	54440.99	25559.15	1.09	10857.27	5097.31	1.87	16033.63	7527.52
2459107	4.48	60419.88	28366.14	1.11	11203.52	5259.87	2.00	17223.82	8086.30
2459117	4.92	69458.12	32609.45	1.14	11694.39	5490.32	2.20	19195.47	9011.96
2459127	5.46	82454.05	38710.82	1.15	12334.03	5790.62	2.55	22625.94	10622.51
2459137	6.02	99436.07	46683.60	1.15	13037.70	6120.99	3.12	28490.92	13376.02

Table 35 MHV Orbit Characterisitcs for 2020

Julian date	TOF [days]	$V_{infinity}$ Departure [km/s]	$V_{infinity}$ Arrival [km/s]	Transfer Angle [rad]	Semi_major Axis [km]
2459047	211.52	3.46	2.28	2.681	1.9740E+08

2459048	211.43	3.45	2.27	2.673	1.9740E+08
2459050	209.58	3.45	2.28	2.641	1.9747E+08
2459052	208.55	3.45	2.27	2.616	1.9747E+08
2459054	207.48	3.45	2.27	2.590	1.9746E+08
2459057	205.58	3.48	2.27	2.550	1.9741E+08
2459067	200.38	3.66	2.26	2.424	1.9680E+08
2459077	194.53	4.04	2.30	2.292	1.9556E+08
2459087	281.46	4.06	2.05	2.900	1.9887E+08
2459097	274.63	4.65	2.22	2.756	1.9774E+08
2459107	269.54	5.41	2.50	2.624	1.9632E+08
2459117	265.56	6.33	2.91	2.500	1.9459E+08
2459127	265.86	7.36	3.52	2.407	1.9305E+08
2459137	272.12	8.33	4.40	2.359	1.9219E+08

Table 36 MHV Total ΔV (before aero-braking) for 2020

Julian date	Tot. ΔV [km/s]	Total Mass Propellant [kg]
2459047	6.31	68696.50
2459048	6.31	68704.27
2459050	6.31	68757.53
2459052	6.32	68811.71
2459054	6.32	68896.91
2459057	6.34	69103.76
2459067	6.43	70464.36
2459077	6.60	73123.94
2459087	6.83	76555.71
2459097	7.13	81331.89
2459107	7.59	88847.22
2459117	8.25	100347.98
2459127	9.16	117414.01
2459137	10.29	140964.68

Table 37 MHV Launch Dates and Burn Analysis for 2022

Julian date	ΔV 1 [km/s]	Mass Propellant [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant [kg]	Burn Time [sec]
2459823	3.79	47377.77	22243.09	0.98	9611.32	4512.36	1.76	14979.07	7032.43
2459824	3.79	47384.41	22246.20	0.98	9594.86	4504.63	1.76	14993.95	7039.41
2459826	3.79	47443.13	22273.77	0.98	9587.43	4501.14	1.76	15000.74	7042.60
2459828	3.80	47551.66	22324.72	0.98	9594.99	4504.69	1.76	14999.56	7042.05

2459830	3.79	47471.11	22286.91	0.98	9596.92	4505.59	1.76	15012.56	7048.15
2459833	3.79	47422.88	22264.26	0.98	9606.56	4510.12	1.77	15040.64	7061.33
2459843	3.82	48042.91	22555.36	0.98	9654.89	4532.81	1.79	15269.35	7168.71
2459853	3.96	50460.72	23690.48	1.00	9808.00	4604.69	1.83	15679.25	7361.15
2459863	4.25	55484.00	26048.83	1.01	10055.99	4721.12	1.91	16392.23	7695.88
2459873	4.69	63710.24	29910.91	1.03	10412.40	4888.45	2.04	17583.98	8255.39
2459883	5.31	76455.79	35894.74	1.06	10910.18	5122.15	2.24	19552.33	9179.50
2459893	6.13	95820.83	44986.30	1.08	11535.44	5415.70	2.58	22938.78	10769.38
2459903	7.06	123056.64	57773.07	1.08	12178.48	5717.60	3.16	29016.48	13622.76
2459913	5.96	127663.18	59935.76	0.00	2.90	1.36	6.88	77336.12	36308.04

Table 38 MHV Orbit Characterisitcs for 2022

Julian date	TOF [days]	$V_{infinity}$ Departure [km/s]	$V_{infinity}$ Arrival [km/s]	Transfer Angle [rad]	Semi_major Axis [km]
2459823	286.63	3.56	1.94	3.222	1.9994E+08
2459824	286.18	3.56	1.95	3.209	1.9994E+08
2459826	284.47	3.58	1.95	3.178	1.9992E+08
2459828	282.32	3.60	1.95	3.143	1.9989E+08
2459830	280.44	3.58	1.95	3.110	1.9984E+08
2459833	277.52	3.57	1.96	3.059	1.9974E+08
2459843	268.52	3.67	2.02	2.897	1.9910E+08
2459853	258.51	4.09	2.13	2.726	1.9795E+08
2459863	248.60	4.85	2.31	2.555	1.9620E+08
2459873	239.38	5.87	2.58	2.388	1.9382E+08
2459883	231.39	7.10	2.98	2.229	1.9080E+08
2459893	226.52	8.52	3.57	2.093	1.8738E+08
2459903	227.53	9.95	4.47	2.002	1.8418E+08
2459913	342.33	8.23	9.05	2.838	1.9631E+08

Table 39 MHV Total ΔV (before aero-braking) for 2022

Julian date	Tot. ΔV [km/s]	Total Mass Propellant [kg]
2459823	6.53	71968.16
2459824	6.53	71973.22
2459826	6.53	72031.30
2459828	6.54	72146.21
2459830	6.54	72080.58
2459833	6.54	72070.07
2459843	6.59	72967.15
2459853	6.79	75947.96
2459863	7.17	81932.22

2459873	7.76	91706.62
2459883	8.61	106918.30
2459893	9.80	130295.05
2459903	11.29	164251.60
2459913	12.84	205002.19

Table 40 MHV Launch Dates and Burn Analysis for 2024

Julian date	ΔV 1 [km/s]	Mass Propellant [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant [kg]	Burn Time [sec]
2460585	3.73	46039.31	21614.70	0.64	6316.44	2965.46	1.99	17124.24	8039.55
2460586	3.73	46053.42	21621.32	0.64	6302.60	2958.97	1.99	17150.55	8051.90
2460588	3.74	46115.07	21650.27	0.64	6296.96	2956.32	1.99	17148.96	8051.15
2460590	3.74	46158.31	21670.57	0.64	6283.77	2950.13	1.99	17184.58	8067.88
2460592	3.73	46063.79	21626.19	0.64	6284.97	2950.69	1.99	17194.43	8072.50
2460595	3.73	46034.47	21612.43	0.64	6288.89	2952.53	2.00	17233.64	8090.91
2460605	3.78	46903.72	22020.52	0.65	6368.19	2989.76	2.02	17447.64	8191.38
2460615	3.94	49624.07	23297.68	0.66	6578.45	3088.47	2.06	17785.68	8350.08
2460625	4.27	55161.86	25897.59	0.69	6907.64	3243.02	2.12	18348.68	8614.40
2460635	4.79	64660.99	30357.27	0.73	7406.54	3477.25	2.21	19242.24	9033.92
2460645	5.54	79402.48	37278.16	0.79	8080.75	3793.78	2.35	20686.43	9711.94
2460655	6.54	102256.97	48007.97	0.84	8896.95	4176.97	2.61	23270.13	10924.94
2460665	7.70	135931.16	63817.45	0.84	9363.51	4396.01	3.16	28999.61	13614.84
2460675	4.68	102401.32	48075.74	1.09	17691.65	8305.94	6.70	74594.58	35020.93

Table 41 MHV Orbit Characterisitcs for 2024

Julian date	TOF [days]	$V_{infinity}$ Departure [km/s]	$V_{infinity}$ Arrival [km/s]	Transfer Angle [rad]	Semi_major Axis [km]
2460585	279.69	3.38	2.48	3.213	1.9698E+08
2460586	278.92	3.38	2.48	3.198	1.9696E+08
2460588	277.02	3.40	2.48	3.164	1.9693E+08
2460590	275.28	3.41	2.49	3.132	1.9688E+08
2460592	273.32	3.38	2.49	3.098	1.9682E+08
2460595	270.40	3.37	2.50	3.047	1.9669E+08
2460605	260.07	3.54	2.55	2.871	1.9603E+08
2460615	248.49	4.04	2.62	2.684	1.9492E+08
2460625	236.14	4.90	2.74	2.491	1.9321E+08
2460635	222.74	6.09	2.92	2.288	1.9082E+08
2460645	208.95	7.50	3.19	2.082	1.8759E+08
2460655	196.57	9.16	3.63	1.886	1.8335E+08

2460665	193.18	10.89	4.47	1.760	1.7842E+08
2460675	400.00	5.84	8.85	3.581	1.9920E+08

Table 42 MHV Total ΔV (before aero-braking) for 2024

Julian date	Tot. ΔV [km/s]	Total Mass Propellant [kg]
2460585	6.36	69479.99
2460586	6.36	69506.56
2460588	6.37	69560.99
2460590	6.37	69626.66
2460592	6.37	69543.18
2460595	6.37	69557.00
2460605	6.45	70719.55
2460615	6.66	73988.19
2460625	7.07	80418.18
2460635	7.74	91309.77
2460645	8.68	108169.66
2460655	9.99	134424.05
2460665	11.70	174294.28
2460675	12.47	194687.55

Table 43 MHV Launch Dates and Burn Analysis for 2026

Julian date	ΔV 1 [km/s]	Mass Propellant [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant [kg]	Burn Time [sec]
2461353	3.64	44037.21	20674.75	0.14	1366.47	641.54	2.34	20549.83	9647.81
2461354	3.64	44029.58	20671.16	0.14	1364.21	640.47	2.34	20564.50	9654.69
2461356	3.64	44058.95	20684.95	0.14	1368.10	642.30	2.34	20580.64	9662.27
2461358	3.65	44162.51	20733.57	0.14	1364.23	640.48	2.35	20635.88	9688.21
2461360	3.66	44325.67	20810.17	0.14	1377.21	646.58	2.35	20657.08	9698.16
2461363	3.68	44714.91	20992.92	0.14	1389.79	652.48	2.36	20740.30	9737.23
2461373	3.84	47247.27	22181.82	0.15	1514.93	711.24	2.39	21053.31	9884.18
2461383	4.13	52043.49	24433.56	0.17	1723.90	809.34	2.45	21589.15	10135.75
2461393	4.61	60196.79	28261.40	0.20	2045.91	960.52	2.53	22419.92	10525.78
2461403	5.30	73078.93	34309.36	0.24	2456.57	1153.32	2.66	23741.65	11146.31
2461413	6.20	92229.64	43300.30	0.28	2999.80	1408.36	2.86	25773.29	12100.14
2461423	3.91	66696.04	31312.69	1.00	13262.04	6226.31	4.72	47108.81	22116.81
2461433	4.05	68679.47	32243.88	0.95	12487.20	5862.53	4.65	46178.97	21680.27
2461443	4.20	70791.47	33235.43	0.89	11573.42	5433.53	4.58	45298.17	21266.74

Table 44 MHV Orbit Characterisitcs for 2026

Julian date	TOF [days]	$V_{infinity}$ Departure [km/s]	$V_{infinity}$ Arrival [km/s]	Transfer Angle [rad]	Semi_major Axis [km]
2461353	261.38	3.06	3.16	3.078	1.9155E+08
2461354	260.41	3.06	3.17	3.061	1.9151E+08
2461356	258.38	3.06	3.17	3.025	1.9143E+08
2461358	256.44	3.09	3.18	2.991	1.9132E+08
2461360	254.32	3.12	3.18	2.955	1.9122E+08
2461363	251.23	3.21	3.20	2.901	1.9102E+08
2461373	240.07	3.74	3.25	2.715	1.9011E+08
2461383	228.20	4.57	3.35	2.522	1.8871E+08
2461393	215.43	5.69	3.49	2.321	1.8670E+08
2461403	202.28	7.06	3.70	2.117	1.8394E+08
2461413	188.67	8.62	4.01	1.908	1.8031E+08
2461423	400.00	3.96	6.53	3.941	1.9630E+08
2461433	400.00	4.34	6.44	3.874	1.9693E+08
2461443	400.00	4.73	6.35	3.807	1.9758E+08

Table 26 MHV Total ΔV (before aero-braking) for 2026

Julian date	Tot. ΔV [km/s]	Total Mass Propellant [kg]
2461353	6.12	65953.52
2461354	6.12	65958.28
2461356	6.12	66007.69
2461358	6.14	66162.62
2461360	6.15	66359.96
2461363	6.18	66845.00
2461373	6.38	69815.51
2461383	6.75	75356.55
2461393	7.34	84662.61
2461403	8.19	99277.15
2461413	9.34	121002.73
2461423	9.64	127066.89
2461433	9.65	127345.64
2461443	9.67	127663.05

10.5.1 Matlab script for MHV burn analysis

10.5.1.1 Main.m

```
%MAIN.m
%Meredith Evans, Eric Gustafson, Kimberly Mrozek
%This is an improved version of the code that calculates the
%optimal delta V (and TOF) for a given range of launch dates
%for the HAB trajectory.  The optimization process was improved so that the
%code now runs much faster.
%This code calls on "orbit3" and runs for a specified range of julian
%dates.
clear all; close all; clc; warning off;
tic
format long
counter=0;
jdate_vector = 2461353+[0,1,3,5,7,10:10:90];
%jdate_vector = 2457300:10:2457600;
for jdate= jdate_vector
    fprintf('%5.2f%%    done...\n',    (jdate-jdate_vector(1))/(jdate_vector(end)-
jdate_vector(1))*100);
    counter=counter+1;
    best_TOF = fminbnd('orbit3_minimize', 100, 400, optimset('TolX', .5), jdate);
    output=orbit3(best_TOF, jdate);
    %     [jdateofcounter, TA, tof_day, delta_v1, delta_v2, deltav_tot]=output
    jdateofcounter(counter)=output(1);
    TA(counter)=output(2);
    tof_day(counter)=output(3);
    delta_v1(counter)=output(4);
    delta_v2(counter)=output(5);
    deltav_tot(counter)=output(6);
    a_a(counter)=output(7);
    deltav_inc_change(counter)=output(8);
    mass_prop_A(counter)=output(9);
    time_burn_A(counter)=output(10);
```

```
        mass_prop_inc(counter)=output(11);
        time_burn_inc(counter)=output(12);
        mass_prop_D(counter)=output(13);
        time_burn_D(counter)=output(14);
        mass_prop_total(counter)=output(15);
        v_infinity_D(counter)=output(16);
        v_infinity_A(counter)=output(17);
    end

      TA=TA';
      tof_day=tof_day';
      delta_v1=delta_v1';
      delta_v2=delta_v2';
      deltav_tot=deltav_tot';
      a_a=a_a';
      deltav_inc_change=deltav_inc_change';
      mass_prop_A = mass_prop_A';
      time_burn_A = time_burn_A';
      mass_prop_inc = mass_prop_inc';
      time_burn_inc = time_burn_inc';
      mass_prop_D = mass_prop_D';
      time_burn_D = time_burn_D';
      mass_prop_total = mass_prop_total';
      v_infinity_D = v_infinity_D';
      v_infinity_A = v_infinity_A';

    %%%%%%%%%
    %Output
    %%%%%%%%%
    dataoutput=[delta_v1 mass_prop_D time_burn_D deltav_inc_change mass_prop_inc...
            time_burn_inc delta_v2 mass_prop_A time_burn_A deltav_tot mass_prop_total
    v_infinity_D v_infinity_A TA a_a tof_day];

    save(['window_', num2str(jdateofcounter(1)), '.txt'],'dataoutput','-ASCII')

    toc
```

```
subplot(2,1,1); plot(jdateofcounter-jdateofcounter(1), tof_day); grid on
ylabel('TOF (day)')
title(num2str(jdateofcounter(1)))
subplot(2,1,2); plot(jdateofcounter-jdateofcounter(1), deltav_tot); grid on
ylabel('Total delta V')
xlabel('Julian Date (Days after first date)')
```

10.5.1.2 Orbit 3.m

```
%ORBIT3.m
%Meredith Evans, Eric Gustafson, Kimberly Mrozek
%This code calls on "planet data" and is used to find the best delta_V %and TOF
for a specified Julian date using a built in MATLAB optimizer, %fminbnd.

function out=orbit3_minimize(TOF, jdate)


counter=0;
%for TOF=180:200
 counter=counter+1;
[r1_vector,    v1_vector,    theta_starE,    r2_vector_junk,    v2_vector_junk,
theta_star_junk] =planet_data(jdate);
[RE_vector_junk,   VE_vector_junk,   theta_star_junk,   r2_vector,   v2_vector,
theta_starM] =planet_data(jdate+TOF);
    r2_vector_actual=r2_vector;
    r1_vector(3)=0;
    r2_vector(3)=0;
    v1_vector(3)=0;
    v2_vector(3)=0;
    r1=norm(r1_vector); r2=norm(r2_vector);
    v1=norm(v1_vector); v2=norm(v2_vector);
    c_vector=r2_vector - r1_vector;
    c=norm(c_vector);
    s=0.5*(c+r1+r2);

    r1_hat=r1_vector/r1;
    r2_hat=r2_vector/r2;
    h=cross(r1_vector,r2_vector);
    del=acos((dot(r1_vector,r2_vector))/(r1*r2));
    if h(3)<0
```

```
        h=-h;
        TA(counter)=(2*pi)-del;
    else
        h=h;
        TA(counter)=del;
    end
    TA(counter);
    muE=398600.485043; mu=1.3271244E+11; muM=4.28282868534e+04;
    h_mag=norm(h);
    h_hat=h/h_mag;

    %Earth transformation matrix
    theta_hat=cross(h_hat,r1_hat);
    trans_matrix=[r1_hat(1) theta_hat(1) h_hat(1);r1_hat(2)
    theta_hat(2) h_hat(2);
    r1_hat(3) theta_hat(3) h_hat(3)];
    inc=acos(h_hat(3));
    omega1=acos(h_hat(2)/-sin(inc));
    omega2=asin(h_hat(1)/sin(inc));
    theta1=acos(theta_hat(3)/sin(inc));
    theta2=asin(r1_hat(3)/sin(inc));

    %Mars transformation matrix
    theta_hat_M=cross(h_hat,r2_hat);
    trans_matrix_M=[r2_hat(1) theta_hat_M(1) h_hat(1);r2_hat(2)
    theta_hat_M(2) h_hat(2);r2_hat(3) theta_hat_M(3) h_hat(3)];

    %inclination change
    r2_hat_actual=r2_vector_actual/norm(r2_vector_actual);
    r1_hat_new=cross(r2_hat_actual,h_hat);
    h_new=cross(r1_hat_new,r2_hat_actual);
    h_new_hat=h_new/norm(h_new);
    inclination=acos(h_new_hat(3));
    incli=inclination*180/pi;

    %%%%%%%%%%%%%%%%%%%%%
    %Minimum Conditions
```

```
%%%%%%%%%%%%%%%%%%%%%

a_min=0.5*s;              %Minimum semi-major axis [Km]
energy_min=-mu/(2*a_min);%Energy associated with a_min [Km^2/sec^2]
alpha_min=pi;                              %[rad]
beta_min=2*asin(sqrt((s-c)/(2*a_min))); %[rad]
P_min=(4*a_min*(s-r1)*(s-
r2)*(sin(0.5*(alpha_min+beta_min)))^2)/c^2;
e_min=sqrt(1-P_min/a_min);  %Eccentricity associated with a_min
TOF_min=((a_min^1.5)/sqrt(mu))*((alpha_min-beta_min)-
(sin(alpha_min)-sin(beta_min)));

%%%%%%%%%%%%
%Finding "a"
%%%%%%%%%%%%
TA(counter)=TA(counter)*180/pi;

if TA(counter)<180
    TOF_par=(sqrt(2/mu)*(s^1.5-(s-c)^1.5))/3;   %Type 1
else
    TOF_par=(sqrt(2/mu)*(s^1.5+(s-c)^1.5))/3;   %Type 2
end

TOF_Vector=24*3600*[TOF];
a0=a_min;

for j=1:length(TOF_Vector),
    TOFh=TOF_Vector(j);
    if TA(counter)<180
        if (TOFh<TOF_par)
            %fprintf('\n\nHyperbola-1H\n\n');
            a0h=-a0;
            FUN=inline('(sqrt(mu)*TOFh)-
(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-(2*asinh(sqrt(s/(2*abs(a))))))-
(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
            a(j)=fsolve(FUN,a0h,optimset('Display','off'),s,TOFh,c,mu);
            else
                if (TOFh<TOF_min)
                %    fprintf('\n\Ellipse-1A\n\n');
                    FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
sin(2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN2,a0,optimset('Display','off'),s,TOFh,c,mu);
                else
                %    fprintf('\n\Ellipse-1B\n\n');
                    FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-
2*asin(sqrt(s/(2*a)))-sin((2*pi)-2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-
c)/(2*a)))-sin(2*asin(sqrt((s-c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN3,a0,optimset('Display','off'),s,TOFh,c,mu);
                end
            end
        else
            if (TOFh<TOF_par)
                %fprintf('\n\nHyperbola-2H\n\n');
                a0h=-a0;
                FUN=inline('(sqrt(mu)*TOFh)-
(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
(2*asinh(sqrt(s/(2*abs(a))))))+(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-
(2*asinh(sqrt((s-c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                a(j)=fsolve(FUN,a0h,optimset('Display','off'),s,TOFh,c,mu);
            else
                if (TOFh<TOF_min)
                    %fprintf('\n\Ellipse-2A\n\n');
                    FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
sin(2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN2,a0,optimset('Display','off'),s,TOFh,c,mu);
                else
                    %fprintf('\n\Ellipse-2B\n\n');
                    FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-
2*asin(sqrt(s/(2*a)))-sin((2*pi)-2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-
c)/(2*a)))-sin(2*asin(sqrt((s-c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN3,a0,optimset('Display','off'),s,TOFh,c,mu);
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%
    %Type 1 parameteres
    %%%%%%%%%%%%%%%%%%%
    if TA(counter)<180
        if (TOFh<TOF_par)
            %fprintf('\n\nType-1H\n\n');
            alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
            betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
            P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;
%[Km]
            P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2;
%[Km]

            if P_plus>P_minus
                P=P_plus;
            else
                P=P_minus;
            end
        else
            if (TOFh<TOF_min)
                %fprintf('\n\nType-1A\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                %fprintf('\n\nType-1B\n\n')
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;
%[Km]

                if P_plus<P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            end
        end
    else
        if (TOFh<TOF_par)
            %fprintf('\n\nType-2H\n\n');
            alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
            betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
            P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;
%[Km]
            P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2;
%[Km]

            if P_plus<P_minus
                P=P_plus;
            else
                P=P_minus;
            end
        else
            if (TOFh<TOF_min)
                %fprintf('\n\nType-2A\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus<P_minus
```

```
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                %fprintf('\n\nType-2B\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;
%[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            end
        end
    end
    TA(counter)=TA(counter)*pi/180;

    %time of flight check against stk values
    tof_day(counter)=TOF;
    a_a(counter)=a;

    %%%%%%%%%%%%%%%%%%%%%%%%%
    %Transfer Characteristics
    %%%%%%%%%%%%%%%%%%%%%%%%%

    %Orbit
    r1_per=200+6378.14;
    r2_per=500+3397.00;
    e=sqrt(1-(P/a_a(counter)));
    vD=sqrt(2*((mu/r1)-(mu/(2*a_a(counter)))));
    vA=sqrt(2*((mu/r2)-(mu/(2*a_a(counter)))));
    rD=r1; rA=r2;
```

```
%delta_V for inclination change
theta_star_halfway=pi/2;
r_halfway=a_a(counter)*(1-e);
velocity_at_halfway=sqrt(2*((mu/r_halfway)-(mu/(2*a_a(counter)))));
deltav_inc_change(counter)=2*velocity_at_halfway*sin(inclination/2);

%Angle
theta_star_D=acos((1/e)*((P/r1-1)));
theta_star_A=acos((1/e)*((P/r2-1)));

if theta_star_A-theta_star_D==TA(counter)
    theta_star_A_new=theta_star_A;
    theta_star_D_new=theta_star_D;
elseif -theta_star_A-theta_star_D==TA(counter)
    theta_star_A_new=-theta_star_A;
    theta_star_D_new=theta_star_D;
elseif -theta_star_A+theta_star_D==TA(counter)
    theta_star_A_new=-theta_star_A;
    theta_star_D_new=-theta_star_D;
else
    theta_star_A_new=theta_star_A;
    theta_star_D_new=-theta_star_D;
end

theta_star_D;
theta_star_A;

gamma_D=acos(sqrt(mu*P)/(r1*vD))*sign(theta_star_D_new);
gamma_A=acos(sqrt(mu*P)/(r2*vA))*sign(theta_star_A_new);

%r, h, theta relationship
vD_vector_1=[vD*sin(gamma_D) vD*cos(gamma_D) 0];
vD_vector= vD_vector_1*trans_matrix';
v_infinity_dep=vD_vector'-v1_vector;
v_infinity_mag(counter)=norm(v_infinity_dep);
```

```
    delta_vD(counter)=sqrt(v_infinity_mag(counter)^2+((2*muE)/r1_per)) -
sqrt(muE/r1_per);
    v_infinity_D(counter)=v_infinity_mag(counter);

    vA_vector_1=[vA*sin(gamma_A) vA*cos(gamma_A) 0];
    vA_vector= vA_vector_1*trans_matrix_M';
    v_infinity_arr=v2_vector-vA_vector';
    v_infinity_mag1(counter)=norm(v_infinity_arr);
    delta_vA(counter)=sqrt(v_infinity_mag1(counter)^2+((2*muM)/r2_per)) -
sqrt(muM/r2_per);
    v_infinity_A(counter)=v_infinity_mag1(counter);

    %TOTAL

deltav_tot(counter)=abs(delta_vD(counter))+abs(delta_vA(counter))+abs(deltav_inc_c
hange(counter));

    %Burn Time Analysis

    g=9.81;             %[m/s^2]
    Isp=1007.1;            %Assumed for now
    mdot=2.13;            %based on current engine [kg/s]
    mass_f=76862.21805; %Current MHV mass [kg]

    mass_prop_A(counter)=mass_f*exp(delta_vA(counter)*10^3/(Isp*g)) - mass_f;
    time_burn_A(counter)=mass_prop_A(counter)/mdot;

    mass_prop_inc(counter)=(mass_prop_A(counter) +
mass_f)*exp(deltav_inc_change(counter)*10^3/(Isp*g)) - (mass_prop_A(counter) +
mass_f);
    time_burn_inc(counter)=mass_prop_inc(counter)/mdot;

    mass_prop_D(counter)=(mass_prop_inc(counter) + mass_prop_A(counter) +
mass_f)*exp(delta_vD(counter)*10^3/(Isp*g)) - (mass_prop_inc(counter) +
mass_prop_A(counter) + mass_f);
    time_burn_D(counter)=mass_prop_D(counter)/mdot;
```

```
mass_prop_total(counter)=mass_prop_A(counter)+mass_prop_inc(counter)+mass_prop_D(c
ounter);
%end

%best_counter=find(deltav_tot==min(deltav_tot));

out= deltav_tot;
```

10.5.1.3 Planetdata.m

```
%PLANETDATA/m
%Output the heliocentric position and velocity of Earth and Mars for a %given
Julian Date positions are in km, and velocities are in km/s, %theta_stars are in
radians the planetary data is taken from STK
%all output vectors are column vectors

function [pos_earth, vel_earth, thst_earth, pos_mars, vel_mars, thst_mars] =
planet_data(JD_interp)

if JD_interp<2451758 | JD_interp>2462867
    disp 'The JD being input to planet_data is out of range!'
    return
end

load earth_data
load mars_data

%variables loaded into workspace are:
%JD ,x_earth, y_earth, z_earth, x_vel_earth, y_vel_earth, z_vel_earth,
%x_mars, y_mars, z_mars, x_vel_mars, y_vel_mars, z_vel_mars

%store all the STK data in an array, that will be used to interpolate
pos_earth_vector = [x_earth, y_earth, z_earth];
vel_earth_vector = [x_vel_earth, y_vel_earth, z_vel_earth];
pos_mars_vector = [x_mars, y_mars, z_mars];
vel_mars_vector = [x_vel_mars, y_vel_mars, z_vel_mars];
```

```
%interpolate to find requested Julian date
pos_earth = interp1(JD, pos_earth_vector, JD_interp)';
vel_earth = interp1(JD, vel_earth_vector, JD_interp)';
pos_mars = interp1(JD, pos_mars_vector, JD_interp)';
vel_mars = interp1(JD, vel_mars_vector, JD_interp)';

a_earth = 1.49597807e8; %km
e_earth = .01670545;
p_earth = a_earth*(1-e_earth^2);
thst_earth      =      acos(1/e_earth*(p_earth/norm(pos_earth)      -      1))      *
sign(dot(pos_earth,vel_earth)) ;
%                                                              ---------- quad check
--------

a_mars = 2.27936636e8; %km
e_mars = .09341233;
p_mars = a_mars*(1-e_mars^2);
thst_mars       =       acos(1/e_mars*(p_mars/norm(pos_mars)       -       1))       *
sign(dot(pos_mars,vel_mars)) ;
%                                                          -------- quad check -----
---
```

10.6 Bibliography

- Howell, K. C. “AAE 532 Orbital Mechanics Lecture Notes”
- Howell, K. C. “AAE 340 Orbital Mechanics Lecture Notes”
- Bate, Roger R., Mueller, Donald D., White, Jerry E. “Fundamentals of Astrodynamics” Dover Publications, Inc. .New York, NY. 1971
- Longuski, J. M. “AAE 340 Dynamics and Vibrations Lecture Notes”
- Longuski, J. M. “Error Analysis for Pulsed Maneuvers of a Dual-Spin Spacecraft”
- Mars’ atmosphere: http://www.grc.nasa.gov/WWW/K-12/airplane/atmosmrm.html
- Aerobraking: http://mpfwww.jpl.nasa.gov/mgs/sci/aerobrake/SFD/SFD-Paper.html
- Aerobraking: http://mpfwww.jpl.nasa.gov/mgs/mgs-links.html

11 Feuerbron, Steve

11.1 Cryogenic Storage Appendix

Author: Steve Feuerborn

11.1.1 System Description

As presented in the main body of the report, the long-term cryogenic storage of certain fluids (primarily Hydrogen and Oxygen) poses a significant challenge to our mission design. We have presented what is hopefully a feasible solution to the problem, but would also like to identify some of the possible complications in implementing such a design at today's technology level.

11.1.2 Design Issues

Zero Boiloff (ZBO) systems are currently being proposed as the answer to any long-term space mission which requires the storage of cryogenic liquids. Unfortunately ZBO is a purely conceptual technology at this point. While the concept is sound, and fairly simple, there are a number of complications involved.

First off, no practical testing has been done, especially on the scale involved in this mission. This means that any number of unforeseen issues may arise with the practicality of the design. The only testing that the team could find was some proposed small-scale testing of spherical tanks in the hundreds of kilograms range, with a significant amount of extra mass of insulation and testing equipment, as seen in Figure 11-1. Unfortunately our mission would involve non-ideal, non-spherical tanks on a much larger scale than these tests. It is clear that the immediate testing push would not be very similar to the size of system needed for a manned mars mission.

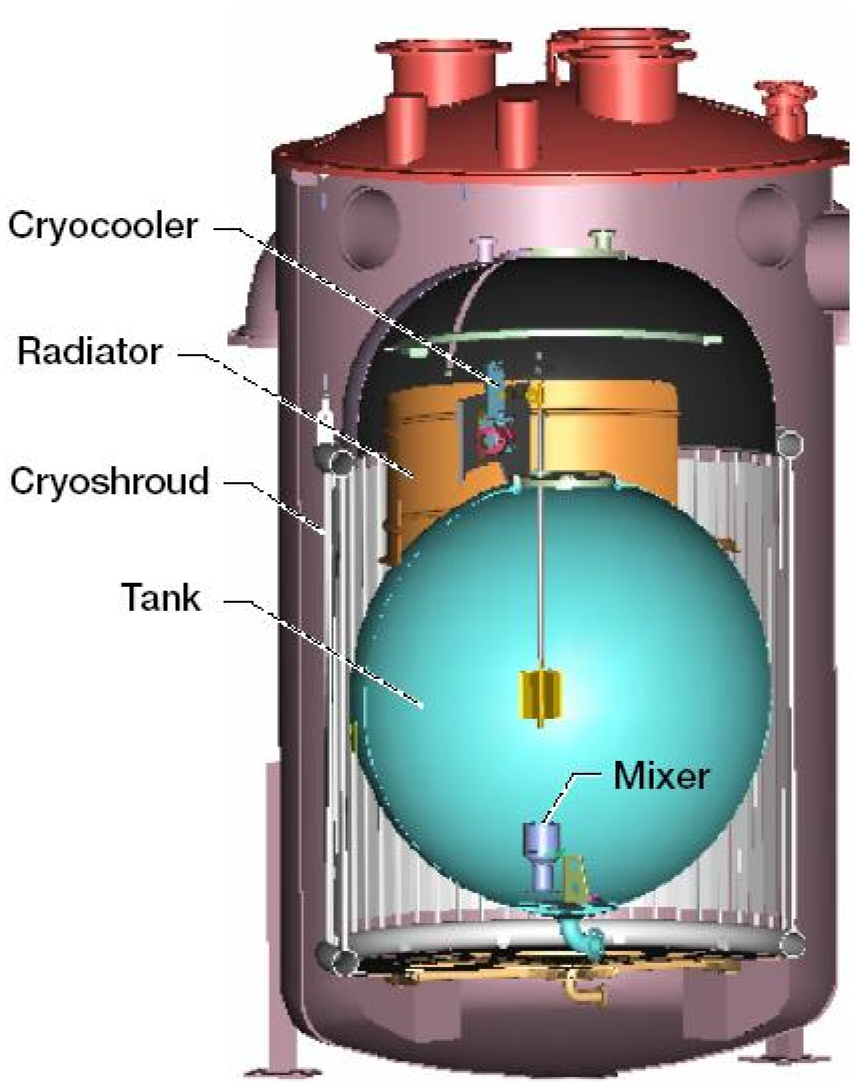


Figure 11-1: Small-Scale ZBO Test Rig [1]

Another area of concern is that of mixing. Unfortunately, fluid properties in zero-g environments are not very well characterized at the moment. While of great interest, and currently undergoing a great deal of research, the exact nature of a feasible mixing device is unknown. While current technology could probably provide some sort of working mixing device, it would be far from ideal, and may cause further unforeseen problems.

Additionally, the actual heat pump system proposed is somewhat of a black box. While low temperature heat pumps are proven technologies, the following issues exist. Zero-G rated heat pumps of the like have only been implemented on very small scales, in order to cool sensitive scientific equipment, such as deep space telescopes. While these pumps do work in their given function, they have much different design parameters than those which would be needed for the mission. Larger heat pumps have been produced for similar ranges, however, they have to date only been in terrestrial implementation. In short, portions of the heat pump design are feasible, but it is not guaranteed that trying to space-rate a large-scale heat pump would not raise further problems.

Next comes the radiators. We believe this to be a more or less insignificant portion of the problem, due to the very small amount of heat to be rejected, but it is not clear whether additional radiators would need to be designed, or if indeed the tank-mounted system described would be feasible.

Finally, all current ZBO designs assume perfect efficiency, which in theory is correct. Unfortunately, we know that such a complex, active system is never going to be 100% efficient, and since it is effectively unproven technology, there is an even greater chance that the system will have significant inefficiencies. For our purposes, we assumed that 1% of the propellant would still be lost due to inefficiencies in the mixing device, seals, seepage, etc… but unfortunately it might be significantly more when actually applied.

In summary, it should be noted that this is an area where the proposed design is relatively valid, but would require a great deal more testing and development before entrusting human lives to it.

11.1.3 Sizing Code

Below is attached the Matlab Sizing Code used to determine mass and power requirements of the ZBO system for each tank that it is applied to. The sole user input is the tank surface area, which is used to calculate the applied heat load. From there, the code determines the efficiency of powered system to use, and finds the final power and masses for both active and passive portions of the ZBO system.

This code, and associated analysis, was applied to all vehicles with cryogenic tanks in orbit, the Crew Transport Vehicle (CTV), Mars Habitat Vehicle (MHV) as well as for the rough sizing estimate for the Mars Lander Vehicle (MLV).

```
% Steve Feuerborn
% AAE 450
% ZBO Thermal Control Sizing Code
% Notes on Usage:
% This code should be accurate for the sizing of liquid hydrogen tanks
% within the space environment.  This is designed primarily for use in the
% Earth-Mars mission, with LEO solar heating rates as the design point (as
% they should be the worst case for the given mission)

% This may be used as an approximation for Liquid Oxygen, or other
% cryogenic liquid storage, but should not be taken as the final design for
% said systems

% User Input
Tank_Area=321;     % m2

% Calculation of qheat
qin=1370;          % Wt/m2 (at LEO)
insulation_factor=1.68769E-4; % Ratio of qheat/qin based empirically on given MLI configuration
q=qin*.5*Tank_Area*insulation_factor; % Wt

% Empirical Sizing Factors - derived from sources (1) and (2)
MLI_Area_Density=2.4;    % (kg/m2)
if q<=25
    Active_Mass_Ratio=30;     % (kg/Wt) (qheat)
elseif and(q>25,q<=70)
    Active_Mass_Ratio=20;     % (kg/Wt) (qheat)
else
    Active_Mass_Ratio=15;     % (kg/Wt) (qheat)
end
Cryo_Power_Ratio=0.16;  % (kWe/Wt)

% Final Values
Total_MLI_Mass=Tank_Area*MLI_Area_Density       % Total mass of MLI component (kg)
Total_Active_Mass=q*Active_Mass_Ratio           % Total mass of Active component (kg)
Total_Cryo_Power=q*Cryo_Power_Ratio             % Total power required for active component (kWe)
Total_Cryo_Mass=Total_MLI_Mass+Total_Active_Mass% Total mass of combined system (kg)

%(1) Beke et al. "Nuclear Thermal Rocket/Vehicle Design Options for Future NASA Missions to the Moon
and Mars." AIAA-1993-4170
%(2) Zubrin, Robert. "The Use of Dual Mode Nuclear Thermal Rocket Engines to Support Space Exploration
Missions." AIAA-1991-3406
%(3) Kittel, Peter. Plachta, David. "An Updated Zero Boil-Off Cryogenic Propellant Storage Analysis
Applied to Upper Stages or Depots in an LEO Environment." AIAA-2002-3589
```

References

[1] Kittel, Peter. And David Plachta. An Updated Zero-Boiloff Cryogenic Propellant Storage Analysis Applied to Upper Stages or Depots in an LEO Environment. AIAA–2002–3589.

[2] Millis, Marc G. Design Factors for Applying Cryogen Storage and Delivery Technology to Solar Thermal Propulsion. NASA-TM-107379.

11.2 Power and Propulsion Cooling System Appendix

Author: Steve Feuerborn

11.2.1 System Description

There are at least a handful of notes to take on the reactor thermal control section presented earlier in this report. In general, the design presented is a very surface level analysis, which should by no means be taken as the be-all and-all solution.

11.2.2 Possible Changes and Applications

One note to take is that the current design does not take into account the power and mechanism required to pump the potassium working fluid through the heat pipe system. In the SP-100 design literature there is some mention of electromagnetic pumps which pump both the internal an external heat pipe loops. These pumps were integrated somehow into the thermal-electric elements in some way, and it is unclear if this has been taken into account in the current improved design. The worst case is that additional power and mass needs to be added to some combination of the power and thermal sides of the reactor design to account for replacement pumps of some sort, but this should be relatively minor. It should also be noted that the operating conditions of the given system will likely limit what kind of conventional pumps can be used. Unfortunately most pump systems are designed to pump materials such as water and ammonia, not liquid metals.

Also of possible concern is the selection of working fluid. While potassium should be sufficient for the design, it may not be the optimal choice. There has also been some testing on other similar materials, in particular sodium. While sodium is usable over a similar temperature range, and has somewhat similar properties, it is unclear as to which would better serve the mission. This is largely due to the fact that the heat-transportation properties of liquid metals are mostly uncharacterized. While there has been years of practical experience using water, ammonia and other fluids in heat pipes, the applications of sodium and potassium are much more limited. Therefore, we recommend that further testing be done of said systems in order to pin down the benefits and drawbacks of the different materials.

It should also be noted that the analysis presented for the power reactor cooling is used for all vehicles which possess the same basic reactor. While each instance of the reactor has a different power output, the same basic analysis holds, with a scaling factor based on its power capacity. The vehicles using this reactor cooling system are the Crew Transport Vehicle (CTV), Mars Habitat Vehicle (MHV), Mars Orbiting Relay Satellites (MORS) and Heliocentric Relay Satellite (HRS). A final note is that the MHV version of the cooling system is sized to the space radiative environment, which means that it is more than sufficient for use on the surface, where the cooling will be greater due to convective heat transfer.

11.2.3 References

[1] El-Genk, Mohamed, and Jean-Michel Tournier. Conceptual Design of a 100-kWe Space Nuclear Reactor Power System with High-Power AMTEC. Albuquerque: University of New Mexico, 2004.

[2] Krotiulk, William J. Thermal Hydraulics for Space Power, Propulsion, and Thermal Management System Design. Washington: AIAA, 1990.

12 Friel, Jason

12.1 Appendix: Design of the MHV Ballute

Author: Jason Friel

12.1.1 System Description

One of the major problems encountered in designing the MHV system is being able to slow it down to the point that it can successfully land. Originally we thought this would be accomplished by aerobraking around Mars, significantly reducing our orbital velocity. To accomplish aerobraking within a reasonable span of time we decided to attach a ballute (balloon-parachute) to it, which greatly increases the amount of drag acting on the vehicle by increasing the cross-sectional area. Problems were also encountered during the planetary entry portion however, in that Mars' thin atmosphere was not creating enough drag on the MHV to stop it from hitting the ground while still traveling at above 1000 m/s. Therefore we decided to use the ballute during the descent to Mars' surface as well as during aerobraking. To save mass we also decided to use the same ballute for each maneuver.

12.1.2 Theory and Assumptions

The concept of a ballute has been around since the 1960s when Goodyear researched a similar device, and there has been much research of late into using ballutes for maneuvers such as aerocapture and aerocapture. Angus McRonald at the Jet Propulsion Laboratory has also investigated the feasibility of using them as a drag producing device for planetary entry [3]. Both of these concepts are very important to our mission, so the ballute is a critical technology for the MHV mission to be successful. Ball Aerospace [2] and several other companies have been contracted to do ballute research for aerocapture purposes, and it is viewed as a vital technology for future space missions. Therefore we feel that it an acceptable assumption to state that we will be able to use a ballute.

The theory behind a ballute is that by increasing the cross-sectional area (S) that is affecting the vehicle, one can increase the amount of drag the vehicle generates. This is useful for craft with small values of S that don't produce much drag by themselves; an example is the Mars Global Surveyor that used a ballute to reduce the amount of time necessary for it to complete aerobraking. A ballute is also useful for situations where the air density is low, such as Mars. A low air density reduces the amount of drag generated as can be deduced from

$$D = \frac{1}{2}\rho V^2 S C_D$$ **12-1 Drag Equation**

This reduced effectiveness of drag can be compensated for by increasing S by attaching a ballute to the vehicle.

The increase in drag provided by a ballute consequently reduces the heating experienced by a vehicle, because the vehicle will be traveling at a slower speed. This is desirable because it will reduce the size of the heat shield required for planetary entry. One could simply increase the size of the vehicle to the required cross-sectional area, but this would result in a huge and very heavy spacecraft. A ballute weighs significantly less and is easily storable, all it requires is a tank of a chosen gas to inflate it.

A critical assumption we made in using a ballute is that its material will be able to withstand the heating encountered during Martian entry, and that it won't structurally fail. Current ballute designs use the Kapton thin-films which theoretically could take the heat loads. Doing a proper heating analysis is beyond the scope of this class however, so we will just assume that it is possible. We also assume that the techniques necessary to manufacture a ballute of the size we require will be available during the project timeframe.

12.1.3 Analysis

Ballute designs are generally either toroidal or spherical in nature, and are clamped or towed behind the vehicle. For the MHV, we decided the slightly higher drag coefficient and the ability to create choked flow through the hole of the toroid makes it the better choice. We also decided to tow it behind the MHV as it states in [1] that there has not been sufficient research on the possible instabilities caused by clamping a ballute to the rear of the craft. This means the ballute has a C_D of approximately 1.4 and will be towed some length behind the MHV. This length would be determined by the proper length for all the turbulent flow coming off of the MHV to pass through the center of the toroid and create choked flow, which would be found through computational fluid dynamics or wind tunnel testing.

Figure 12-1 A spacecraft with a trailing toroidal ballute. Picture taken from Ball Aerospace

Using the ballute for planetary entry as well as aerobraking greatly increased its size. Originally the ballute had an outer diameter of 28 meters and an inner diameter of 7 meters. After recognizing the need to use the ballute to generate drag during entry as well, the size increased considerably. It now has an outer diameter of 120 meters and an inner diameter of 100 meters, and has a width of 10 meters. This provides us with a cross-sectional area of 3456 m^2. This huge increase is necessary to provide enough drag so that the MHV can successfully deploy parachutes and perform a final powered descent.

The material used for the ballute is Kapton polyimide film, a thin film that can withstand high temperatures. Using a thickness of 0.7 mm, we calculated the mass of the ballute as 3797 kg with a storable volume of 0.6 m2. In order to successfully

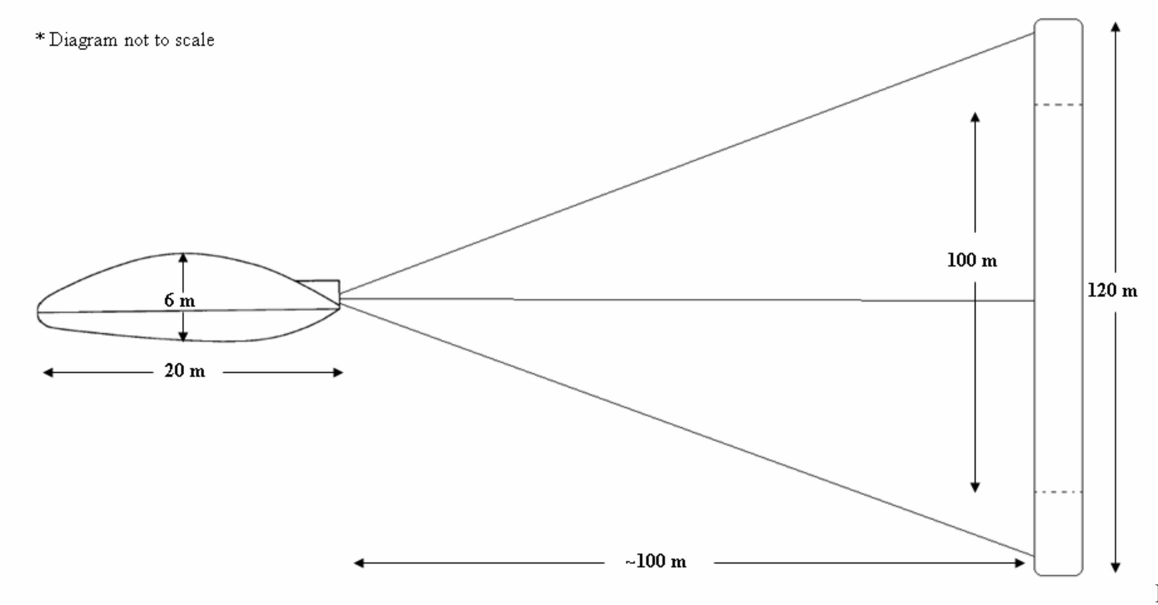


Figure 12-2 MHV Aeroshell with toroidal ballute attached

deploy, the ballute will need to be inflated with helium. This requires that a helium tank be taken along and also a pump. We estimated the pump mass as 100 kg after researching industrial air pumps; this is probably a conservative guess. 6175 kg of helium are needed to fill the internal volume of the ballute; we decided that a 1 cm thick aluminum tank that is 9 m long with a diameter of 4 m would be the correct size, which means the internal pressure is 35 kPa. This is a much lower pressure than industrial helium tanks seem to be able to withstand, which means the tank size could go down appreciably. We decided it would be wiser to go with a more conservative estimate in this case.

12.1.4 Code

Hetank.m – code used to compute what the pressure of the helium tank would be at a given volume.

```
% Jason Friel
% 3/25/05
% AAE 450
% hetank - program to calculate the pressure exerted by helium on the tank
% it will be in

clear
close all
clc

m = 6175; % kg, mass of helium in tank
R = 8.314; % universal gas constant
T = 280; %K, temp
M = 4.003; % kg/kmol, molecular weight of helium
V = 0.1:1:1240; % m^3, range of volumes being considered for tank

p = m*R*T./(M*V); % Pa, pressure that the helium will be at

plot(V,p)
title('Pressure of Helium as a function of tank volume at T = 280 K, m = 6175 kg')
xlabel('volume (m^3)')
ylabel('pressure (Pa)')
```

12.1.5 References

[1] Gnoffo, Peter & Anderson, Brian. “Computational Analysis of Towed Ballute Interactions”. Paper AIAA 2002-2997, 2002. **http://techreports.larc.nasa.gov/ltrs/PDF/2002/aiaa/NASA-aiaa-2002-2997.pdf**

[2] Miller, K., Gulick, D., Lewis, J., Trochman, B., Stein, J., Lyons, D., Wilmoth, R. “Trailing Ballute Aerocapture: Concept and Feasibility Assessment”. AIAA 2003-4655, 2003.
http://www.inspacepropulsion.com/tech/pubs/2003_4655_Trailing%20Ballute%20Aerocapture.pdf

[3] McRonald, Angus. “A Light Weight Inflatable Hypersonic Drag Device for Planetary Entry”. Atmospheric Reentry Vehicles and Systems, 1999

http://techreports.jpl.nasa.gov/1999/99-0422.pdf

12.2 Appendix: Aerodynamic Loading of the ELV

12.2.1 Topic Description

During it's ascent to low Earth orbit, the ELV will experience structural loading from several sources. One of these sources is the aerodynamic forces acting on the vehicle during its atmospheric flight. In order to properly design the structure of the ELV, the loading caused by these aerodynamic loads must be known.

12.2.2 Theory and Assumptions

The main aerodynamic forces acting on the ELV during its ascent are the lift and drag it produces. Due to the fact that ELV is a symmetric body and will be flying straight up during the portion of its flight in the lower atmosphere, we will assume that it produces zero lift. Therefore the only force to consider is the drag acting on the ELV.

$$D = \frac{1}{2}\rho V^2 S C_D$$ **12-2 Drag Equation**

$$L = \frac{1}{2}\rho V^2 S C_L = 0$$ **12-3 Lift Equation**

Finding the largest drag force acting on the ELV will give the worst case scenario for the structures team to design for. In order to do this, the velocity history of the ELV must be known. After being provided with the trajectory code for the ELV by Caley Burke, we were able to generate the velocity profile. Now that the velocity history is known, we wrote a Matlab script to calculate the drag acting on the ELV. As can be seen in the following plot, drag is only significant at low altitudes. Air density decreases with altitude to the point that after an altitude of 35 km drag is insignificant. The ELV gains most of its velocity at higher altitudes where the increase in velocity is less relevant than the decrease in air density. Therefore the max drag and hence max aerodynamic loading occurs shortly after take-off.

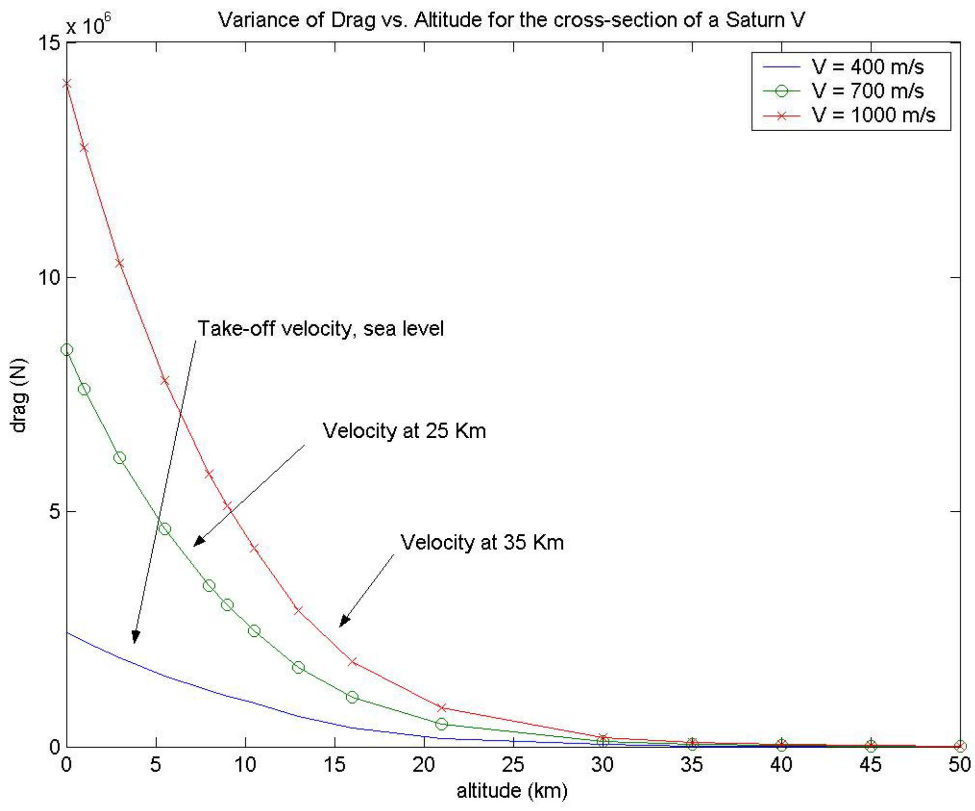


Figure 12-3

Using the given dimensions of the ELV results in the max stress experienced by the ELV due to aerodynamic loading is 70.723 kPa. This is a small number compared to the other structural loading the ELV will encounter, which means the aerodynamic loads are not a major factor. This will be true as long as the ELV travels at slow velocities in the lower part of the atmosphere.

12.3 Code

Dragforce.m – calculates the drag force experienced by a typical rocket shaped vehicle traveling through the Earth's atmosphere. The code to evaluate the coefficient of drag was provided by Phil Spindler

```
% Jason Friel
% AAE 450
% Drag force evaluation on a Saturn V

clear
clc

S = 0.25*pi*(6.604^2); % m^2.  taken from www.apollosaturn.com
V1 = 400; % m/s^2
```

```
V2 = 700;
V3 = 1000; % at 67 km

h = [0 1 3 5.5 8 9 10.5 13 16 21 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100];
rho = 1.225*[1 0.9075 0.7423 0.5694 0.4292 0.3813 0.3172 0.2176 0.1359 0.06181...
         0.01458 6.705e-3 3.144e-3 1.536e-3 7.980e-4 4.381e-4 2.354e-4 1.219e-4...
         6.061e-5 2.846e-5 1.282e-5 6.710e-6 2.789e-6 1.137e-6 4.575e-7];
a = 340.294*[1 9.887e-1 9.656e-1 9.359e-1 9.053e-1 8.927e-1 8.736e-1 8.671e-1 8.671e-
1...
         8.691e-1 8.869e-1 9.070e-1 9.354e-1 9.591e-1 9.692e-1 9.489e-1 9.229e-1...
         8.962e-1 8.687e-1 8.469e-1 8.261e-1 8.261e-1 8.261e-1 8.261e-1 8.261e-1];
M1 = V1./a;
M2 = V2./a;
M3 = V3./a;

Cd1 = dragRocket(M1);
Cd2 = dragRocket(M2);
Cd3 = dragRocket(M3);

D1 = 0.5.*rho.*(V1.^2).*Cd1.*S;
D2 = 0.5.*rho.*(V2.^2).*Cd2.*S;
D3 = 0.5.*rho.*(V3.^2).*Cd3.*S;

plot(h,D1,h,D2,'-o',h,D3,'-x');
title('Variance of Drag vs. Altitude for the cross-section of a Saturn V')
xlabel('altitude (km)')
ylabel('drag (N)')
legend('V = 400 m/s','V = 900 m/s','V = 1000 m/s')
```

Dragrocket.m – calculates the C_D of a rocket based on the Mach number it is traveling at.

```
function Cd=dragRocket(M)
% calculate the drag of a rocket based on frontal area
% Given by Phil Spindler

    if M <= .8
         Cd = .4;
    elseif M <= 1.5
         Cd = .8570.*M - .2857;
    elseif  M > 1.5
        Cd = .55 + .45.*exp( - .9.*(M - 1.5));
    end
```

13 Golbov, Conrad

13.1.1.1 Appendix - Solar Flare Detection

Author(s): Conrad Golbov

In contrast to past manned missions, Project Legend involves long periods of time spent outside the Van Allen radiation belts. One of the dangers that face our astronauts in deep space is the event of a solar flare. We recognize the danger, and as a result have included the “safe room” radiation-shielded areas which will minimize this threat. However, solar flares are unpredictable, and though certain techniques can help us to be alert, only constant vigilance will allow our astronauts to safely complete this mission. This section’s purpose is to document and declare these dangers and their solutions.

The sun on whole goes through an 11-year cycle regarding the frequency of sunspots, the darker, cooler which appear and from which solar flares are thought to arise. The peak frequencies of this cycle fall on 2000, 2011, 2022, and so on. While GCR radiation does increase somewhat during the “troughs” of this cycle, solar flare frequency is down. Naturally, it is not all solar flares which our crew needs to worry about. Often, there are many solar flares per day ranging from the smaller C-class, medium M-class, and the heavy X-class flares. It has been estimated that an astronaut exposed to an X-class flare in only a spacesuit might absorb as much as 50 rem of radiation instantly.[1] This would certainly bring on radiation sickness within a day or so, requiring immediate medical attention. Therefore the X-class rays are most dangerous to our mission. As shown in Figure 1-1, the frequency of these can look quite disturbing without some discerning factors.

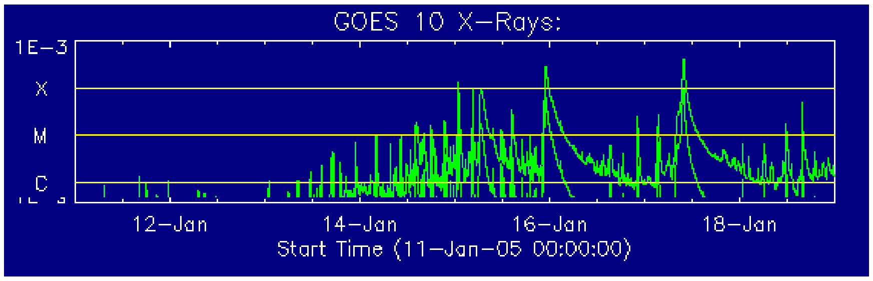


Figure 1-1. Solar flares during mid-January, 2005. [2]

The data in Figure 1-1 looks discouraging until we understand that only the solar flares *in the crew's direction* are necessary to detect. The flares shown above occur all across the sun's surface. However, to detect the flares heading toward our Crew Transport Vehicle (CTV) or the Mars Habitat Vehicle (MHV), we simply cannot rely on warning from Earth-based detectors. The time delay as we travel further and further from earth makes Earth-based warning too late, especially when Mars and Earth are on opposite sides of the sun. Therefore, we place solar flare detectors on both the CTV and the MHV.

The "anatomy" of a solar flare is important to our ability to detect them before they strike. There exists a precursor wave of rising (non-lethal) x-ray and gamma-ray radiation which occurs shortly before a larger spike of radiation hits. The crew *must* be inside the shielded room inside their vehicle when this hits to prevent taking a dangerous dose of radiation. Figure 1-2 shows this precursor wave.

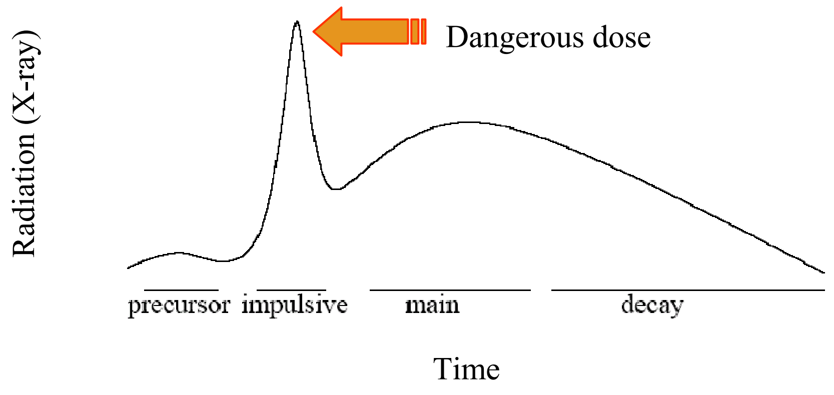


Figure 1-2 Solar flare anatomy, qualitatively. [3]

There exists a satellite in orbit about Earth called HESSI (High Energy Solar Spectroscopic Imager). This satellite tracks sunspot activity on the surface of the sun and records the solar flare radiation data around Earth. We believe that with very little modification, such a system would serve as a quality solar flare detection system. Since the likelihood of solar flares in the crew's direction increases with the number of sunspots on the sun's surface normal to the crew, tracking these sunspots would allow us to give a qualitative "Sunspot Likelihood Index" to the crew each day, giving them an idea as to how far from the habitat they might want to be that day.

During the detection phase, the modified HESSI detector would need to sound an alarm shortly after a rise in the x-rays in a manner suggesting a high M-class flare or above. Research into this area is currently underway by the HESSI team, and more will be known about the pre-flare characteristics of solar flares within the next few years. On hearing such an alarm, the crew would need to get to the shielded room *immediately*, and would have 1-5 minutes between precursor and the radiation spike (See Figure 1-2).

The modified HESSI detectors here would need to be installed on both vehicles that the astronauts will inhabit for any considerable amount of time. While it would serve the purpose to put such a detector on the Mars Orbiting Relay Satellites, this system will likely be costly, and the added warning time would be negligible. Furthermore, while the habitat is facing away from the sun, any solar flares racing toward the crew would be absorbed by Mars' surface. Therefore, though it makes research a bit more cumbersome, we recommend that night vision be a standard feature on the astronauts' spacesuits, as the only time they can safely explore away from the habitat is at night.

The mass and volume cost of the modified HESSI detectors is surprisingly small. HESSI's specifications call for no more than 120 kg for the entire original satellite. As our detector will be housed within our vehicle, we can likely shave this mass down to around 45 kg, preserving the core computing hardware and the detection arrays. However, as this detector seems at this time to be mission critical, we choose a completely redundant backup detector in case of failure of the first. Thus, the CTV and MHV will each carry around 90 kg of solar flare detection equipment, not including the mass of the safe room. This mass can be encapsulated within about 1 m^3 of rectangular space, and may be placed on the exterior of the vehicle, provided it be protected from atmospheric entry.

References:

[1] Phillips, Dr. Tony. *Moon Solar Flares*. http://www.firstscience.com/SITE/ARTICLES/moonsolarflares.asp , 2005.

[2] Freeland, Samuel. *GOES Data Event Browser*. http://www.lmsal.com/SXT/show_gev.html, 2005.

[3] Shibasaki, Kiyoto. *Signature of Energy Release and Particle Acceleration Observed by the Nobeyama Radioheliograph.* http://solar.nro.nao.ac.jp/user/shibasak/CESRA2001/CESRA2001.pdf

[4] Abramenko, et al. *Signature of Avalanche in Solar Flares as Measured by Photospheric Magnetic Fields.* http://www.aas.org/publications/baas/v35n3/spd2003/24.htm, 2003.

13.1.1.2 Appendix – Battery Calculations

Author(s): Conrad Golbov

The purpose of this section is to document the calculation of battery masses and volumes for the Earth Launch Vehicle (ELV) and the Ascent & Return Vehicle (ARV). The following spreadsheets are used. References to sources are included in the tables where appropriate.

Hercules - Earth Launch Vehicle (ELV)
Battery Power Requirements
Major Use - Primary Power
Updated - 2/16/05

Legend	***Color***
Static Values	⬜
Input Values	🟦
Totals	🟨

Baseline Power Needs (kW)		
Human Factors	0	
D&C	0.1	
Communications	0.15	
Propulsion	0.1	
Thermal	0.05	
Yet Unforseen	0.5	
Total Required	**0.9**	**kW**

Powered Time (hours)		
Pre-launch	0.1	
Launch & Ascent	0.1	
Payload Delivery	0.5	
Yet Unforseen Issues	0.3	
Total Time	**1**	**hours**

Type	**Spec. En. (Wh/kg)**	**En. Density (Wh/L)**	**Required Mass (kg)**	**Required Volume (m^3)**
NiCd	35	45	25.71	0.020
NiH2	55	32	16.36	0.028
NiMH	60	86	15.00	0.010
Li-Ion	130	160	6.92	0.006
LISO2	170	350	**5.29**	**0.003**
NaS	132	165	6.82	0.005

Note: Battery Data Taken From Table 20-10 from *Human Spaceflight* (Larson, Pranke)

System Totals		
Battery Units	**2**	(redundant systems)
Total Mass	**10.59**	kg
Total Volume	**0.005**	m^3

Note: LiSO2 source:
http://www.eaglepicher.com/NR/rdonlyres/1BA28676-3FB4-4EDD-86D9-8960F04312D1/0/LiSO2.pdf

Phoenix - Ascent & Return Vehicle (ARV)
Battery Power Requirements
Major Use - Primary Power
Updated - 2/16/05

Legend	***Color***
Static Values	⬜
Input Values	🟦
Totals	🟨

Baseline Power Needs (kW)		
Human Factors	1.1	(i.e. navigation, attitude control, telemetry downlink, computers, etc)
D&C	0.1	
Communications	0.9	
Propulsion	0	
Thermal	0.1	
Yet Unforseen	0.1	
Total Required	**2.3**	**kW**

Powered Time (hours)		
Ascent & Return (each)	30	
Rechargeable Efficiency	0.75	
Effective Total Time	**40**	**hours**

Note: Batteries to be recharged for Return by CTV while in-transit

Type	**Spec. En. (Wh/kg)**	**En. Density (Wh/L)**	**Required Mass (kg)**	**Required Volume (m^3)**
NiCd	35	45	2628.57	2.04
NiH2	55	32	1672.73	2.88
NiMH	60	86	1533.33	1.07
Li-Ion	130	160	707.69	0.58
LISO2	170	350	**541.18**	**0.26**
NaS	132	165	696.97	0.56

Note: Battery Data Taken From Table 20-10 from *Human Spaceflight* (Larson, Pranke)

System Totals		
Battery Units	**2**	(redundant systems)
Total Mass	**1082.35**	kg
Total Volume	**0.53**	m^3

14 Gramm, Paul

14.1 Oxygen and water production

Author: Paul Gramm

Make use of the process of electrolysis

$$2H_2O+\text{Energy} \rightarrow 2H_2+O_2+\text{Heat} \quad (14\text{-}1)$$

Elektron Machine (Russian Made)

- TRL 9, 150kg, ~860-1000 Watts
- TRL stands for "Test Readiness Level"
- Lifetime only about 3 years

The Elektron machine was the first large scale electrolysis system to be used successfully in a space environment. It was designed in the early 1980's and used on MIR in 1987, a newer version of this device is used as a basis for our model and it is described below.

SPE ® Oxygen Generation Assembly

- 4-20.4 lbs or 1.81-9.25 kg a day
- Lighter Weight, smaller
- Not as well tested TRL ~6+

This system designed by Hamilton Sundstrand has undergone extensive testing on Earth but has not yet been used extensively in space. Within the timeframe of our mission it is expected that this particular design will reach a TRL of 9 and possibly even undergo improvements.

14.2 Sabatier Reactor

Author: Paul Gramm

Converts:

$$CO_2 + 4H_2 \rightarrow CH_4 + 2H_2O + \text{Heat} \qquad (14\text{-}2)$$

Makes Usable Water and Unusable methane

We determined after exploring many options regarding the use of methane that the best option would be to vent it into the Martian Atmosphere. The primary idea researched was to use the methane as a combustive fuel to run a small electric generator to provide an additional power source for perhaps missions outside the habitat. Unfortunately the combustion process produces two negative side effects for our mission. These side effects include the use of oxygen in the chemical reaction with methane, and the production of carbon as a side product which would be dirty and useless for any of our purposes. Furthermore, towards the end of the design process it was determined that an excess of power was already available on the MHV so an additional generator would serve no practical purpose other than for use outside the MHV or as an emergency back up. It turns out that batteries better serve any emergency situation, and that there would be no need for a generator for the short term rock collection expeditions that will take place. The other option was to bottle the methane for storage but there is no real practical use for this. Finally, the generator using the methane has to be small due to the small amount of methane produced from the Sabatier reactor.

$M_s = 0.008*M_f0.4556$ [8]

- Where M_s is the mass of the Sabatier Reactor

M_f is the mass of fuel produced (methane and water)

Mass of condenser ~3kg

The condenser mass is based on historical data

14.2.1 Mass of Heat Exchanger

$M_{he}=0.0036*M_f - 0.0916$

- With M_{he} as the mass of the heat exchanger
- Once again M_f is the mass of the fuel produced (CH_4)

14.2.2 Volume of the Reactor

$Vs=3.6*m_{co}(\omega_o p_{co} p_{ni} SA)-1$

Vs is the volume of the reactor

- M_{co} is the mass flow rate of carbon dioxide
- P_{co} is the density of carbon dioxide
- P_{ni} is the density of the nickel catalyst
- SA is the surface area of the nickel catalyst

After the volume is known the cylinder size can be found using

- $r_s = (V_s/(2*\pi))(1/3)$ Radius
- $L_s = 0.05 + V_s/(\pi*r_s2)$ Length

The size of the Sabatier reactor itself is quite small because all it essentially consists of is a cylinder that contains a nickel catalyst to promote the reaction desired. For our purposes it was assumed that the reaction carries out to 100% completion.

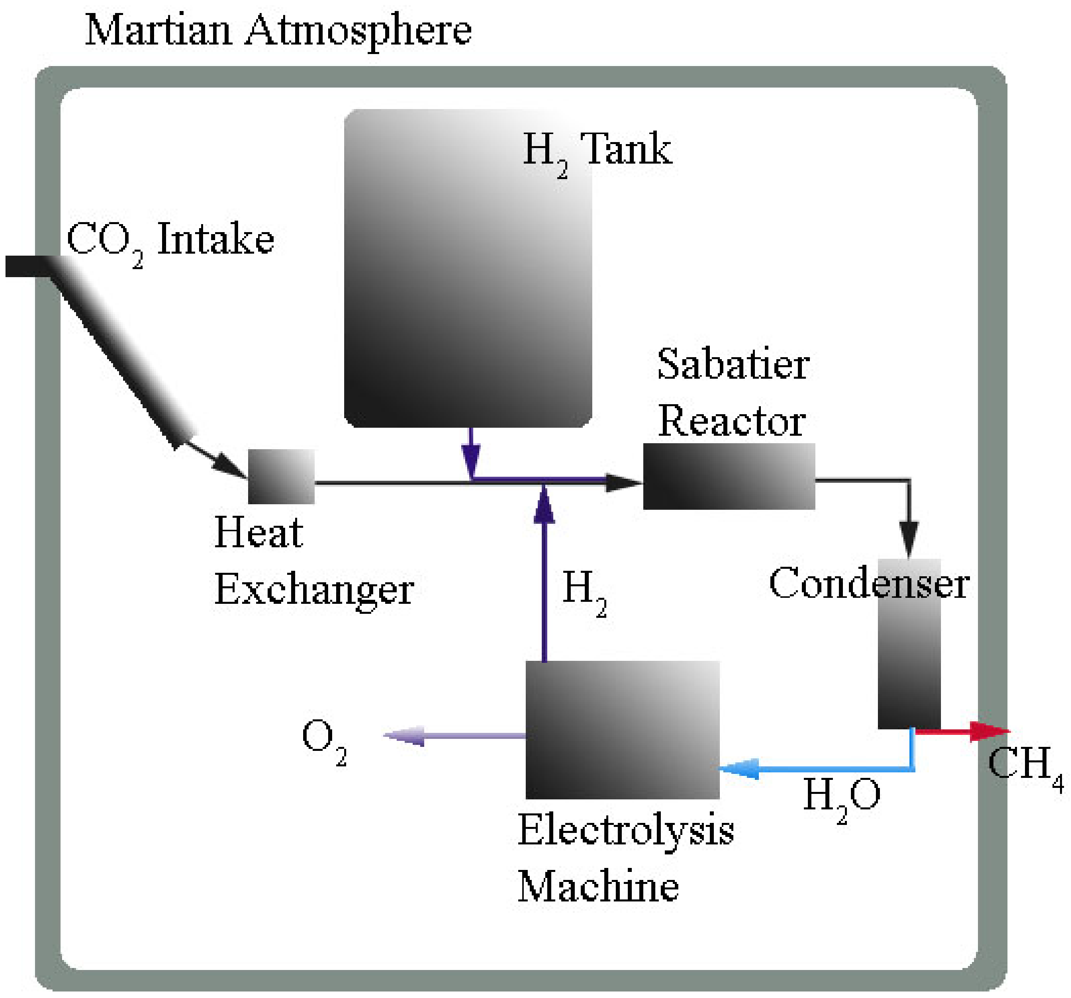


Figure 14-1 The above is the same flow chart seen in the MHV atmospheric supply portion of the report. It is a flow chart of how the conversion process takes place.

The system below is simply a three dimensional representation of the system above. Once again it should be noted that this is primarily an artist's rendition of what the system could look like, and this exact configuration was not used in our MHV. The largest change is the existence of the external Hydrogen tank traveling through a longer pipe.

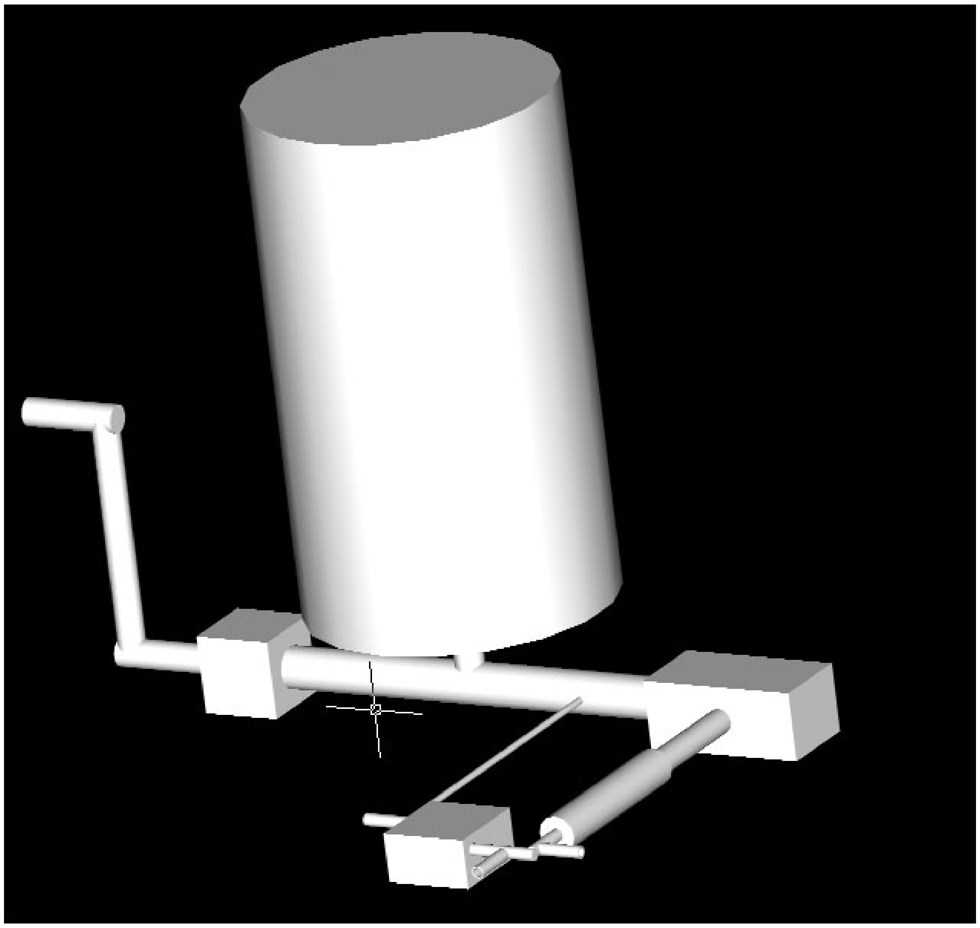


Figure 14-2 The cylinder design was abandoned during the internal configuration stage of the MHV to better make use of the internal volume on the bottom floor. It was determined that the tank could be placed externally and it was formed into the shape of a toroid.

14.2.3 Sizing the Heat Exchanger

Solve the heat transfer with $q_{tot} = m_{av}c_p(Ts – Te)$

$L_{he} = q_{tot} / (UA\pi D\Delta T_{ln})$

16g of methane is produced with 38g of water

Based on this linear relationship we can determine how much methane will be produced based on how much water we need. This allows us to size the Sabatier reactor as closely as possible

Assuming we want a maximum output of 10kg of water per day

$10kg\ H_20 / (2.25kg\ H_20/kg\ CH_4) = 4.44kg\ CH_4$

$M_f = 365*4.44kg = 1620.6kg / year$

$M_s = 0.008*1620.60.4556 = 0.232\ kg$

M_{he}=0.0036*1620.6 - 0.0916 = 5.74 kg

14.2.4 Mass of hydrogen tank

About 15.4% of the reactants are hydrogen, about half of the hydrogen is reintroduced back into the system assuming that we are making oxygen and not water

14.44kg prod/day*0.154 = 2.22kg of H/day * 365days * 2years = 1621.7 kg of H

The hydrogen amount was reduced dramatically when the capabilities of the system were changed from producing 10 kg of water down to 4kg of water.

Using a safety factor of 1.2, this means that we would need to bring about 9.23% of the needed hydrogen which is 973kg or ~1000kg of Hydrogen in storage to perform this process for 2 years.

The above analysis was performed prior to the development of a code by our structures group on cryogenic tank mass, that code was used to estimate 347kg.

Table 14-1 The above table describes a mass breakdown of the Sabatier / Electrolysis system, it does not include an allocation of 110kg for spare part.

Component	Mass
Sabatier	**0.23 kg**
Hydrogen	**540.96 kg**
Hydrogen Tank	**347 kg**
Condenser	**3 kg**
Heat Exchanger	**5.74 kg**
Electrolysis Machine	**15 kg**
Piping and Valves	**~ 5 kg**
Total	**916.93 kg**

For every 1kg of water that goes through electrolysis, about 25L of Oxygen are created (which is approximately what 1 person uses a day)

Therefore about 4kg of H2O a day could be enough for the whole crew

Many of the above equations were altered by hand as the design went through the finalization process, resulting in the final values that are displayed in Table 1

14.3 Caloric Intake Requirements

Author: Paul Gramm

Minimum requirements on Earth

For Males and Females

Mc = 66 + (13.7*W) + (5*H) + (6.8*A)

Fc = 655 + (9.6*W) + (1.7*H) + (4.7*A)

- W is weight in kilograms
- H is height in centimeters
- A is age in years

These equations produce under estimates for our mission and instead of being used; advice by Dr. Katherine Mauer was instead upheld. We used an average caloric intake of approximately 3500 per day per crew member.

14.4 Human Material Needs

Author: Paul Gramm

Contributor: Tim Szamborski

14.4.1 Medical Equipment

Mass: 6 people require ~500kg per 500 day mission so for the CTV [10]

500kg/(6people-1day)*4people*1100days = 733kg

500kg/(6p-1d)*4p*500d= 333kg for MHV

Volume: 2.5m3/500kg*733kg=3.7m^3 for CTV

2.5m3/500kg*333kg=1.7m^3 for MH

14.4.2 Bunks

10kg/person*4p=40kg

14.4.3 Exercise Equipment

145kg/6crew*4crew=96.7kg

14.4.4 Clothing

Assume Flight Suits last through 90 days of wear and tear, and that they weigh approximately 1kg a piece. Assume boxer shorts and feminine support undergarments weight 4oz apiece, and each male crew member receives 30 pairs of boxers and each female receives 30 pairs of both undergarments.

CTV: 1100days/90days = 12.2 ~ 12 flight suits per member

MHV: 500days/90days = ~6 flight suits per member

The clothing number was eventually halved on the MHV based on the mission change from 500 to a 250 day surface stay.

CTV:

12suits*1kg*4crew+30boxers*4crew*(1/9)kg+30bras*2crew*(1/9)kg+30socks*3oz*4crew+6shoes*8oz*4crew=83.4kg

MHV:

6suits*1kg*4crew+30boxers*4crew*(1/9)kg+30bras*2crew*(1/9)kg+15socks*3oz*4crew+4shoes*8oz*4crew=56.7kg

14.4.5 Laundry Water

Washer takes 51.3kg per load of water, for a loss of 5.13kg per load that can not be recovered

Each load of laundry can hold 41 pairs of underwear, 4 flight suits

The laundry load capability is made under the assumption that the mass of the clothes is more of a determining factor than the volume they take up.

3 loads/month for underwear

1.5 loads/month for bras

10 loads/month for flight suits

5.13kg*15load/month=77kg water lost per month from laundry

36.7 months * 77kg = 2823.33kg water for CTV

16.4 months * 77kg = 1262.8kg water for the MHV

The above calculations are not necessarily what were used in the final design but the process remains the same.

14.4.6 Cleaning wipes

Cleaning wipes will replace the need for showers for the crew and will be used for various other activities in the spacecraft.

6 days of wipes weighs 0.2kg for 4 crew members

0.2/6 = 0.033kg/day * 1100days = 36.7kg for CTV

0.033kg/day * 500 days = 16.5kg for MHV

0.0067m3/kg * 36.7kg = 0.245m^3

0.0067m3/kg * 16.5kg = 0.111m^3

14.4.7 Kitchen Utensils

0.5kg/person * 4 people = 2kg for the CTV and for the MHV

14.5 LED power usage

Author: Paul Gramm

80-90% more efficient of incandescent bulbs [3]

We assume conservatively that efficiency is 85%.

2.1kw m^2 * 0.15 = 0.385kw m^2 * 80m^2 = 16.8kw

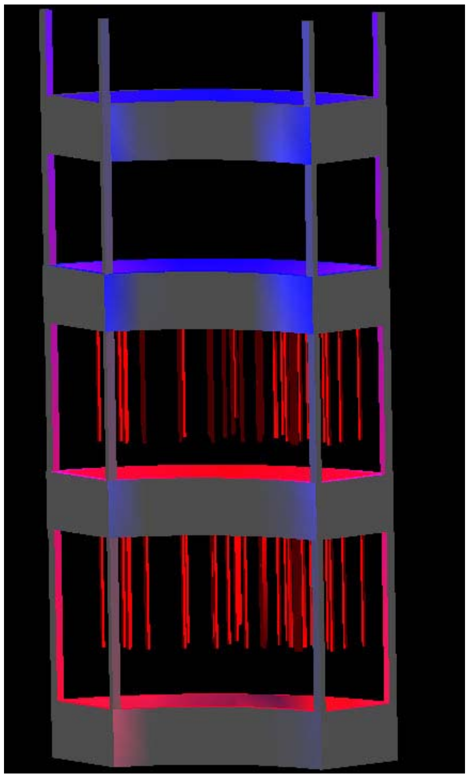


Figure 14-3 The hydroponics trays are arranged in a four level fashion in order to maximize the square area of plants per volumetric meter of inner volume. This picture depicts the configuration of LEDs as if vegetables are grown on the top two layers and wheat is grown on the bottom two. This tray shelf was designed around an early version of a MHV floor lay out (see Fig. 1-6) which has changed slightly in the final design. One could imagine the above figure extended (arc wise) to fit the configuration of the "green" colored plant trays visible in many other portions of this report and appendix.

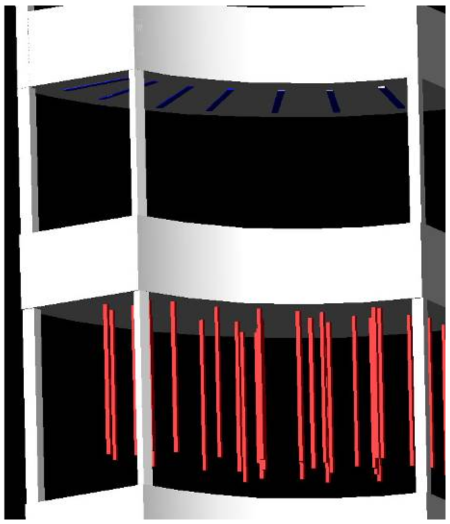


Figure 14-4 This figure shows a close up view of both the red vertical and blue horizontal LEDs.

14.6 Hydroponics

7g/L is the density of solid nutrients per liter of water [2]

The amount of nutrients required by the plants can not exceed the amount of material the bio reactor is able to produce on a daily basis. Material to be reused includes human feces (estimated to be 0.6kg per person per day in dry mass) and food products that are inedible (such as lettuce roots). To hold all of the nutrients it is necessary to design a tank.

14.6.1 Nutrient tank:

The nutrient tank is based on holding an initial value of 64kg worth of nutrients and then a need of 1.28 kg/day after that before we can produce enough inedible biomass to self-sustain the nutrient tank

64kg + 1.28kg * 80 days = 166.4kg

Based on nutrient densities this requires about a 1168.8 L tank which takes up a volume of 1.2m^3

14.6.2 Structure

Trays ~ 78lbs per 1144 inches squared with 6 inch tall walls

Adjust for walls… 1 - 2600/3744 = 0.306 + 0.1 = 0.406

124000/1444 = 108.39*78 = 8454.54lbs = 3834kg * 0.406 = 1560kg

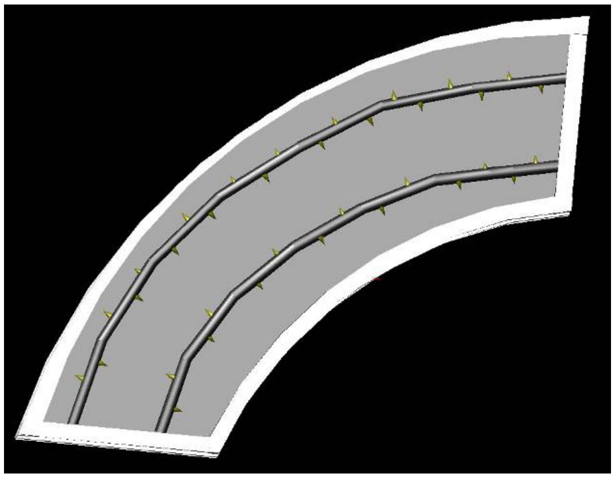


Figure 14-5 This is a top down view of one of the trays, each tray has two nutrient feeding pipes with multiple spigots to promote an even flow and distribution of nutrients throughout all of the plant roots. There will be a thin growth film placed above the water level in these trays to hold the seeds in place and promote plant growth.

The hydroponics system mass was estimated by comparing it to existing systems used on Earth and then assuming that it would be made out of a material with similar characteristics to aluminum. The system was also initially capable of handling 80 square meters of plants and was since reduced to only 60 square meters.

14.6.3 Layout

Author: Paul Gramm

Contributors: Entire MHV group

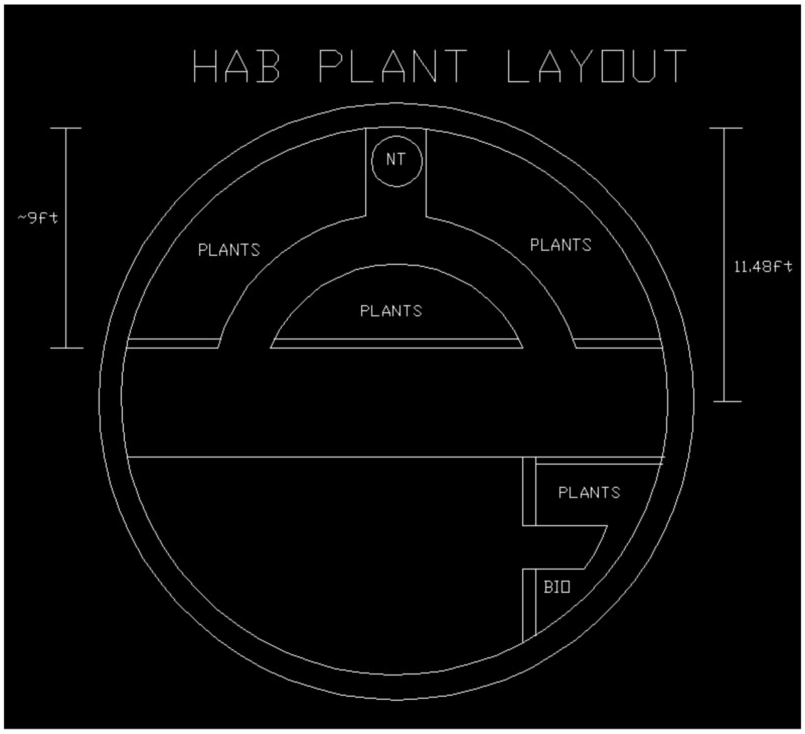


Figure 14-6 This was the initial MHV layout for the bottom floor; it has since been optimized slightly.

The MHV first floor layout depicted above was changed after the volume plants required was reduced due to the reduction of mission time from 500 to only 250 days. The "room" in the bottom right hand corner of Figure 1-6 was removed and consolidated into the other portion of the floor. The hydroponics system was designed with a few key points in mind, specifically the ability of the craft to maintain proper temperature control, and also of the crew to harvest the plants grown. We limit the trays depth to 1 meter so that the crew members are able to reach all the way to the back to manually harvest the plant material. A small walkway goes in-between the two main bodies of plants and it will likely be necessary for some kind of stepping stool to be used when harvesting the top level of plants. The nutrient tank was moved to a more centralized location to maximize internal volume efficiency and also the ability to distribute pipes for the system in an organized fashion. All of these changes result in the finalized configuration drawing below.

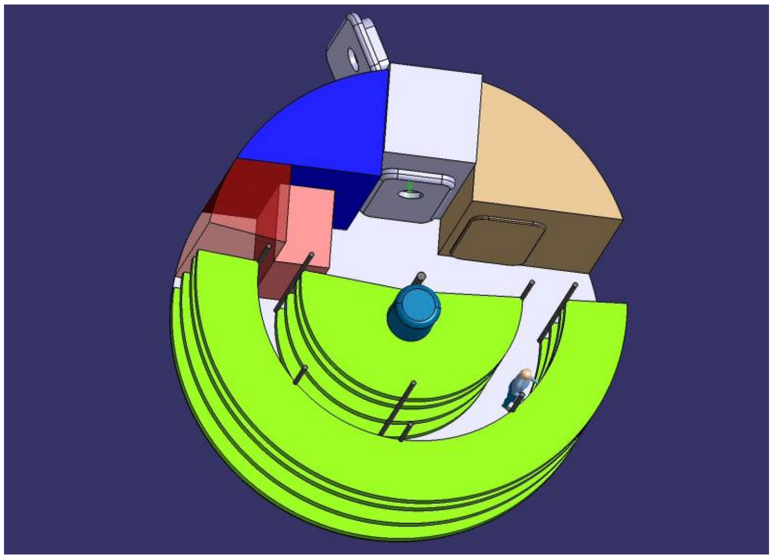


Figure 14-7 (Aaron Kottlowski) The green shelves represent the hydroponics system with the blue cylindrical nutrient tank in the center. A person can be seen in the lower right hand corner for size reference.

References:

[1] NASA Explorers. Plants in Space!. 13 Mar. 2003. New Science. 23 Jan. 2005. <http://liftoff.msfc.nasa.gov/news/2003/news-plants.asp>

[2] Hydroponics UK. The Merging of Nature and Technology. 2005. Esoteric Hydroponics. 20 Feb. 2005. <http://1-hydroponics.co.uk/>

[3] Ledtronics Inc. Miniature Base LED Lamps Add Up to Huge Savings in Industrial Applications. 16 Oct. 2000. Ledtronics News. 20 Feb. 2005. <http://www.led.net/pages/pr_101600.htm>

[4] Barry, Patrick & Philips, Dr. Tony. Water on the Space Station. 2005. First Science. 25 Jan. 2005. <http://www.firstscience.com/site/articles/water.asp>

[5] Dunn, Dr. Bruce. Space Storage of Liquid Hydrogen and Liquid Oxygen. Nov. 2001. Sci.space.tech Newsgroup. 10 Feb. 2005. <http://www.dunnspace.com/cryogen_space_storage.htm>.

[6] Oberg, James. The Elektron Device. 2005. News and Events. 25 Jan. 2005. <http://www.jamesoberg.com/elektron2_tec.html>

[7] Oxygen Generation. 2005 Hamilton Sundstrand. 2003. Space Systems International. 27 Jan. 2005. <http://www.hsssi.com/Applications/EChem/Oxygen/OGA.html>

[8] Gilbert, Canton. Sizing of a Combined Sabatier Reaction and Water Electrolysis Plant for Use in In-Situ Resource Utilization on Mars Sep. 2001. University of Florida. 10 Feb. 2005. <http://www.clas.ufl.edu/CLAS/jur/0901/cantonpaper.html>

[9]Larson, Wiley J., and Linda K. Pranke. Human Spaceflight Mission Analysis and Design. New York: McGraw-Hill Companies, Inc. 1999.

[10] "Advanced Life Support Baseline Values and Assumptions Document" CTSD-ADV-484 A. Crew and Thermal Systems Division. NASA Johnson Space Center. Houston, TX. 16 Aug. 2004. <http://advlifesupport.jsc.nasa.gov/documents/SIMADocs/CR_2004_208941.pdf>

15 Gray, Stas

15.1 Extra Engine for Mars Habitat Vehicle

Author: Stas Gray

A large part of designing the power distribution system is the mass of all the chords involved. To get the mass of all the chords that will be necessary to run power from the reactors, batteries, or fuel cells we wrote a code. The code can be found below.

15.1.1 MLV Code

```
%Stas Gray
%MLV wire sizing code

%Gore's Cable Type SPM variant number 47 for electronics

diameter = 3.69; %measured in mm
r = diameter/2/1000; %in meters


SPM_rho = 48.5; %kg/km

%Communications
t1 = 21.6; %approximate radus of MLV
num_com_sys = 1; %number of communications systems that need power

l = t1; %span of necessary wire
volume = pi*r^2*l*num_com_sys;
weight = l*SPM_rho*num_com_sys;

%Human Factors
num_hum_sys = 3; %number of electronic systems that will require power

volume = volume + pi*r^2*l*num_hum_sys;
weight = weight + l*SPM_rho*num_hum_sys;

%D&C
num_dnc_sys = 1;

volume = volume + pi*r^2*l*num_dnc_sys;
weight = weight + l*SPM_rho*num_dnc_sys;

%Power
```

```
num_pow_sys = 1;

volume = volume + pi*r2^2*l*num_pow_sys;
weight = weight + l*SPM_rho*num_pow_sys;

%Thermal
num_th_sys = 1;

volume = volume + pi*r^2*l*num_th_sys
weight = (weight + l*SPM_rho*num_th_sys)/1000
```

15.1.2 ARV Code

```
%Stas Gray
%ARV wire sizing code

%Gore's Cable Type SPM variant number 47 for electronics

diameter = 3.69; %measured in mm
r = diameter/2/1000; %in meters


SPM_rho = 48.5; %kg/km

%Communications
t1 = 8.3; %approximate radus of ARV
num_com_sys = 1; %number of communications systems that need power

l = t1; %span of necessary wire
volume = pi*r^2*l*num_com_sys;
weight = l*SPM_rho*num_com_sys;

%Human Factors
num_hum_sys = 3; %number of electronic systems that will require power

volume = volume + pi*r^2*l*num_hum_sys;
weight = weight + l*SPM_rho*num_hum_sys;

%D&C
num_dnc_sys = 1;

volume = volume + pi*r^2*l*num_dnc_sys;
weight = weight + l*SPM_rho*num_dnc_sys;

%Power
num_pow_sys = 1;
```

```
volume = volume + pi*r2^2*l*num_pow_sys;
weight = weight + l*SPM_rho*num_pow_sys;

%Thermal
num_th_sys = 1;

volume = volume + pi*r^2*l*num_th_sys
weight = (weight + l*SPM_rho*num_th_sys)/1000
```

15.1.3 CTV Code

```
%Stas Gray
%CTV wire sizing code

%Gore's Cable Type SPM variant number 47 for electronics
%Gore's Cable Type SPP variant number 1 for generator
diameter = 3.69; %measured in mm
diameter2 = 11.8;
r = diameter/2/1000; %in meters
r2 = diameter2/2/1000;

SPM_rho = 48.5; %kg/km
SPC_rho = 542; %kg/km

%Communications
t1 = 25; %distance in meters from generator of CTV to center
w = 20; %width of the middle section of the CTV
num_com_sys = 5; %number of communications systems that need power

l = t1+w; %span of necessary wire
volume = pi*r^2*l*num_com_sys;
weight = l*SPM_rho*num_com_sys;

%Human Factors
t2 = 25; %distance in meters from center of CTV to crew quarters
crew_r = 4.5;
num_hum_sys = 15; %number of electronic systems that will require power

l = t2+2*pi*crew_r;
volume = volume + pi*r^2*l*num_hum_sys;
weight = weight + l*SPM_rho*num_hum_sys;

%D&C
num_dnc_sys = 8;
```

```
l = t2+2*pi*crew_r;
volume = volume + pi*r^2*l*num_dnc_sys;
weight = weight + l*SPM_rho*num_dnc_sys;

%Power
num_pow_sys = 1;

l = t1; %carry the electricty to the transformers in the center
volume = volume + pi*r2^2*l*num_pow_sys;
weight = weight + l*SPC_rho*num_pow_sys;

%Thermal
num_th_sys = 4;
power_r = 5; %radius in meters of non-human section

l = t1+pi*power_r^2;
volume = volume + pi*r^2*l*num_th_sys
weight = (weight + l*SPM_rho*num_th_sys)/1000
```

15.1.4 ELV Code

```
%Stas Gray
%ELV wire sizing code

%Gore's Cable Type SPM variant number 47 for electronics

diameter = 3.69; %measured in mm
r = diameter/2/1000; %in meters


SPM_rho = 48.5; %kg/km

%Communications
t1 = 10; %approximate radus of ELV
num_com_sys = 1; %number of communications systems that need power

l = t1; %span of necessary wire
volume = pi*r^2*l*num_com_sys;
weight = l*SPM_rho*num_com_sys;

%Human Factors
num_hum_sys = 3; %number of electronic systems that will require power

volume = volume + pi*r^2*l*num_hum_sys;
weight = weight + l*SPM_rho*num_hum_sys;
```

```
%D&C
num_dnc_sys = 1;

volume = volume + pi*r^2*l*num_dnc_sys;
weight = weight + l*SPM_rho*num_dnc_sys;

%Power
num_pow_sys = 1;

volume = volume + pi*r2^2*l*num_pow_sys;
weight = weight + l*SPM_rho*num_pow_sys;

%Thermal
num_th_sys = 1;

volume = volume + pi*r^2*l*num_th_sys
weight = (weight + l*SPM_rho*num_th_sys)/1000
```

15.1.5 MHV Code

```
%Stas Gray
%MHV wire sizing code

%Gore's Cable Type SPM variant number 47 for electronics
%Gore's Cable Type SPP variant number 1 for generator

diameter = 3.69; %measured in mm
diameter2 = 11.8;
r = diameter/2/1000; %in meters
r2 = diameter2/2/1000;

SPM_rho = 48.5; %kg/km
SPC_rho = 542; %kg/km

%Communications
D = 7; %distance in meters from generator of MHV to center
L = 5.47; %width of the middle section of the MHV
num_com_sys = 3; %number of communications systems that need power

l = D+L; %span of necessary wire
volume = pi*r^2*l*num_com_sys;
weight = l*SPM_rho*num_com_sys;

%Human Factors
num_hum_sys = 19; %number of electronic systems that will require power
```

```
volume = volume + pi*r^2*l*num_hum_sys;
weight = weight + l*SPM_rho*num_hum_sys;

%D&C
num_dnc_sys = 4;

volume = volume + pi*r^2*l*num_dnc_sys;
weight = weight + l*SPM_rho*num_dnc_sys;

%Power
num_pow_sys = 1;

volume = volume + pi*r2^2*l*num_pow_sys;
weight = weight + l*SPC_rho*num_pow_sys;

%Thermal
num_th_sys = 4;
power_r = 5; %radius in meters of non-human section

volume = volume + pi*r^2*l*num_th_sys
weight = (weight + l*SPM_rho*num_th_sys)/1000
```

15.2 Extra Engine for Mars Habitat Vehicle

Author: Stas Gray

When coming into the Martian orbit the Mars Habitat Vehicle (MHV) must make a series of maneuvers. If there is a problem with the landing and the vehicle needs to enter a Martian orbit, it will need extra fuel and a throttle able engine for that maneuver. We designed such an engine in case the existing nuclear engines were unfit for such a scenario.

A small liquid oxygen liquid hydrogen engine design exists even though it was not deemed necessary for the mission. The engine has an ISP of 307 seconds and requires 10, 677 kg of fuel. The code used to design the engine can be found below.

```
%Stas Gray

%Rocket design for MHV maneuver into Mars orbit
format long g
clear all
clc

R_bar = 8.314472; %J/(mol.K)
delV = 500; %m/s
v_init = 0; %m/s
F = 1000; %N
g = 9.8;

mf=[69737.585,69737.585,69737.585,69737.585,69737.585,69737.585,69737.585];

%Choosing liquid oxygen and liquid hydrogen for fuel
%Obtain data from NASA thermochemistry code (see attached)
OF = 3.5; %One of the inputs to the NASA code
Pc = 34; %chamber pressure in atm

%Output of ODE code
gamma = 1.2;
mol_wt = 9.03;
Tc = 2714;
Isp = 307; %s


%Design Calculations
MR = 1/exp((delV-v_init)/Isp/g); %mo/mf
```

```
mo = MR*mf;
mp = mf-mo;
mp'

Tt = Tc*(1/(1+(gamma-1)/2));
Pt = Pc*(1/(1+(gamma-1)/2))^(-gamma/(gamma-1));
m_dot = F/Isp;
R = R_bar/mol_wt;

At = m_dot/Pt*sqrt(R*Tt/gamma/g);
Me = sqrt(2/(gamma-1)*((Pc)^((gamma-1)/gamma)-1));
Ae                =                At/Me*((1+(gamma-1)/2*Me^2)/((gamma+1)/2))^((gamma+1)/2/(gamma-1));
```

15.3 Communication Satellites

Author: Stas Gray

We performed a case study to determine the best possible power source for the Mars Orbit Satellites (MORS). Since communication must be available at all times during the mission a long life span at a great distance from earth is the driving requirement. Among sources considered were batteries, RTG's, solar cells, fuel cells, and nuclear fission reactors. The limiting parameter left only the options of solar cells and nuclear reactors available.

The power requirement for communications and dynamics and controls summed to be twenty kilowatts. To generate that kind of power the solar panels on the MORS satellites would need to be unreasonably large. The final choice was made to use the same reactor that had been designed for the Crew Transport Vehicle (CTV) and Mars Habitat Vehicle (MHV). The reactor produces 44.4 kW of power and takes up 2.3 cubic meters of space with a mass of five thousand kilograms.

While in the conceptual design phase a sketch of one of the satellites was made, as seen below in Figure 3-1.

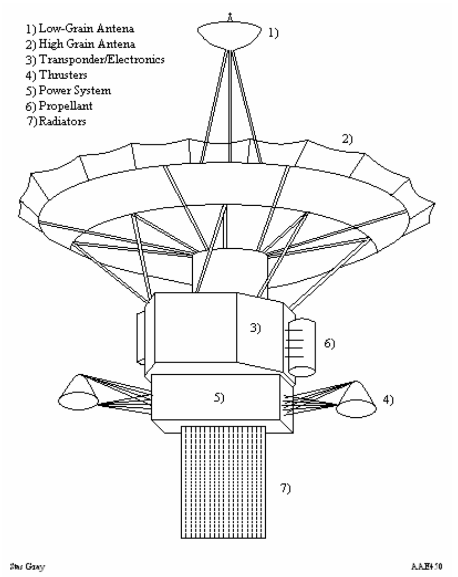


Figure 3-1 Communication Satellite

The initial mass approximation was 7200 kg but that mass is now almost 9000 kg. The largest part of the satellite is the shielding for the reactor which totals four thousand kg. To launch the MORS we will be using the Earth Launch Vehicle (ELV). After the ELV takes the MORS out of orbit each satellite is equipped with its own propulsion system to place it in Mars orbit.

15.4 Preliminary Fission Reactor Design

Author: Stas Gray

The power group decided that our first task should be designing the power source for the Crew Transport Vehicle (CTV). Every reasonable power source was considered at first. Strong arguments were found for solar and nuclear power. After considering the significant drop-off in solar power at great distances away from the sun, solar power was ruled out. Other considerations for solar power were brought up in the design of the Mars Habitat Vehicle (MHV) and the Mars Orbit Satellites (MORS) and these will be discussed later on in the appendix.

Nuclear power brought many options. Fusion reactors, fission reactors, and RTG's were considered. Since fusion reactors have not yet been developed or tested and RTG's only provide a small amount of power, a fission reactor was decided upon. Fission reactors have been designed and tested for space travel. The reactor we based this design on is the SP100 which was developed by the military in the 1980's. The initial design consisted of the flow chart that can be seen below in Figure 1.

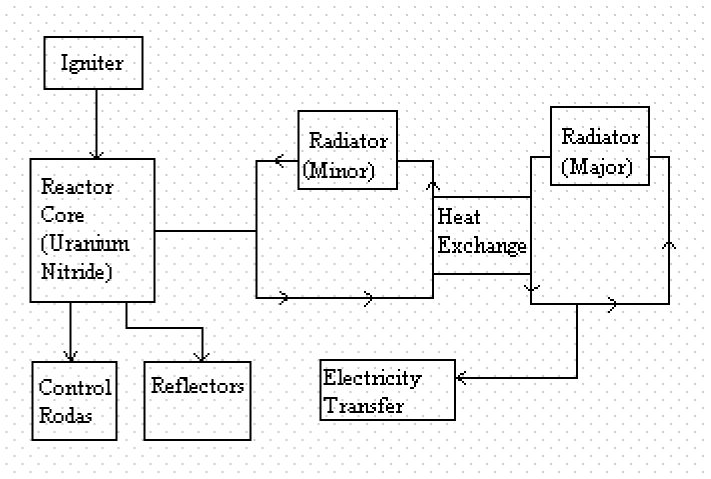


Figure 3-1 Reactor Design

All of the components are specially designed for mission Legend. The shielding which is not mentioned in the flow chart was designed specifically for each vehicle depending on the radioactive hazards.

16 Gustafson, Eric

16.1 Power Systems Alternatives and Reliability for Project Legend

Author: Eric Gustafson

16.1.1 Solar Power System Characterization

For long-duration vehicles with power demands less than about 5kW, solar power is an excellent choice for a primary power source. The only vehicle that met this condition in our project was the Mars Habitat Vehicle (MHV) during its transit to Mars, however, there is already a nuclear reactor on board, so another power system is unnecessary. Table 16-1 shows existing or previously studied solar power systems for comparison with the needs of this project [1,2]. The last row is a system that would meet the needs of the MHV in transit.

Table 16-1 Solar Power Systems

Description	Power near Earth (kW)	Power near Mars (35% Earth) (kW)	Mass (kg)	Volume (m^3)	Array Area (m^2)
ISS PV Module	28	9.8	3710		750
Mars DRM	30	10.5	3842.5	4.417	719
Mars DRM	5	1.8	1048	1.289	203.5
ISS scaled	80	28	11000		2142.9
*Mars DRM scaled	14.3	5	2500	~3.68	581.4

The power levels of the vehicles are either too large for solar power to be viable, as in the case of the Crew Transport Vehicle (CTV), or too short in duration, such as the Ascent & Recovery Vehicle (ARV).

16.1.2 Power System Reliability

Space power systems are one of the most reliable elements in space missions. Even the most advanced power systems we use for Project Legend are evolutionary in nature, which makes the risk of power system failure very small. In addition to enhance reliability of power systems, we choose to include appropriate redundancies, such as a dual-core nuclear reactor. Table 16-2 summarizes the reliabilities of the Project Legend power system.

Table 16-2 Power System Reliability

System Description	Possible Failure Modes	Failure Probability
Solar array	Meteoroid impacts, failed bypass diodes, degradation over time	Very small (<.1%) within lifetime
Fuel cell	Membrane failure and degradation	Shuttle: <1% failure rate over more than 1000 days in flight [2]
Battery	Short circuit, electrolyte leakage	Very small (<.1%) within lifetime
Nuclear reactor	Thermal system failure	< 0.07% failure chance over mission resulting in radiation exposure [3]
Distribution / Regulation (PMAD)	Short circuits, random electronic failure	Very small (<.1%) within lifetime

16.1.3 References

[1] Bailey, S., Landis, L., and McKissock, B. "Designing Power Systems." Human Spaceflight Mission Analysis and Design. Larson, W., and Pranke, L. New York, New York: McGraw Hill, 1999. Pages 643-664.

[2] CBS News Space Statistics. "SPACE SHUTTLE ORBITER STATISTICS."

<http://cbsnews.cbs.com/network/news/space/spacestats.html#ORBITER%20STATISTICS>

[3] Damon, D. "SP-100 Space Reactor Risk Assessment: Lessons Learned."

<http://www3.inspi.ufl.edu/space/program/abstracts/1159.pdf>

16.2 Electric Propulsion Possibilities for Project Legend

Author: Eric Gustafson

16.2.1 Electric Propulsion Introduction

Electric propulsion is considered for use as a primary propulsion system in Project Legend due to the extremely high Isp and the resulting propellant mass savings. Unfortunately, we see from the following discussion that electric propulsion is not feasible for these two vehicles. Electric propulsion is used for other purposes in the project, however, as discussed in the Crew Transport Vehicle (CTV) Guidance and Control Hardware section and the Communications Satellite System section.

16.2.2 Electric Propulsion for the Mars Habitat Vehicle

Extensive analysis was performed to investigate the possibility of using electric propulsion for the Mars Habitat Vehicle (MHV), including low-thrust trajectory optimization. The end result is that the MHV is simply too massive for electric propulsion to be feasible with available technology. Using an engine with an Isp of 5,000 seconds, and an input power of 100 kW, it would take the MHV well over 10 years to complete the trip to Mars. Figure 16-1 shows an example plot of the trajectory for a very optimistic case where the mass of the MHV was cut in half, and the power supplied to the ion engine was doubled. The spiral nature of the trajectory is the reason for the significant trip time due to the multiple revolutions around the Sun.

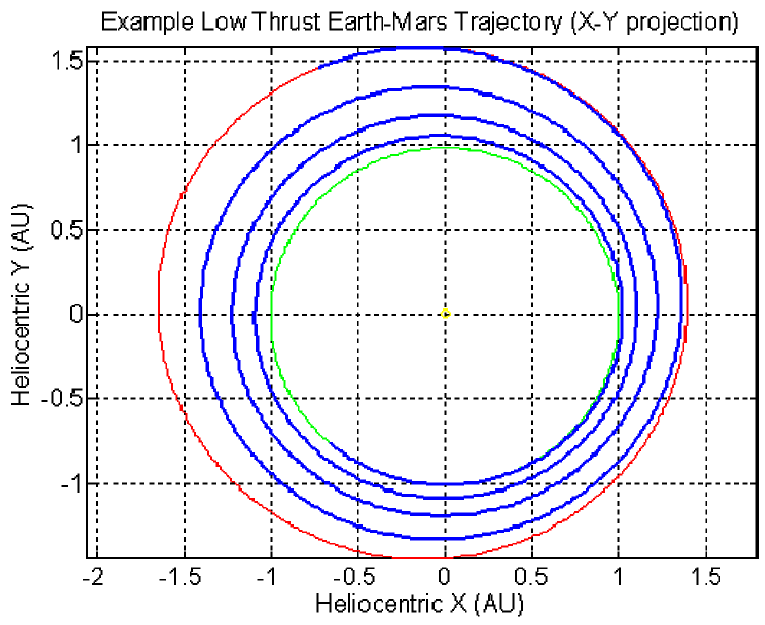


Figure 16-1 Optimistic MHV Low-Thrust Trajectory

Given the disadvantages of low thrust propulsion for this particular application, we choose to use Nuclear Thermal Rockets (NTRs) as the primary propulsion method for the MHV. Although the propellant mass ratios (final mass divided by initial mass) are better with high-Isp electric propulsion in this case, the decision to use NTRs is driven by the schedule of the mission and long-term storage concerns for the food on the MHV.

16.2.3 Electric Propulsion for the Crew Transport Vehicle

In our initial orbit design analysis, it was determined that there could be large mass savings by using electric propulsion to boost the Crew Transport Vehicle (CTV) from Low Earth Orbit (LEO) to a High-Period Parking Orbit (HPPO) before using the Nuclear Thermal Rockets (NTRs) to propel the CTV the rest of the way to Mars. Unfortunately, further analysis reveals that electric propulsion does not save any mass for the same reason that electric propulsion is infeasible for the MHV – the mass of both vehicles is too large for the available power. For the spiral time to be less than one years, it would require 21 electric thrusters, each producing 2.5 N of thrust, using 100kW of power. The large number of thrusters necessitates the launch of many nuclear reactors, which causes the mass of a low-thrust system to balloon to 240,000 kg. It would take two launches to place the electric propulsion hardware into LEO. On the other hand, a NTR system would have a mass of only 82,000 kg and could be easily launched with one Earth Launch Vehicle (ELV). The clear solution is to use the NTRs already on the CTV and simply launch an auxiliary propellant tank, which will be jettisoned shortly after use.

16.2.4 Analysis of Electric Propulsion for the Mars Habitat Vehicle

The low-thrust optimization analysis was performed using an available code from NASA called CHEBYTOP. The code is fairly easy to use and is publicly available from [1]. The following sections or the code used to wrap around CHEBYTOP and perform the analysis.

16.2.4.1 MHV Low-Thrust PERL Wrapper

This program runs many cases varying the launch date and trip time of the orbit in order to find the optimal combination.

```
#!/usr/bin/perl -w
#Eric Gustafson
#This file is “loop.pl”
#This program creates input files for CHEBYTOP for various launch dates, trip times, and heliocentric
revolution numbers, then calls CHEBYTOP and analyzes the output file, storing pertinant information in
a file which can be read into MATLAB to create plots
@days_in_month = (31,28,31,30,31,30,31,31,30,31,30,31);
open DATA, ">data.txt" or die "Cannot open data.txt for append :$!";
print DATA "%year month day JDL tend rn m/m0 a\n";
```

```
for ($year=2014; $year<=2029; $year++){
   for ($month=1; $month<=12; $month+=2){
      for ($day=1; $day<=$days_in_month[$month-1]; $day+=32){
         for ($tend=2000; $tend<=3000; $tend+=100){
        for ($rn=.25; $rn<=3.25; $rn++){
           #define the filenames
           $out_inp=$year."_".$month."_".$day.".inp";
           $out_out=$year."_".$month."_".$day.".out";
           system("cp 450_head $out_inp");

           #print the changing inputs to the file
           open OUT, ">>$out_inp" or die "Cannot open $out_inp for append :$!";
           print OUT " JDL=".$year.",".$month.",".$day.",\n";
           print OUT " tend=".$tend.",\n";
           print OUT " rn=".$rn.",\n";
           print OUT " \&\n \&end\n";
           close OUT;

           #run CHEBYTOP
           system("../../chebytop $out_inp");

           #analyze the output
           open IN, "$out_out" or die "Cannot open $out_out for reading :$!";
           $a_max = 0;
           while (<IN>){
              if (/Departure planet/){
                 @line_split1 = split /\s+/, $_;
              }
             if (/time         x         y         z/){
                $_ = <IN>; $_=<IN>; #kill the blank line and start on the next line (the one with the
data)
                while (/./){ #keep looping as long as there's a decimal point somewhere in the line
                   $_ = <IN>;
                   @data_line = split /\s+/, $_;
                   if ($data_line[8] > $a_max){ #if the acceleration is a new high, store it
                      $a_max = $data_line[8];
                   }
                }
             }
             if (/m\/m0/){
                @line_split2 = split /\s+/, $_;
                print DATA $year." ".$month." ".$day." ".$line_split1[7]." ".$tend." ".$rn."
".$line_split2[$#line_split2]." ".$a_max."\n";
             }
           }
           close IN;

           #clean up files
             #system("gzip $out_out");
             #system("mv $out_out".".gz data/.");
           #system("mv $out_out data/.");
           system("rm $out_inp");
           system("rm $out_out");
        }
      }
        }
   }
}
close DATA
```

16.2.4.2 CHEBYTOP Input File Template

The following file was used as a CHEBYTOP template input file. The code above adds case specific inputs to the file, feeds it to CHEBYTOP, then analyses the output automatically.

```
&INPUT  HEAD='BASELINE',
 SHOTA='Earth',BULSI='Mars',npow=0,
 is=5000,bb=.65,dd=0.,p0=100.,m0=131395.0,
 nv1=3,nv2=3,
 alt=200.,
 alta=3747.,
 nlv=-1, nb1=2, nb2=2,
 delpo=1.,
 ;
```

16.2.4.3 CHEBTYOP Trajectory Plotting

The following MATLAB m-file reads in output files from CHEBYTOP and plots the low-thrust orbit, overlaid with the planet's orbit tracks.

```
%Eric Gustafson
%This code reads in the trajectory produced by CHEBYTOP, calculated the orbits of Earth and Mars from
%the ephemeris data embedded in CHEBYTOP, then plots the trajectory overlaid with the planetary orbit
tracks

%find oribital elements from chebytop:
clear; clc; close all
format compact
km_per_AU = 1.49597807e8; %km/AU
sec_per_year = 365*24*3600; %seconds/year

%DATA FROM CHEBYTOP - system/pldata.f
%EQUATION is from system/plelem.f
% Orbit Elements for Earth
% earth_elem = [a a1 a2; e e1 e2; ...]
% a (AU), e, i (deg), Omega (deg), omega (deg)
earth_elem = [1.00000023d0,.01670911d0,.0d0,174.876384d0,102.940753d0, 357.525443d0;
               .0d0,-.42052d-4,.01305635d0,-.24161358d0, .32343001d0,.9856002586d0;
               .0d0,-.126d-6,-.91465d-5,.9822d-5, .150003d-3,-.155d-3]'
% Orbit Elements for Mars
mars_elem = [1.52369150d0,.09340489d0,1.849709d0,49.559177d0,336.059380d0,19.388107d0;
              .0d0,.9191d-4,-.82027565d-2,-.29546843d0,.4441893d0,.5240207766d0;
              .0d0,-.77d-7,-.199914d-4,-.6487619d-3,-.18d-3,.1825972d-3]'

%define elements at launch date:
epoch = 2451545.0d0; %julian date of J2000
tau = 2457410.502; %departure date (from chebytop output)
time = tau - epoch;
t = time / 36525; %in centuries
for i=1:5
    earth_orb(i) = earth_elem(i,1) + t*(earth_elem(i,2) + t*earth_elem(i,3));
    mars_orb(i) = mars_elem(i,1) + t*(mars_elem(i,2) + t*mars_elem(i,3));
    %convert to km and radians
    if (i == 1)
        earth_orb(i) = earth_orb(i)*km_per_AU;
        mars_orb(i) = mars_orb(i)*km_per_AU;
    end
    if (i==3 | i==4 | i==5)
        earth_orb(i) = earth_orb(i)*pi/180;
        mars_orb(i) = mars_orb(i)*pi/180;
    end
end

%--------------------------------------------------
%--------------------------------------------------
%old plotting program now:
a_mars = mars_orb(1); %km
e_mars = mars_orb(2);
i_mars = mars_orb(3); %rad
Omega_mars = mars_orb(4); %rad
omega_mars = mars_orb(5); %rad
%
p_mars = a_mars*(1-e_mars^2); %km

%Earth orbital elements
a_earth = earth_orb(1); %km
e_earth = earth_orb(2);
```

```
i_earth = earth_orb(3); %rad
Omega_earth = earth_orb(4); %rad
omega_earth = earth_orb(5); %rad
%
p_earth = a_earth*(1-e_earth^2); %km

%from chebytop output
r_earth_xyz = [ -.52417  .83300 -.00003]*km_per_AU; %km
v_earth_xyz = [ -5.42095 -3.37039 .00014]*km_per_AU/sec_per_year; %km/s
r_earth_hat_xyz = r_earth_xyz / norm(r_earth_xyz);

r_mars_xyz = [1.01226 -.94861 -.04472]*km_per_AU; %km
v_mars_xyz = [ 3.68905 4.16735 -.00322]*km_per_AU/sec_per_year; %km/s
r_mars_hat_xyz = r_mars_xyz / norm(r_mars_xyz);


dtheta = .01;
thst = 0:dtheta:2*pi+dtheta;
r_earth = p_earth./(1+e_earth*cos(thst));
r_mars = p_mars./(1+e_mars*cos(thst));
for i = 1:length(thst)
    th_earth = thst(i) + omega_earth;
    th_mars =  thst(i) + omega_mars;
    earth_orbit_xyz(:,i) = rotation_matrix(Omega_earth, th_earth, i_earth)'  * [r_earth(i) 0 0]';
    mars_orbit_xyz(:,i)  = rotation_matrix(Omega_mars, th_mars, i_mars)'  * [r_mars(i) 0 0]';
end

plot(earth_orbit_xyz(1,:)/km_per_AU, earth_orbit_xyz(2,:)/km_per_AU, 'g-')
hold on
plot(mars_orbit_xyz(1,:)/km_per_AU, mars_orbit_xyz(2,:)/km_per_AU, 'r-')
plot(0,0,'yo')

load plot_data.txt;
x_traj = plot_data(:,2);
y_traj = plot_data(:,3);
H = plot(x_traj,y_traj, 'b-');
set(H, 'LineWidth', 2);
axis equal
grid on
title('Example Low Thrust Earth-Mars Trajectory (X-Y projection)');
xlabel('Heliocentric X (AU)')
ylabel('Heliocentric Y (AU)')
%-------------------------------------------
function out = rotation_matrix(Omega, th, i)
%vector_xyz = rotation_matrix(Omega, th, i)  * vector_rthh
%vector_rthh  = rotation_matrix(Omega, th, i)' * vector_xyz
out = [cos(Omega)*cos(th)-sin(Omega)*cos(i)*sin(th), -cos(Omega)*sin(th)-sin(Omega)*cos(i)*cos(th),
sin(Omega)*sin(i);
       sin(Omega)*cos(th)+cos(Omega)*cos(i)*sin(th), -sin(Omega)*sin(th)+cos(Omega)*cos(i)*cos(th), -
cos(Omega)*sin(i);
                   sin(i)*sin(th)                 ,               sin(i)*cos(th)                  ,
cos(i)    ];
```

16.2.4.4 Mass Fraction Contour Plot Code

The following MATLAB m-file reads in the results from many CHEBYTOP runs and creates a contour plot to show the effect that the launch date and trip time have on mass fractions.

```
%Eric Gustafson
%This code reads in the data file created by "loop.pl" which contains all the output from CHEBYTOP
%The plot produced is a contour plot of the mass fraction as a function of launch date and trip time

clc; clear all; close all

load data.txt
S = size(data);

%make vector of existing JDL's
JDL(1) = min(data(:,4));
```

```
c = 1;
for i=2:S(1)
    if data(i,4) ~= JDL(c) %if the current line's JDL is different than the previous stored JDL,
then...
        %this only works because the JDL's increment sequentially
        c = c+1;
        JDL(c) = data(i,4);
    end %otherwise, just keep going
end

%make vector of existing tend's
tend(1) = min(data(:,5));
tend_ordered = sort(data(:,5));
c = 1;
for i=2:S(1)
    if tend_ordered(i) ~= tend(c) %if the current line's tend is different than the previous stored
tend, then...
        c = c+1;
        tend(c) = tend_ordered(i);
    end %otherwise, just keep going
end

%initialize mass fraction matrix
mf = zeros(length(JDL), length(tend));
a_limit = .134*4; %acceleration limit in AU/yr^2
for i=1:S(1)    %loop through the whole file and input the numbers
    JDL_index = find(JDL==data(i,4));
    tend_index = find(tend==data(i,5));
    if (data(i,7) > mf(JDL_index,tend_index) & data(i,8)<=a_limit)%if this line's mf is better than
what's already store, use the new number
        mf(JDL_index,tend_index) = data(i,7);
    end
end
if max(max(mf)) ==0
    disp 'No solution with given restraints';
    x=-1:1;
    plot(x,x); hold on; plot(x,-x); axis equal
    return;
end

%make contour plot
[JDL_matrix tend_matrix] = meshgrid(JDL,tend);
contourf(JDL_matrix-JDL(1), tend_matrix, mf', [0:1/20:1]);
colorbar
%make more readable colors
x = colormap(hot);
for i=1:64
    if i<=38
        x(i,:) = [0 0 0]; %if the mass fraction is below a certain threshold, just black it out
    end
end
colormap(x);
xlabel(['Launch Date (Days after Jan. 1 2014)', char(10), '[Jan. 1 2014 = JDL 2456658.5]'])
ylabel('Trip time (Days)')
title('Mass fraction (m_f / m_0) vs. Trip Time and Launch Date')
```

16.2.5 Analysis of Electric Propulsion for the Crew Transport Vehicle

16.2.5.1 Time to Spiral From Low Earth Orbit to a Long Period Parking Orbit using One Thruster

We wish to find how long it will take the Crew Transport Vehicle (CTV) to spiral from a Low Earth Orbit (LEO) at an altitude of 200 km to a Long Period Parking Orbit (LPPO) with a period of 10 days. We start by noting that the the velocity of the CTV in LEO at 200km, v_c, is 7.784 km/s. The velocity at perigee of a 10-day period LPPO, v_p, is 10.92 km/s. This means the ΔV (assuming an impulsive burn) to get from LEO to the LPPO is $v_p - v_c = 3.13$ km/s.

This ΔV number can be treated as an impulse per mass. Impulse, *I*, is

$$\int F dt$$

Equation 16-1

where *F* is the thrust and *t* is the burn time. For a constant thrust, the impulse per mass is simply

$$\frac{Ft}{m}$$

Equation 16-2

where *m* is the mass of the spacecraft. This impulse per mass can be set equal to the ΔV to get a rough estimate of spiraling time with continuous thrusting:

$$\Delta v = \frac{Ft}{m} \Rightarrow t = \frac{m\Delta v}{F} = \frac{(400{,}000\,\text{kg})(3{,}130\,\text{m/s})}{2\,\text{N}} = 6.26\times10^{8}\ \text{s} = 19.9\ \text{years}$$

Equation 16-3

The bottom line of this analysis is that it takes on the order of 20 years to spiral out from LEO to a LPPO with a 2 N thruster. This motivates us to find what kind of electric propulsion system would be needed to complete the maneuver in a shorter period of time. The analysis which answer this question follows in the next section.

16.2.5.2 Electric Propulsion System To Spiral From Low Earth Orbit to a Long Period Parking Orbit using One Thruster in a Given Period of Time

The goal of this analysis is to find the number of electric thrusters needed to complete a given Δv in a given burn time, t_b. We start with these initial assumptions and nomenclature:

- $I_{sp} = 5000\text{s}$ [specific impulse]
- $\Delta v = 3130$ m/s [circular LEO at 200km altitude to 10-day period orbit with 200km periapsis altitude]
- $m_{CTV,full} = 300{,}000$ kg [this is the CTV mass loaded with hydrogen]
- $m_{th} = 10{,}000$ kg [this is the mass of a thruster, including nuclear power source]
- $F_{th} = 2.5$ N [force of one thruster]
- $t_b = 1\ \text{year} = 31{,}536{,}000\,\text{s}$ [do the maneuver in less than a year]

- Let n be the number of thrusters on the spacecraft

We seek to calculate the number and mass of thrusters, propellant, and power supplies needed to provide the given Δv within the allowed burn time. We start by rewriting the ΔV equation using Newton's Second Law:

$$\Delta v = \int a(t)dt = \int_0^{t_b} \frac{F_{total}}{m_{total}(t)} dt = \int_0^{t_b} \frac{F_{th}n}{m_{CTV,full} + m_{th}n + m_p - \dot{m}t} dt$$

Equation 16-4

The mass flow rate, $\dot{m}$, of the ship is the propellant mass divided by the burn time. Equation 16-4 can be written as:

$$\Delta v = \int_0^{t_b} \frac{F_{th}n}{m_{CTV,full} + m_{th}n + m_p\left(1 - \frac{t}{t_b}\right)} dt$$

Equation 16-5

The propellant mass (for the "space-tug") is given by the rocket equation:

$$m_p = (m_{CTV,full} + m_{th}n)\left(1 - e^{\frac{-\Delta v}{gIsp}}\right)$$

Equation 16-6

Substituting and pulling constants out of Equation 16-4 gives

$$\Delta v = \frac{F_{th}n}{m_{CTV,full} + m_{th}n} \int_0^{t_b} \frac{1}{1 + \left(1 - e^{\frac{-\Delta v}{gIsp}}\right)\left(1 - \frac{t}{t_b}\right)} dt$$

Equation 16-7

Completing the integration gives

$$\Delta v = \frac{F_{th}n}{m_{CTV,full} + m_{th}n} \left[\frac{t_b}{e^{\frac{-\Delta v}{gIsp}} - 1} \ln\left(\frac{e^{\frac{-\Delta v}{gIsp}} - 1}{t_b} t + 2 - e^{\frac{-\Delta v}{gIsp}} \right) \right]_{t=0}^{t=t_b}$$

Equation 16-8

Plugging in the integration limits:

$$\Delta v = \frac{F_{th} n}{m_{CTV,\,full} + m_{th} n}\left[\frac{t_b}{1 - e^{\frac{-\Delta v}{gIsp}}}\ln\left(2 - e^{\frac{-\Delta v}{gIsp}}\right)\right]$$

Equation 16-9

Now the number of thrusters, *n*, can be solved for explicitly:

$$n = \frac{\Delta v \cdot m_{CTV,\,full}}{\left[\frac{t_b}{1 - e^{\frac{-\Delta v}{gIsp}}}\ln\left(2 - e^{\frac{-\Delta v}{gIsp}}\right)\right]F_{th} - \Delta v \cdot m_{th}}$$

Equation 16-10

Plugging in given numbers for the CTV yields

$$n = \frac{(3130\ \text{m/s})\cdot(300000\ \text{kg})}{\left[\frac{(31536000\ \text{s})}{1 - e^{-3130\ \text{m/s}/(9.81\ \text{m/s})(5000\ \text{s})}}\ln\left(2 - e^{-3130\ \text{m/s}/(9.81\ \text{m/s})(5000\ \text{s})}\right)\right](2.5\ \text{N}) - (3130\ \text{m/s})\cdot(10000\ \text{kg})}$$

Equation 16-11

The answer for our case is *n* is 21 thrusters. As mentioned before, the mass of such a system (including nuclear power sources and radiators) is 240,000 kg.

16.2.5.3 CTV Electric Propulsion Spiral Estimation Code

The following is a MATLAB m-file which carries out the above equations, and reports the masses associated with the electric propulsion option versus the nuclear thermal option.

```
%Eric Gustafson - 2/14/05
%This codes simply calculates the number and related masses of low-thrust
%electric thrusters needed to boost the CTV from LEO to LPPO in a given
%amount of time (using the equation derived in the previous section).

clc; clear; format long; format compact

m_CTV = 300000; %CTV mass, full
m_th = 10000; %thruster mass with nuclear reactor + radiator
deltav = 3130; %m/s
g = 9.81; %m/s
Isp = 5000; %s
tb = 1 * 365 * 24 * 3600; %burn time in seconds
F_th = 2.5; %N

n = deltav * m_CTV / (tb/(1-exp(-deltav/(g*Isp)))*log(2-exp(-deltav/(g*Isp)))*F_th - deltav*m_th)
mass_hardware = m_th * n

mp_nuke_elec = (m_CTV+mass_hardware)*(1-exp(-deltav/(g*5000)))
m_total_elec = mass_hardware + mp_nuke_elec

Isp_nuke_thermal = 1000;
mp_nuke_thermal = m_CTV*(1-exp(-deltav/(g*Isp_nuke_thermal)))
```

16.2.6 References

[1] Riehl, J. "Re: Hello from Purdue." E-mail to the author. 18 Jan. 2005.

<http://trajectory.grc.nasa.gov/tools/chebytop.shtml>

[2] Masa Okutsu. "AAE450 Low thrust traj. optimization." E-mail to the author. 1 Jan. 2005.

16.3 Mars Habitat Vehicle (MHV) Nuclear Reactor Placement

Author: Eric Gustafson

16.3.1 MHV Nuclear Reactor Placement Overview

To decide how to handle the placement of the surface nuclear reactor we analyze the reactor's radiation emission and power loss resulting from remotely locating the reactor. We find that when the reactor is placed 350 m away from the Mars Habitat Vehicle (MHV) that the radiation emitted from the minimally shielded reactor is sufficiently low enough to leave the MHV completely unshielded. Even with no radiological protection on the MHV, the surface reactor only contributes a small portion of the mission-duration radiation doses received by the crew. Therefore it is highly advantageous to move the reactor away from the MHV to save mass on shielding. The remote placement of the reactor necessitates an increased power production of about 20 kW due to line losses, and the cables have a mass of 320 kg.

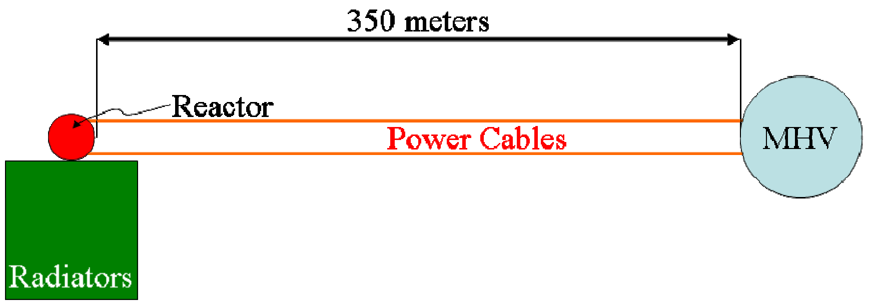


Figure 16-2 Schematic Placement of Surface Reactor

The reactor will be deployed using a rover which is integrated into the reactor structure, as shown in Figure 16-3 Reactor Rover (courtesy Aaron Kottlowski).

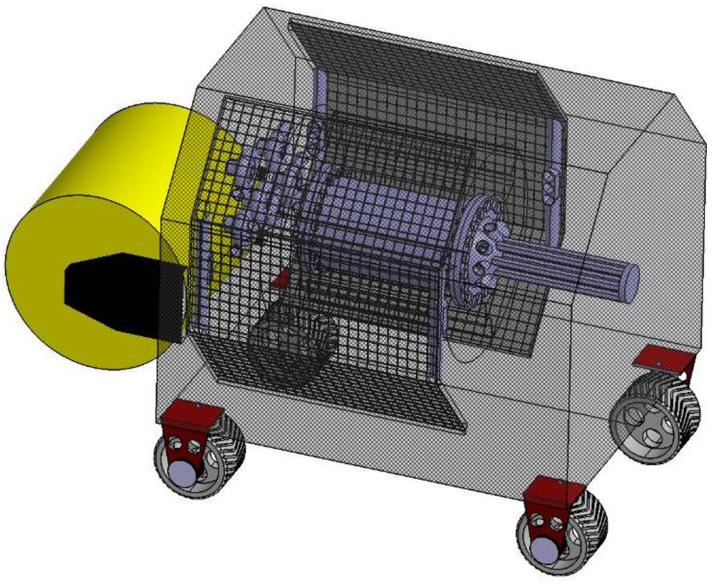


Figure 16-3 Reactor Rover (courtesy Aaron Kottlowski)

16.3.2 MHV Nuclear Reactor Placement Analysis

The following MATLAB m-file calculates the resistance and power loss incurred by running wires between the MHV and the reactor. The inputs are the maximum allowable heating rate in the wire, the power draw of the MHV, and voltage output of the reactor. The final solution requires an iterative procedure coupled with the radiation calculations, performed by Aaron Kottlowski.

```
%Eric Gustafson - 2/28/05
%This code calculates the mass and other properties of power cables if we
%locate the nuclear reactor away from the HAB

clc; clear; close all; format long; format compact

%physical constants for copper
rho = 1.68e-8; %Ohm*m (copper resistivity)
dens = 8900; %kg/m^3 (copper density)
c = 386; %J/(kg * K) (copper specific heat)

%parameters
V_plant = 500; %V (voltage out of nuclear plant)
P_hab = 140e3 %W (power needed at HAB)
P_plant = P_hab+ 19.2e3 %W (power out of nuclear plant)
P_wire_per_meter = 30 %W (power dissipated by one wire per meter of wire)
%P_wire100 = P_wire_per_meter

%calculations that are independant of length
i = P_plant / V_plant %current in the loop
V_hab = P_hab / i
V_wire = (V_plant - V_hab) / 2 %voltage across one wire (of two)
L_max = (P_plant - P_hab) / 2 / P_wire_per_meter %maximum distance based on the power per length of
wire

L_vector = 1:350; %m
n = 0; %counter
for L = L_vector
    n = n+1;
```

```
        if L<=L_max
            P_wire(n) = P_wire_per_meter * L;
            R_wire(n) = P_wire(n) / i^2;
            A_wire(n) = rho * L / R_wire(n);
            d_wire(n) = sqrt(4*A_wire(n)/pi);
            m_wire(n) = A_wire(n)*L * dens; %mass of one wire
            T_wire(n) = P_wire(n) * 60 / (m_wire(n) * c);
        else
            P_wire(n) = NaN;
            R_wire(n) = NaN;
            A_wire(n) = NaN;
            d_wire(n) = NaN;
            m_wire(n) = NaN; %mass of one wire
            T_wire(n) = NaN;
        end
    end

    %output:
    L_act = 320;
    n_100 = find(L_vector==L_act);
    fprintf(['\n\n****** Information at ', num2str(L_act),'m ******\n']);
    fprintf('Total wire mass (kg): %f\n', 2*m_wire(n_100));
    fprintf('Total power loss (kW):  %f\n', 2*P_wire(n_100)/1000);
    fprintf('Total power loss per meter of wire (W/m):  %f\n', P_wire(n_100)/L);
    fprintf('Wire diameter (cm):  %f\n', d_wire(n_100)*100);
    fprintf('Total wire resistance (Ohms):  %f\n', R_wire(n_100));
    fprintf('Temperature rise in 1 minute w/ no cooling (C):  %f\n', T_wire(n_100));


    figure
    plot(L_vector,2*m_wire)
    grid on
    xlabel('Reactor Distance from the HAB (m)')
    ylabel('Total Mass of Power Cables (kg)')
    title('Cable Mass vs. Reactor-HAB Separation Distance')

    figure
    plot(L_vector, 2*P_wire/1000)
    grid on
    xlabel('Reactor Distance from the HAB (m)')
    ylabel('Power Loss Through Cables (kW)')
    title('Power Loss vs. Reactor-HAB Separation Distance')
```

16.4 Mars Habitat Vehicle (MHV) Launch Windows

Author: Eric Gustafson

Contributors: Meredith Evans, Kimberly Mrozek

16.4.1 MHV Launch Window Visualization

We plot the ΔV required for the Mars Habitat Vehicle mission as a function of launch date and trip time. The results, shown in Figures 2 through 8, clearly show periodic minimum ΔV solutions occurring every synodic period. Delta-V's greater than 10 km/s are blacked out because the mission would not be launched in this region. For details on the computation of the ΔV for a general case, see Kim Mrozek's sections on MHV Non-Free Return Trajectory.

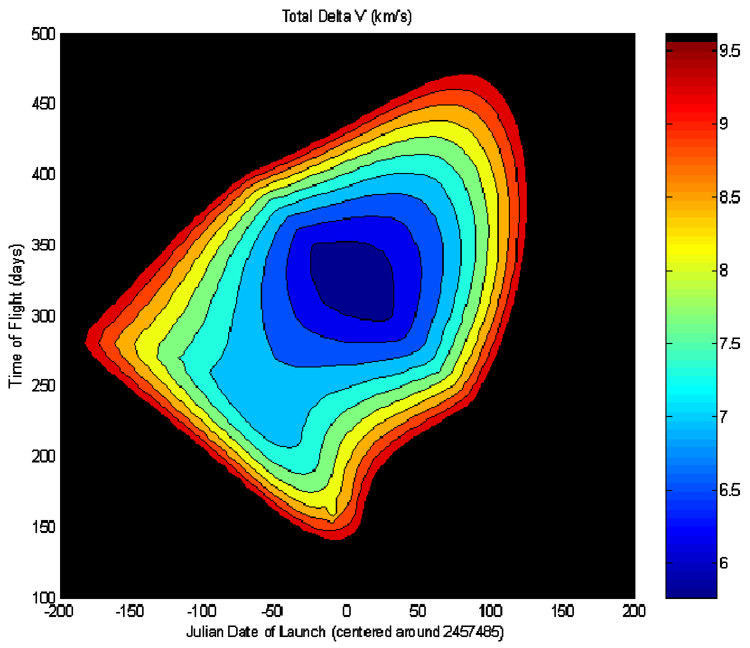


Figure 16-4 MHV Delta-V

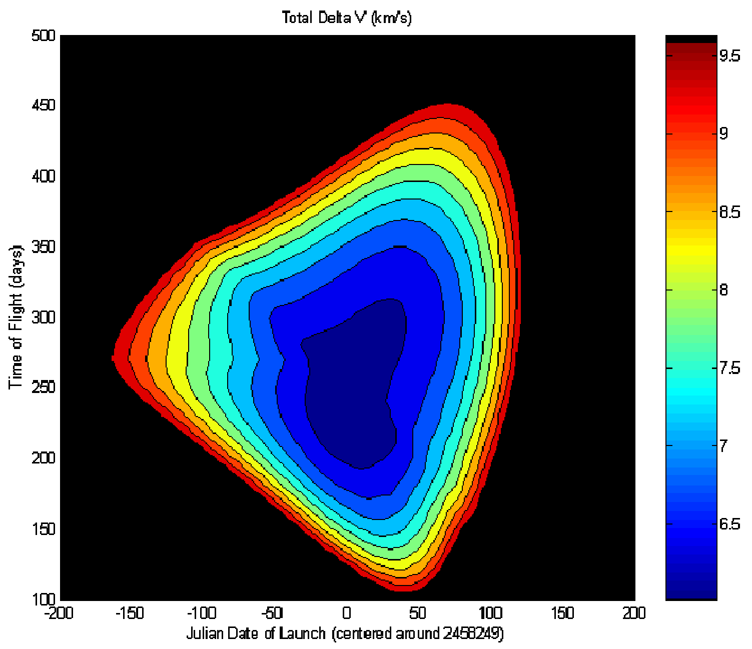


Figure 16-5 MHV Delta-V

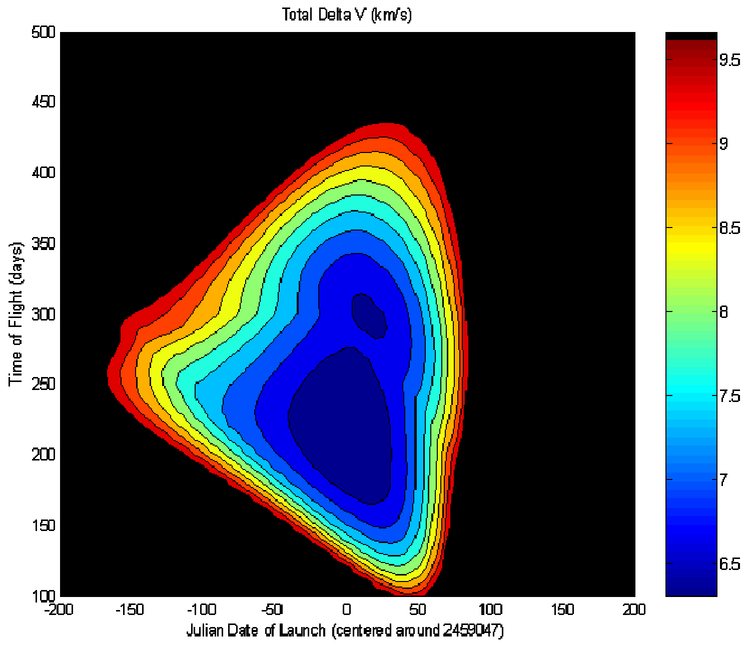


Figure 16-6 MHV Delta-V

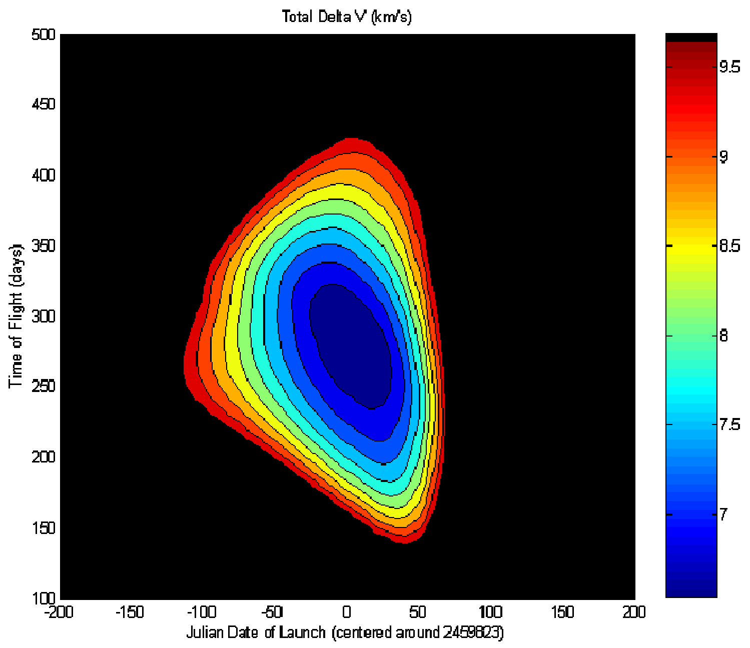


Figure 16-7 MHV Delta-V

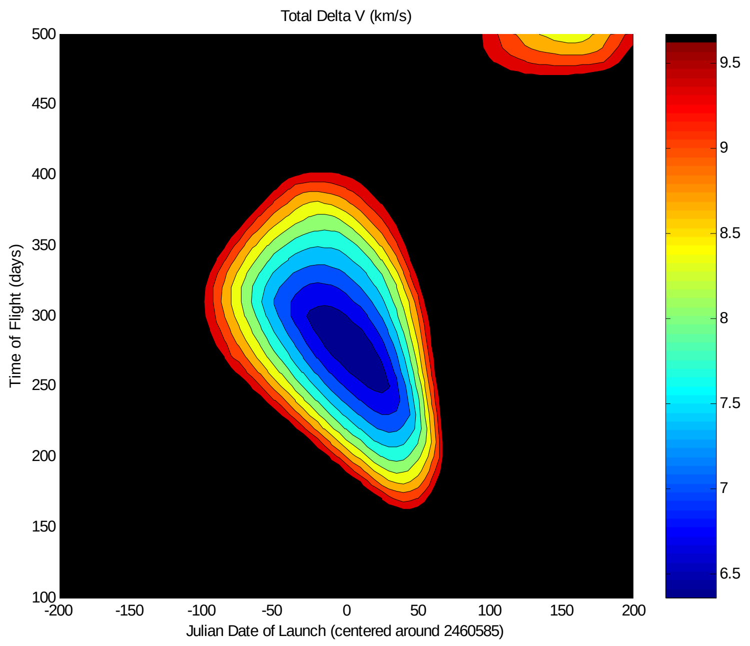


Figure 16-8 MHV Delta-V

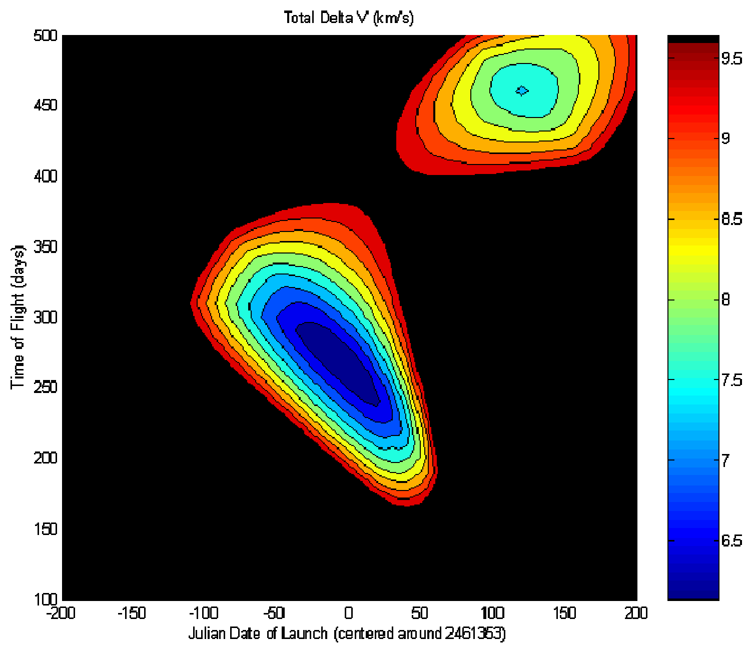


Figure 16-9 MHV Delta-V

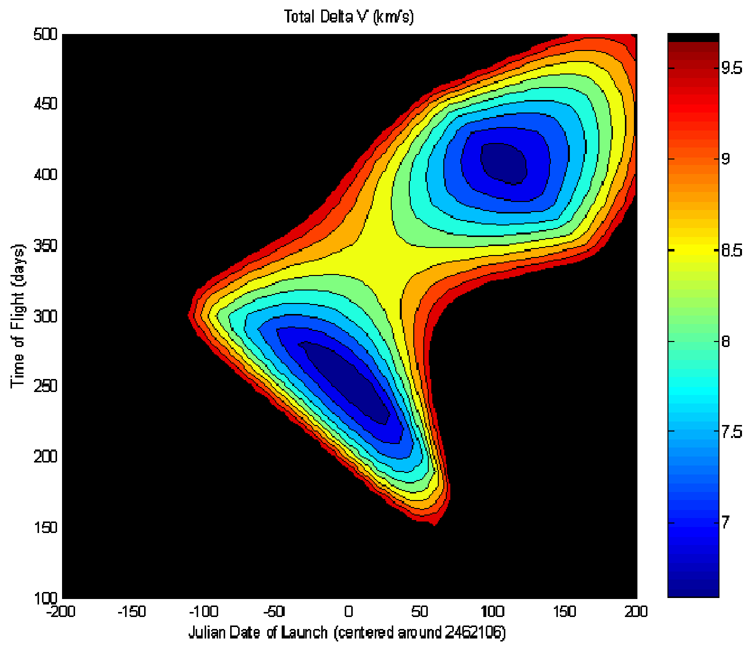


Figure 16-10 MHV Delta-V

16.4.2 MATLAB Code

The following sections are the trajectory analysis codes developed by Meredith Evans, Kimberly Mrozek, and the author.

16.4.2.1 Main Function

The purpose of this MATLAB m-file is to run through a set of Julian Dates and calculate the ΔV, propellant mass, and other pertinent information for the MHV trajectory.

```
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This is an improved version of the code that calculates the
%optimal delta V (and TOF) for a given range of launch dates
%for the HAB trajectory.  The optimization process was improved so that the
%code now runs much faster.
%This code calls on "orbit3" and runs for a specified range of julian
%dates.
clear all; close all; clc; warning off;

tic

format long
counter=0;
jdate_vector = 2459047+[0,1,3,5,7,10:10:90];
%jdate_vector = 2457300:10:2457600;
for jdate= jdate_vector
    fprintf('%5.2f%% done...\n', (jdate-jdate_vector(1))/(jdate_vector(end)-jdate_vector(1))*100);
    counter=counter+1;
    best_TOF = fminbnd('orbit3_minimize', 100, 400, optimset('TolX', .5), jdate);
    output=orbit3(best_TOF, jdate);
    %     [jdateofcounter, TA, tof_day, delta_v1, delta_v2, deltav_tot]=output
    jdateofcounter(counter)=output(1);
    TA(counter)=output(2);
    tof_day(counter)=output(3);
    delta_v1(counter)=output(4);
    delta_v2(counter)=output(5);
    deltav_tot(counter)=output(6);
    a_a(counter)=output(7);
    deltav_inc_change(counter)=output(8);
    mass_prop_A(counter)=output(9);
    time_burn_A(counter)=output(10);
    mass_prop_inc(counter)=output(11);
    time_burn_inc(counter)=output(12);
    mass_prop_D(counter)=output(13);
    time_burn_D(counter)=output(14);
    mass_prop_total(counter)=output(15);
    v_infinity_D(counter)=output(16);
    v_infinity_A(counter)=output(17);
end

  TA=TA';
  tof_day=tof_day';
  delta_v1=delta_v1';
  delta_v2=delta_v2';
  deltav_tot=deltav_tot';
  a_a=a_a';
  deltav_inc_change=deltav_inc_change';
  mass_prop_A = mass_prop_A';
  time_burn_A = time_burn_A';
  mass_prop_inc = mass_prop_inc';
  time_burn_inc = time_burn_inc';
  mass_prop_D = mass_prop_D';
  time_burn_D = time_burn_D';
  mass_prop_total = mass_prop_total';
  v_infinity_D = v_infinity_D';
  v_infinity_A = v_infinity_A';

%%%%%%%%%
%Output
%%%%%%%%%
dataoutput=[delta_v1 mass_prop_D time_burn_D deltav_inc_change mass_prop_inc...
        time_burn_inc  delta_v2  mass_prop_A  time_burn_A  deltav_tot  mass_prop_total  v_infinity_D
v_infinity_A TA a_a tof_day];

save(['window_', num2str(jdateofcounter(1)), '.txt'],'dataoutput','-ASCII')

toc

subplot(2,1,1); plot(jdateofcounter-jdateofcounter(1), tof_day); grid on
ylabel('TOF (day)')
title(num2str(jdateofcounter(1)))
subplot(2,1,2); plot(jdateofcounter-jdateofcounter(1), deltav_tot); grid on
ylabel('Total delta V')
xlabel('Julian Date (Days after first date)')
```

16.4.2.2 Orbit Mechanics Core Code

This is the m-file where all the orbital mechanics calculations are performed. The input is a Julian date of launch, and a time of flight. The outputs are transfer angle, ΔV s for departure, arrival, plane change, propellant masses, and ΔV_∞ information.

```
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This code calls on "planet data" and outputs the delta_V, and other info
%from a given time of flight and a specified Julian date.


function out=orbit3(TOF, jdate)

counter=0;
%for TOF=180:200
    counter=counter+1;
    [r1_vector,   v1_vector,   theta_starE,   r2_vector_junk,   v2_vector_junk,   theta_star_junk]
=planet_data(jdate);
    [RE_vector_junk,   VE_vector_junk,   theta_star_junk,   r2_vector,   v2_vector,   theta_starM]
=planet_data(jdate+TOF);
    r2_vector_actual=r2_vector;
    r1_vector(3)=0;
    r2_vector(3)=0;
    v1_vector(3)=0;
    v2_vector(3)=0;
    r1=norm(r1_vector); r2=norm(r2_vector);
    v1=norm(v1_vector); v2=norm(v2_vector);
    c_vector=r2_vector - r1_vector;
    c=norm(c_vector);
    s=0.5*(c+r1+r2);

    r1_hat=r1_vector/r1;
    r2_hat=r2_vector/r2;
    h=cross(r1_vector,r2_vector);
    del=acos((dot(r1_vector,r2_vector))/(r1*r2));
    if h(3)<0
        h=-h;
        TA(counter)=(2*pi)-del;
    else
        h=h;
        TA(counter)=del;
    end
    TA(counter);
    muE=398600.485043; mu=1.3271244E+11; muM=4.28282868534e+04;
    h_mag=norm(h);
    h_hat=h/h_mag;

    %Earth transformation matrix
    theta_hat=cross(h_hat,r1_hat);
    trans_matrix=[r1_hat(1)    theta_hat(1)    h_hat(1);r1_hat(2)    theta_hat(2)    h_hat(2);r1_hat(3)
theta_hat(3) h_hat(3)];
    inc=acos(h_hat(3));
    omega1=acos(h_hat(2)/-sin(inc));
    omega2=asin(h_hat(1)/sin(inc));
    theta1=acos(theta_hat(3)/sin(inc));
    theta2=asin(r1_hat(3)/sin(inc));

    %Mars transformation matrix
    theta_hat_M=cross(h_hat,r2_hat);
    trans_matrix_M=[r2_hat(1)   theta_hat_M(1)   h_hat(1);r2_hat(2)   theta_hat_M(2)   h_hat(2);r2_hat(3)
theta_hat_M(3) h_hat(3)];

    %inclination change
    r2_hat_actual=r2_vector_actual/norm(r2_vector_actual);
    r1_hat_new=cross(r2_hat_actual,h_hat);
    h_new=cross(r1_hat_new,r2_hat_actual);
    h_new_hat=h_new/norm(h_new);
    inclination=acos(h_new_hat(3));
    incli=inclination*180/pi;

    %%%%%%%%%%%%%%%%%%%%%
    %Minimum Conditions
    %%%%%%%%%%%%%%%%%%%%%
    a_min=0.5*s;                            %Minimum semi-major axis [Km]
    energy_min=-mu/(2*a_min);               %Energy associated with a_min [Km^2/sec^2]
    alpha_min=pi;                           %[rad]
    beta_min=2*asin(sqrt((s-c)/(2*a_min))); %[rad]
    P_min=(4*a_min*(s-r1)*(s-r2)*(sin(0.5*(alpha_min+beta_min)))^2)/c^2;
    e_min=sqrt(1-P_min/a_min);               %Eccentricity associated with a_min
    TOF_min=((a_min^1.5)/sqrt(mu))*((alpha_min-beta_min)-(sin(alpha_min)-sin(beta_min)));

    %%%%%%%%%%%%
    %Finding "a"
    %%%%%%%%%%%%
    TA(counter)=TA(counter)*180/pi;

    if TA(counter)<180
        TOF_par=(sqrt(2/mu)*(s^1.5-(s-c)^1.5))/3;   %Type 1
    else
        TOF_par=(sqrt(2/mu)*(s^1.5+(s-c)^1.5))/3;   %Type 2
    end

    TOF_Vector=24*3600*[TOF];
    a0=a_min;

    for j=1:length(TOF_Vector),
        TOFh=TOF_Vector(j);
        if TA(counter)<180
            if (TOFh<TOF_par)
                fprintf('\n\nHyperbola-1H\n\n');
                a0h=-a0;
                FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
(2*asinh(sqrt(s/(2*abs(a))))))-(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                a(j)=fsolve(FUN,a0h,[],s,TOFh,c,mu);
            else
                if (TOFh<TOF_min)
                    fprintf('\n\Ellipse-1A\n\n');
                    FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
sin(2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN2,a0,[],s,TOFh,c,mu);
                else
                    fprintf('\n\Ellipse-1B\n\n');
                    FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN3,a0,[],s,TOFh,c,mu);
                end
            end
        else
            if (TOFh<TOF_par)
                fprintf('\n\nHyperbola-2H\n\n');
                a0h=-a0;
                FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
(2*asinh(sqrt(s/(2*abs(a))))))+(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                a(j)=fsolve(FUN,a0h,[],s,TOFh,c,mu);
            else
                if (TOFh<TOF_min)
                    fprintf('\n\Ellipse-2A\n\n');
                    FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
sin(2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN2,a0,[],s,TOFh,c,mu);
                else
                    fprintf('\n\Ellipse-2B\n\n');
                    FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN3,a0,[],s,TOFh,c,mu);
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%
    %Type 1 parameteres
    %%%%%%%%%%%%%%%%%%%
    if TA(counter)<180
        if (TOFh<TOF_par)
            fprintf('\n\nType-1H\n\n');
            alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
            betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
            P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
            P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

            if P_plus>P_minus
                P=P_plus;
            else
                P=P_minus;
            end
        else
            if (TOFh<TOF_min)
                fprintf('\n\nType-1A\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                fprintf('\n\nType-1B\n\n')
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                if P_plus<P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            end
        end
    else
        if (TOFh<TOF_par)
            fprintf('\n\nType-2H\n\n');
            alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
            betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
            P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
            P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

            if P_plus<P_minus
                P=P_plus;
            else
                P=P_minus;
            end
        else
            if (TOFh<TOF_min)
                fprintf('\n\nType-2A\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus<P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                fprintf('\n\nType-2B\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            end
        end
    end
```

```
TA(counter)=TA(counter)*pi/180;

%time of flight check against stk values
tof_day(counter)=TOF;
a_a(counter)=a;

%%%%%%%%%%%%%%%%%%%%%%%%%
%Transfer Characteristics
%%%%%%%%%%%%%%%%%%%%%%%%%

%Orbit
r1_per=200+6378.14;
r2_per=500+3397.00;
e=sqrt(1-(P/a_a(counter)));
vD=sqrt(2*((mu/r1)-(mu/(2*a_a(counter)))));
vA=sqrt(2*((mu/r2)-(mu/(2*a_a(counter)))));
rD=r1; rA=r2;

%delta_V for inclination change
theta_star_halfway=pi/2;
r_halfway=a_a(counter)*(1-e);
velocity_at_halfway=sqrt(2*((mu/r_halfway)-(mu/(2*a_a(counter)))));
deltav_inc_change(counter)=2*velocity_at_halfway*sin(inclination/2);

%Angle
theta_star_D=acos((1/e)*((P/r1-1)));
theta_star_A=acos((1/e)*((P/r2-1)));

if theta_star_A-theta_star_D==TA(counter)
    theta_star_A_new=theta_star_A;
    theta_star_D_new=theta_star_D;
elseif -theta_star_A-theta_star_D==TA(counter)
    theta_star_A_new=-theta_star_A;
    theta_star_D_new=theta_star_D;
elseif -theta_star_A+theta_star_D==TA(counter)
    theta_star_A_new=-theta_star_A;
    theta_star_D_new=-theta_star_D;
else
    theta_star_A_new=theta_star_A;
    theta_star_D_new=-theta_star_D;
end

theta_star_D;
theta_star_A;

gamma_D=acos(sqrt(mu*P)/(r1*vD))*sign(theta_star_D_new);
gamma_A=acos(sqrt(mu*P)/(r2*vA))*sign(theta_star_A_new);

%r, h, theta relationship
vD_vector_1=[vD*sin(gamma_D) vD*cos(gamma_D) 0];
vD_vector= vD_vector_1*trans_matrix';
v_infinity_dep=vD_vector'-v1_vector;
v_infinity_mag(counter)=norm(v_infinity_dep);
```

```
    delta_vD(counter)=sqrt(v_infinity_mag(counter)^2+((2*muE)/r1_per)) - sqrt(muE/r1_per);
    v_infinity_D(counter)=v_infinity_mag(counter);

    vA_vector_1=[vA*sin(gamma_A) vA*cos(gamma_A) 0];
    vA_vector= vA_vector_1*trans_matrix_M';
    v_infinity_arr=v2_vector-vA_vector';
    v_infinity_mag1(counter)=norm(v_infinity_arr);
    delta_vA(counter)=sqrt(v_infinity_mag1(counter)^2+((2*muM)/r2_per)) - sqrt(muM/r2_per);
    v_infinity_A(counter)=v_infinity_mag1(counter);

    %TOTAL
    deltav_tot(counter)=abs(delta_vD(counter))+abs(delta_vA(counter))+abs(deltav_inc_change(counter));

    %Burn Time Analysis

    g=9.81;             %[m/s^2]
    Isp=1007.1;           %Assumed for now
    mdot=2.13;           %based on current engine [kg/s]
    mass_f=76862.21805; %Current MHV mass [kg]

    mass_prop_A(counter)=mass_f*exp(delta_vA(counter)*10^3/(Isp*g)) - mass_f;
    time_burn_A(counter)=mass_prop_A(counter)/mdot;

    mass_prop_inc(counter)=(mass_prop_A(counter)                                              +
mass_f)*exp(deltav_inc_change(counter)*10^3/(Isp*g)) - (mass_prop_A(counter) + mass_f);
    time_burn_inc(counter)=mass_prop_inc(counter)/mdot;

    mass_prop_D(counter)=(mass_prop_inc(counter)           +           mass_prop_A(counter)          +
mass_f)*exp(delta_vD(counter)*10^3/(Isp*g))   -   (mass_prop_inc(counter)   +   mass_prop_A(counter)   +
mass_f);
    time_burn_D(counter)=mass_prop_D(counter)/mdot;

    mass_prop_total(counter)=mass_prop_A(counter)+mass_prop_inc(counter)+mass_prop_D(counter);
%end

best_counter=find(deltav_tot==min(deltav_tot));

out=[jdate,          TA(best_counter),          tof_day(best_counter),          delta_vD(best_counter),
delta_vA(best_counter),...
        deltav_tot(best_counter), a_a(counter),deltav_inc_change(counter), mass_prop_A(counter),...
        time_burn_A(counter),mass_prop_inc(counter),time_burn_inc(counter),mass_prop_D(counter),...
        time_burn_D(counter),mass_prop_total(counter) v_infinity_D(counter) v_infinity_A(counter)];
```

16.4.2.3 Orbit Mechanics Optimization Code

This code is similar to the core optimization code, however, it is modified to be called by the MATLAB function `fminbnd` to find the optimal time of flight for a given launch date.

```
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This code calls on "planet data" and is used to find the best delta_V and TOF
%for a specified Julian date using a built in MATLAB optimizer, fminbnd.
```

```
function out=orbit3_minimize(TOF, jdate)

counter=0;
%for TOF=180:200
    counter=counter+1;
    [r1_vector,    v1_vector,    theta_starE,    r2_vector_junk,    v2_vector_junk,    theta_star_junk]
=planet_data(jdate);
    [RE_vector_junk,    VE_vector_junk,    theta_star_junk,    r2_vector,    v2_vector,    theta_starM]
=planet_data(jdate+TOF);
    r2_vector_actual=r2_vector;
    r1_vector(3)=0;
    r2_vector(3)=0;
    v1_vector(3)=0;
    v2_vector(3)=0;
    r1=norm(r1_vector); r2=norm(r2_vector);
    v1=norm(v1_vector); v2=norm(v2_vector);
    c_vector=r2_vector - r1_vector;
    c=norm(c_vector);
    s=0.5*(c+r1+r2);

    r1_hat=r1_vector/r1;
    r2_hat=r2_vector/r2;
    h=cross(r1_vector,r2_vector);
    del=acos((dot(r1_vector,r2_vector))/(r1*r2));
    if h(3)<0
        h=-h;
        TA(counter)=(2*pi)-del;
    else
        h=h;
        TA(counter)=del;
    end
    TA(counter);
    muE=398600.485043; mu=1.3271244E+11; muM=4.28282868534e+04;
    h_mag=norm(h);
    h_hat=h/h_mag;

    %Earth transformation matrix
    theta_hat=cross(h_hat,r1_hat);
    trans_matrix=[r1_hat(1)    theta_hat(1)    h_hat(1);r1_hat(2)    theta_hat(2)    h_hat(2);r1_hat(3)
theta_hat(3) h_hat(3)];
    inc=acos(h_hat(3));
    omega1=acos(h_hat(2)/-sin(inc));
    omega2=asin(h_hat(1)/sin(inc));
    theta1=acos(theta_hat(3)/sin(inc));
    theta2=asin(r1_hat(3)/sin(inc));

    %Mars transformation matrix
    theta_hat_M=cross(h_hat,r2_hat);
    trans_matrix_M=[r2_hat(1)   theta_hat_M(1)   h_hat(1);r2_hat(2)   theta_hat_M(2)   h_hat(2);r2_hat(3)
theta_hat_M(3) h_hat(3)];

    %inclination change
    r2_hat_actual=r2_vector_actual/norm(r2_vector_actual);
```

```
        r1_hat_new=cross(r2_hat_actual,h_hat);
        h_new=cross(r1_hat_new,r2_hat_actual);
        h_new_hat=h_new/norm(h_new);
        inclination=acos(h_new_hat(3));
        incli=inclination*180/pi;

        %%%%%%%%%%%%%%%%%%%%%
        %Minimum Conditions
        %%%%%%%%%%%%%%%%%%%%%

        a_min=0.5*s;                            %Minimum semi-major axis [Km]
        energy_min=-mu/(2*a_min);               %Energy associated with a_min [Km^2/sec^2]
        alpha_min=pi;                           %[rad]
        beta_min=2*asin(sqrt((s-c)/(2*a_min))); %[rad]
        P_min=(4*a_min*(s-r1)*(s-r2)*(sin(0.5*(alpha_min+beta_min)))^2)/c^2;
        e_min=sqrt(1-P_min/a_min);               %Eccentricity associated with a_min
        TOF_min=((a_min^1.5)/sqrt(mu))*((alpha_min-beta_min)-(sin(alpha_min)-sin(beta_min)));

        %%%%%%%%%%%%
        %Finding "a"
        %%%%%%%%%%%%
        TA(counter)=TA(counter)*180/pi;

        if TA(counter)<180
            TOF_par=(sqrt(2/mu)*(s^1.5-(s-c)^1.5))/3;   %Type 1
        else
            TOF_par=(sqrt(2/mu)*(s^1.5+(s-c)^1.5))/3;   %Type 2
        end

        TOF_Vector=24*3600*[TOF];
        a0=a_min;

        for j=1:length(TOF_Vector),
            TOFh=TOF_Vector(j);
            if TA(counter)<180
                if (TOFh<TOF_par)
                    %fprintf('\n\nHyperbola-1H\n\n');
                    a0h=-a0;
                    FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
    (2*asinh(sqrt(s/(2*abs(a))))))-(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
    c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN,a0h,optimset('Display','off'),s,TOFh,c,mu);
                else
                    if (TOFh<TOF_min)
                    %    fprintf('\n\Ellipse-1A\n\n');
                        FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
    sin(2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN2,a0,optimset('Display','off'),s,TOFh,c,mu);
                    else
                    %    fprintf('\n\Ellipse-1B\n\n');
                        FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
    2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN3,a0,optimset('Display','off'),s,TOFh,c,mu);
                    end
                end
            else
                if (TOFh<TOF_par)
                    %fprintf('\n\nHyperbola-2H\n\n');
                    a0h=-a0;
                    FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
    (2*asinh(sqrt(s/(2*abs(a))))))+(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
    c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN,a0h,optimset('Display','off'),s,TOFh,c,mu);
                else
                    if (TOFh<TOF_min)
                        %fprintf('\n\Ellipse-2A\n\n');
                        FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
    sin(2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN2,a0,optimset('Display','off'),s,TOFh,c,mu);
                    else
                        %fprintf('\n\Ellipse-2B\n\n');
                        FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
    2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN3,a0,optimset('Display','off'),s,TOFh,c,mu);
                    end
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%
        %Type 1 parameteres
        %%%%%%%%%%%%%%%%%%%
        if TA(counter)<180
            if (TOFh<TOF_par)
                %fprintf('\n\nType-1H\n\n');
                alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                if (TOFh<TOF_min)
                    %fprintf('\n\nType-1A\n\n');
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
```

```
                    P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                    if P_plus>P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                else
                    %fprintf('\n\nType-1B\n\n')
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                    P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                    if P_plus<P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                end
            end
        else
            if (TOFh<TOF_par)
                %fprintf('\n\nType-2H\n\n');
                alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus<P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                if (TOFh<TOF_min)
                    %fprintf('\n\nType-2A\n\n');
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                    P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                    if P_plus<P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                else
                    %fprintf('\n\nType-2B\n\n');
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                    P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                    if P_plus>P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                end
            end
        end
        TA(counter)=TA(counter)*pi/180;

        %time of flight check against stk values
        tof_day(counter)=TOF;
        a_a(counter)=a;

        %%%%%%%%%%%%%%%%%%%%%%%%%
        %Transfer Characteristics
        %%%%%%%%%%%%%%%%%%%%%%%%%

        %Orbit
        r1_per=200+6378.14;
        r2_per=500+3397.00;
        e=sqrt(1-(P/a_a(counter)));
        vD=sqrt(2*((mu/r1)-(mu/(2*a_a(counter)))));
        vA=sqrt(2*((mu/r2)-(mu/(2*a_a(counter)))));
        rD=r1; rA=r2;

        %delta_V for inclination change
        theta_star_halfway=pi/2;
        r_halfway=a_a(counter)*(1-e);
        velocity_at_halfway=sqrt(2*((mu/r_halfway)-(mu/(2*a_a(counter)))));
        deltav_inc_change(counter)=2*velocity_at_halfway*sin(inclination/2);

        %Angle
        theta_star_D=acos((1/e)*((P/r1-1)));
        theta_star_A=acos((1/e)*((P/r2-1)));

        if theta_star_A-theta_star_D==TA(counter)
            theta_star_A_new=theta_star_A;
            theta_star_D_new=theta_star_D;
        elseif -theta_star_A-theta_star_D==TA(counter)
            theta_star_A_new=-theta_star_A;
            theta_star_D_new=theta_star_D;
        elseif -theta_star_A+theta_star_D==TA(counter)
            theta_star_A_new=-theta_star_A;
            theta_star_D_new=-theta_star_D;
        else
            theta_star_A_new=theta_star_A;
            theta_star_D_new=-theta_star_D;
        end

        theta_star_D;
```

```
    theta_star_A;

    gamma_D=acos(sqrt(mu*P)/(r1*vD))*sign(theta_star_D_new);
    gamma_A=acos(sqrt(mu*P)/(r2*vA))*sign(theta_star_A_new);

    %r, h, theta relationship
    vD_vector_1=[vD*sin(gamma_D) vD*cos(gamma_D) 0];
    vD_vector= vD_vector_1*trans_matrix';
    v_infinity_dep=vD_vector'-v1_vector;
    v_infinity_mag(counter)=norm(v_infinity_dep);
    delta_vD(counter)=sqrt(v_infinity_mag(counter)^2+((2*muE)/r1_per)) - sqrt(muE/r1_per);
    v_infinity_D(counter)=v_infinity_mag(counter);

    vA_vector_1=[vA*sin(gamma_A) vA*cos(gamma_A) 0];
    vA_vector= vA_vector_1*trans_matrix_M';
    v_infinity_arr=v2_vector-vA_vector';
    v_infinity_mag1(counter)=norm(v_infinity_arr);
    delta_vA(counter)=sqrt(v_infinity_mag1(counter)^2+((2*muM)/r2_per)) - sqrt(muM/r2_per);
    v_infinity_A(counter)=v_infinity_mag1(counter);

    %TOTAL
    deltav_tot(counter)=abs(delta_vD(counter))+abs(delta_vA(counter))+abs(deltav_inc_change(counter));

    %Burn Time Analysis

    g=9.81;             %[m/s^2]
    Isp=1007.1;           %Assumed for now
    mdot=2.13;           %based on current engine [kg/s]
    mass_f=76862.21805; %Current MHV mass [kg]

    mass_prop_A(counter)=mass_f*exp(delta_vA(counter)*10^3/(Isp*g)) - mass_f;
    time_burn_A(counter)=mass_prop_A(counter)/mdot;

    mass_prop_inc(counter)=(mass_prop_A(counter)                                                  +
mass_f)*exp(deltav_inc_change(counter)*10^3/(Isp*g)) - (mass_prop_A(counter) + mass_f);
    time_burn_inc(counter)=mass_prop_inc(counter)/mdot;

    mass_prop_D(counter)=(mass_prop_inc(counter)           +           mass_prop_A(counter)           +
mass_f)*exp(delta_vD(counter)*10^3/(Isp*g))   -   (mass_prop_inc(counter)   +   mass_prop_A(counter)   +
mass_f);
    time_burn_D(counter)=mass_prop_D(counter)/mdot;

    mass_prop_total(counter)=mass_prop_A(counter)+mass_prop_inc(counter)+mass_prop_D(counter);
%end

%best_counter=find(deltav_tot==min(deltav_tot));

out= deltav_tot;
```

16.4.2.4 Planetary Position and Velocity Code

This function outputs the heliocentric position and velocity of Earth and Mars for a given Julian Date. The input data is a report from Satellite Tool Kit (STK) giving the position and velocities of the planets in the Sun Centered Mean Ecliptic J2000 frame at 1-day intervals. The code then interpolates this information as needed.

```
%This function outputs the heliocentric position and velocity and the true anomaly
% of Earth and Mars for a given Julian Date.
%Positions are in km, and velocities are in km/s, theta_stars are in radians
%
%The planetary data is taken from STK in the Sun Centered Mean Ecliptic of J2000 frame
%All output vectors are column vectors
%This function needs the binary data files created by running "make_binary_data" - earth_data.mat and
mars_data.mat
%(Using the binary files just makes the function run a bit faster)
%
%example use: [pos_earth, vel_earth, thst_earth, pos_mars, vel_mars, thst_mars] = planet_data(2456659)
% will return the data for noon on January 1st, 2014
%
%Eric Gustafson - 2/13/2005

function [pos_earth, vel_earth, thst_earth, pos_mars, vel_mars, thst_mars] = planet_data(JD_interp)

if JD_interp<2451758 | JD_interp>2462867
    disp 'The JD being input to planet_data is out of range!'
    return
end

load earth_data
load mars_data

%variables loaded into workspace are:
%JD ,x_earth, y_earth, z_earth, x_vel_earth, y_vel_earth, z_vel_earth,
%x_mars, y_mars, z_mars, x_vel_mars, y_vel_mars, z_vel_mars

%store all the STK data in an array, that will be used to interpolate
pos_earth_vector = [x_earth, y_earth, z_earth];
vel_earth_vector = [x_vel_earth, y_vel_earth, z_vel_earth];
pos_mars_vector = [x_mars, y_mars, z_mars];
vel_mars_vector = [x_vel_mars, y_vel_mars, z_vel_mars];
%interpolate to find requested Julian date
pos_earth = interp1(JD, pos_earth_vector, JD_interp)';
vel_earth = interp1(JD, vel_earth_vector, JD_interp)';
pos_mars = interp1(JD, pos_mars_vector, JD_interp)';
vel_mars = interp1(JD, vel_mars_vector, JD_interp)';

a_earth = 1.49597807e8; %km
e_earth = .01670545;
p_earth = a_earth*(1-e_earth^2);
thst_earth = 180/pi*acos(1/e_earth*(p_earth/norm(pos_earth) - 1)) * sign(dot(pos_earth,vel_earth)) ;
%                                                             ---------- quad check --------

a_mars = 2.27936636e8; %km
```

```
e_mars = .09341233;
p_mars = a_mars*(1-e_mars^2);
thst_mars = 180/pi*acos(1/e_mars*(p_mars/norm(pos_mars) - 1)) * sign(dot(pos_mars,vel_mars)) ;
%                                                         -------- quad check --------
```

16.4.2.5 Launch Window Visualization Code

This function uses the above functions and programs to create a contour plot of the ΔV needed for a given launch window. The x-axis is the launch date, the y-axis is the trip time, and the quantity being contoured is the ΔV to complete the mission at the specified launch date and time of flight.

```
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This code calls on "orbit3" and runs for a specified range of julian
%dates. It then spits out the total delta_v and tof.
clear all; close all; clc;

tic

format long
TOF_counter=0;
jdate_counter=0;
jdate_center = 2459047
jdate_vector = jdate_center-200:5:jdate_center+200;  %step by 5
TOF_vector = 100 : 10 : 500;   %step by 10

for jdate=jdate_vector
    jdate_counter = jdate_counter+1;
    TOF_counter=0;
    for TOF = TOF_vector
        fprintf('%5.2f%% done...\n', (jdate-jdate_vector(1))/(jdate_vector(end)-jdate_vector(1))*100);
        TOF_counter=TOF_counter+1;
        %best_TOF = fminbnd('orbit3_minimize', 100, 400, optimset('TolX', .5), jdate);
        output=orbit3(TOF, jdate);
        %     [jdateofcounter, TA, tof_day, delta_v1, delta_v2, deltav_tot]=output
        %data(jdate_counter, TOF_counter)
        jdateofcounter(jdate_counter, TOF_counter)=output(1);
        TA(jdate_counter, TOF_counter)=output(2);
        tof_day(jdate_counter, TOF_counter)=output(3);
        delta_v1(jdate_counter, TOF_counter)=output(4);
        delta_v2(jdate_counter, TOF_counter)=output(5);
        deltav_tot(jdate_counter, TOF_counter)=output(6);
        a_a(jdate_counter, TOF_counter)=output(7);
        deltav_inc_change(jdate_counter, TOF_counter)=output(8);
        mass_prop_A(jdate_counter, TOF_counter)=output(9);
        time_burn_A(jdate_counter, TOF_counter)=output(10);
        mass_prop_inc(jdate_counter, TOF_counter)=output(11);
        time_burn_inc(jdate_counter, TOF_counter)=output(12);
        mass_prop_D(jdate_counter, TOF_counter)=output(13);
        time_burn_D(jdate_counter, TOF_counter)=output(14);
        mass_prop_total(jdate_counter, TOF_counter)=output(15);
        v_infinity_D(jdate_counter, TOF_counter)=output(16);
        v_infinity_A(jdate_counter, TOF_counter)=output(17);
    end
```

```
end

  TA=TA';
  tof_day=tof_day';
  delta_v1=delta_v1';
  delta_v2=delta_v2';
  deltav_tot=deltav_tot';
  a_a=a_a';
  deltav_inc_change=deltav_inc_change';
  mass_prop_A = mass_prop_A';
  time_burn_A = time_burn_A';
  mass_prop_inc = mass_prop_inc';
  time_burn_inc = time_burn_inc';
  mass_prop_D = mass_prop_D';
  time_burn_D = time_burn_D';
  mass_prop_total = mass_prop_total';
  v_infinity_D = v_infinity_D';
  v_infinity_A = v_infinity_A';

%%%%%%%%%
%Output
%%%%%%%%%
% dataoutput=[delta_v1 mass_prop_D time_burn_D deltav_inc_change mass_prop_inc...
%             time_burn_inc delta_v2 mass_prop_A time_burn_A deltav_tot mass_prop_total v_infinity_D
v_infinity_A TA a_a];
% save('RUN14','dataoutput','-ASCII')

%limit deltav:
deltav_tot(deltav_tot>10) = 10;
contourf(jdate_vector-jdate_center, TOF_vector, deltav_tot)
colorbar
xlabel(['Julian Date of Launch (centered around ', num2str(jdate_center), ')'])
ylabel('Time of Flight (days)')
title('Total Delta V (km/s)')

toc
```

17 Jayleen Guttromson Appendix

1.1 Human G-load Tolerances

Author: Jayleen Guttromson

Historical data for human g-load tolerances varies for individuals depending on the scenario and position in which they experience the acceleration. G-loads define the measurement of acceleration in multiples of the acceleration caused by Earth's gravity. The most important factor for tolerating the acceleration depends on an individual's posture and their position relative to the direction of the acceleration. The best position to minimize g-loads is for the astronaut's entire body to face the direction of acceleration. We incorporate this concept in the Ascent and Recovery Vehicle (ARV) and Mars Lander Vehicle (MLV) interior layouts by locating the crewmember's seats perpendicular to the direction of acceleration.

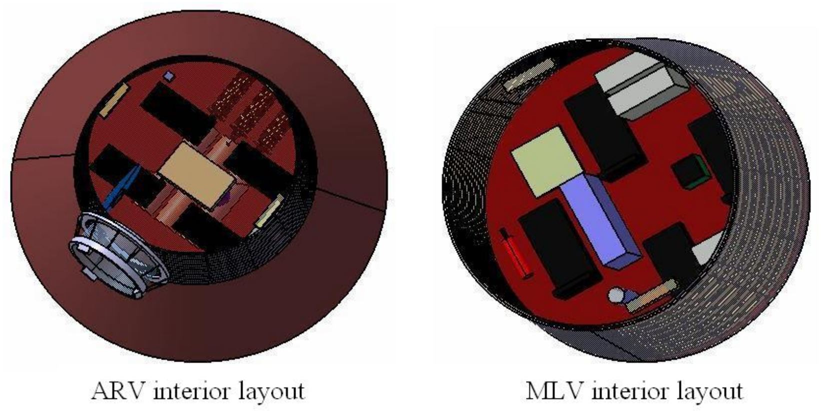


Figure 1-1 Interior layouts of the ARV and MLV showing seat placement (Chris Cunha)

On rollercoaster rides we experience approximately 2 G's and pilots in jet fighters sometimes reach 7 G's. Shuttle crewmembers experience approximately 3G's for 17 minutes and reach a peak of 4 G's when the solid rocket boosters ignite [1]. As a comparison, the Soyuz reaches a maximum of 4 G's during the nine minute launch duration and approximately 6.5 G's during re-entry [2]. NASA-STD-3000 recommends ±4 gx (eyeballs in-out), ±1 gy (eyeballs left-right), ±0.5 gz (eyeballs up-down) loads for emergency entry scenarios [3]. In the early years of space flight, crewmembers were momentarily exposed to as much as 11 G's, using military rockets and parachute landing systems. In addition, SpaceAdventures.com proposes the g-load limit of human survival at 40 G's for less than 0.5 seconds [4].

Current research at the United States Air Force School of Aerospace Medicine (USAFSAM) executes three kinds of centrifuge runs to evaluate G-tolerance on unprotected subjects. Figure 1-2 combines data from the three runs: GOR (gradual onset run) conducted at 0.1 G/sec, ROR (rapid onset run) conducted at 1 G/sec, and VHOG (very high onset G) conducted at 6 G/sec. Table 1-1 presents the ROR G-tolerances of 1000 Subjects [5]. For an unprotected subject, high G-forces cause a decreased blood flow to the brain and loss of consciousness, referred to as GLC or GLOC.

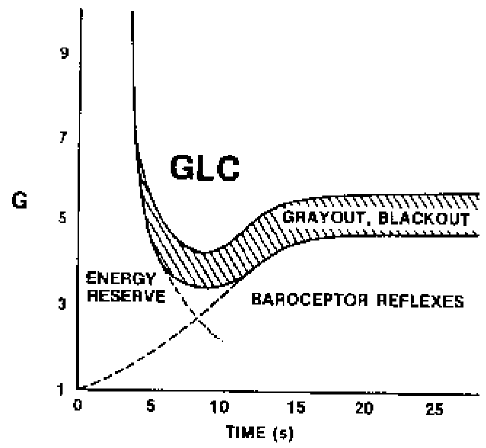


Figure 1-2 G-Time Tolerance Curve (USAFSAM [5])

Table 1-1 USAFSAM Rapid Onset Run (ROR) G Tolerances of 1000 Subjects (1 G/sec onset rate) (USAFSAM [5])

Criterion	Mean Threshold (G units)	Standard Deviation (G units)	Range (G units)
Grayout or loss of peripheral vision	4.1	0.7	2.2-7.1
Blackout	4.8	0.8	2.7-7.8
Unconsciousness	5.4	0.9	3.0-8.4

From these references, we propose the human tolerances for launch and entry maneuvers at 3-4 G's with a maximum of 6 G's for the ARV and the MLV. In addition, we require crewmembers wear g-suits during these mission phases to restrict the flow of blood away from the brain. Our recommendation agrees with published exploration studies by universities and in industry, such as the European Space Agency's Study on the Survivability and Adaptation of Humans to Long-Duration Interplanetary and Planetary Environments [6].

References

[1] Larson, Wiley J., and Linda K. Pranke. Human Spaceflight Mission Analysis and Design. New York: McGraw-Hill Companies, Inc. 1999.

[2] "Experience." Spaceflight Training. Space Adventures, Ltd. 2005. <http://sa.qa.elro.com/training/centrifuge/experience>

[3] Man-Systems Integration Standards. NASA-STD-3000. Revision B. NASA Johnson Space Center. July 1995. Last updated: 10 Mar. 2005. <http://msis.jsc.nasa.gov/default.htm>

[4] "G-loads." Glossary. SpaceAdventures.com. Space Adventures, Ltd. 2005. <http://www.spaceadventures.com/media/info/glossary>

[5] Eldridge, Louis D., M.D., M.P.H. and Susan E. Northrup, M.D., M.P.H., "Chapter 4 Effects of Acceleration." USAF Flight Surgeon's Guide. 20 Jan. 2005 <http://wwwsam.brooks.af.mil/af/files/fsguide/HTML/Chapter_04.html>

[6] "Definition of Reference Scenarios for a European Participation in Human Exploration and Estimation of the Life Sciences and Life Support Requirements." Study on the Survivability and Adaptation of Humans to Long-Duration Interplanetary and Planetary Environments. HUMEX-TN-001 Draft. Prepared by B. Comet, M. Reichert, and G. Horneck. 9 June 2000. < http://www.estec.esa.nl/ecls/melissa/attachments/humex/tn1.pdf>

Bibliography

[1] Comet, B. and Reichert, M. "HUMEX-TN-001 Draft." European Space Agency. 22 Jan. 2005 <http://www.estec.esa.nl/ecls/melissa/attachments/humex/tn1.pdf>

[2] Del Rosso, Dominic. "Re: G-loads." Email to the author. 21 Jan 2005.

[3] Lerner, Preston. "Speed: Formula None." PopSci.com. 2004. Popular Science. 21 Jan. 2005 <http://www.popsci.com/popsci/print/0,21553,681879,00.html>

[4] Rayman, Russell B., M.D., MPH., "Issues In Aerospace Medicine." Jacksonville Medicine. Duval County Medical Society. June 1998. <http://www.dcmsonline.org/jax-medicine/1998journals/june1998/rayman.htm>

1.2 Launch and Entry Suits

Author: Jayleen Guttromson

We require crewmembers to wear launch and entry suits during Earth ascent and entry phases of the mission. Shuttle crews currently use the advanced crew escape suit (ACES). These suits provide protection to a crewmember during ascent and entry for a loss of cabin pressure, exposure to environmental extremes in case of a bailout, protection from a contaminated atmosphere, or loss of oxygen supply [1]. NASA implemented the suits for all crews post-Challenger and we embrace this suggestion for our mission.

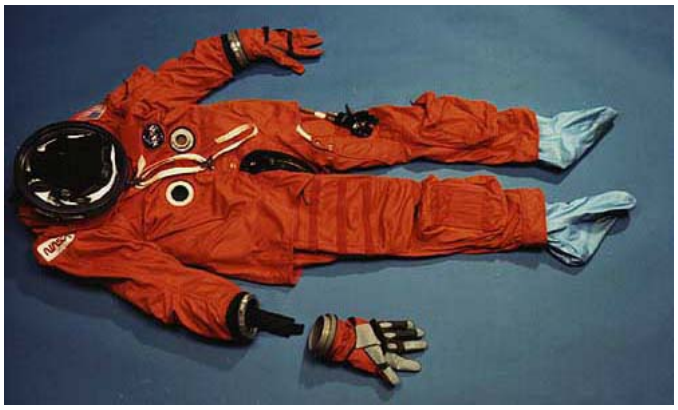


Figure 1-3 ACES Suit Visual (http://neurolab.jsc.nasa.gov/acessuit.htm [2])

In addition to the ACES suit, each astronaut wears a helmet and communications cap. The suit provides pressure protection up to an altitude of 100,000 feet, ambient pressure 0.16 pounds per square inch, and the crewmember's exhaled air maintains the pressure inside the suit [2]. A g-suit applies pressure on the lower body to protect against G-forces and is not integrated into the ACES. We place survival gear in the ACES leg pockets and the Personal Parachute Assembly (PPA). The PPA consists of an emergency oxygen system, pilot and drogue parachutes which operate automatically and have manual backup, a main parachute that operates automatically and has manual backup, a seawater activated parachute release system, flotation devices, a life raft, and survival equipment [3]. We estimate the ACES and PPA weigh 45 kilograms.

Reference

[1] "Escape Systems." 6 Feb. 2005. <http://www.shuttlepresskit.com/scom/210.pdf>

[2] Thomas, Simone and Julie Heath. "Advanced Crew Escape Suit (ACES)." 4 Feb. 2005. <http://neurolab.jsc.nasa.gov/acessuit.htm>

[3] Dismukes, Kim. “Crew Appearal.” NASA Johnson Space Center. 4 Apr. 2002. <http://spaceflight.nasa.gov/shuttle/reference/shutref/crew/apparel.html>

Bibliography

[1] Dismukes, Kim. “Inflight Crew Escape System.” NASA Johnson Space Center. 5 Feb. 2005. <http://spaceflight.nasa.gov/shuttle/reference/shutref/escape/inflight.html>

[2] DX3 Crew Systems, NASA Johnson Space Center, Houston, TX. Phone correspondence. 7 Feb. 2005.

[3] Larson, Wiley J., and Linda K. Pranke. Human Spaceflight Mission Analysis and Design. New York: McGraw-Hill Companies, Inc. 1999.

1.3 Human Factors Equipment on the Ascent and Recovery Vehicle

Author: Jayleen Guttromson

Contributors: Paul Niziolek, Ezdehar Husein

The Ascent and Recovery Vehicle (ARV) Human Factors equipment mass is approximately 875 kilograms and fits within a pressurized volume of 8.3 cubic meters. We assume an average crewmember mass of 70 kilograms, or approximately 155 pounds. According to NASA's Baseline Values and Assumptions Document, a tolerable crew volume assumption is 1.13 cubic meters per crewmember, which means we need less than five cubic meters for the crew while on the ARV [1]. We meet this suggestion and add a requirement that the crew remain in their seats while in the ARV due to space limitations. The seats are ergonomic and easily accommodate the advanced crew escape suits (ACES) and personal parachute assemblies (PPA). Each seat weighs 13 kilograms, based on Martin-Baker crashworthy seats, and integrates into the ARV structure facing the direction of acceleration during launch and entry [2].

The crew's food supply consists of nutrient-rich, individually packaged bars. These snacks fit into one of the ACES suit pockets. We do not schedule or prepare meals for the crew due to the short time they are aboard the ARV. Their water supply, 15 liters, ships in water bottles located next to each of the astronauts' seats. There is no hygiene facility on the ARV, so crewmembers wear maximum absorbency garments or diapers under their ACES to collect urine and feces. Prior to launch, we offer medication to the crew to help minimize bodily functions while seated aboard the ARV.

The ARV initial atmosphere consists of nitrogen and oxygen similar to Earth at 14.7psi. To counteract the crew's oxygen consumption and leaks through pressurized seals, we supply the vehicle with 100 percent oxygen. The tanks containing the re-supply oxygen reside in the non-pressurized volume of the ARV and release the oxygen into the pressurized cabin through pipes and valves. As the crew breathes, we monitor the atmosphere for carbon dioxide and trace contaminates. Lithium hydroxide canisters on-board remove carbon dioxide and the supply assumes a high carbon dioxide generation level by the crew. The fire suppression system includes smoke detectors and fire extinguishers.

The command control station or avionics equipment consists of displays, hand controllers, switches, and monitors. This station provides pertinent information to the pilot and commander and is a subsystem of the guidance system. If the automated docking sequence does not work, the pilot uses this station to manually dock the ARV to the Crew Transport Vehicle (CTV). We focus one of the displays solely on warnings and alarms for the ARV systems and equipment, such as carbon dioxide and oxygen levels.

Table 1-2 Ascent and Recovery Vehicle (ARV) Human Factors Equipment Itemized

Human Factors Equipment	Mass (kg)	Analysis: PV = 8.3m^3; assume all equipment fits inside
Crew	280	70 kg/CM commonsense, volume part of structures PV
Food	1	2 day mission: 1kg; 6 day: 5kg, 14 day: 14kg
Water	15	Stored in water bottles
Air (starting from Earth)	10	Starting at PV*1.2 mass atmosphere Ref. [3]
O2 Consumption	4.2	Consumption rate (0.84kg/CMd HSMAD pg 122 Table 5-16)
O2 re-supply for leaks	1.5	Assume 15% of PV
O2 tanks & piping	2.07	Tanks (0.364*kg/kgO2) BVAD Table 4.1.4 2004
CO2 removal	8.75	1.75 kg/CMd HSMAD Table 17-16 pg 570
Diapers	1	1 diaper/day/CM + storage method (assumed liberal)
Avionics	20	Displays, joystick, switches, command control (HSMAD pg 472 Table 14-6)
Seats	52	13kg/seat Martin-Baker crashworthy seats (49x15x22in)
Suits (4ACES)	180	45kg/ACES - need 1 for each CM WK4-J.Guttromson
Fire Suppression System	5	Smoke detectors, fire extinguisher, warning displays (HSMAD pg 472 Table 14-6 modified)
Lights	5	HSMAD pg 472 Table 14-6
Docking Port	290	Ref. [4]
	875.52	*Subtotal*

Reference

[1] "Advanced Life Support Baseline Values and Assumptions Document." CTSD-ADV-484 A. Crew and Thermal Systems Division. NASA Johnson Space Center. Houston, TX. 16 Aug. 2004. <http://advlifesupport.jsc.nasa.gov/documents/SIMADocs/CR_2004_208941.pdf>

[2] Crashworthy Seats. Martin-Baker Inc. <http://www.martin-baker.com/crashw_Intro.htm>

[3] Niziolek, Paul. "Week 2 Human Factors On-board Atmosphere Parameters." 25 Jan. 2005. <https://engineering.purdue.edu/AAE/Courses/CoursePages/aae450/2005/spring/upload/weekly_presentations/Week_02_1.25.05/WEEK_2_HUMAN_FACTORS_Paul_Niziolek.ppt>

[4] Guttromson, Jayleen. "Week 8 Human Factors Re-evaluation of Docking Mechanism." 8 Mar. 2005.

<https://engineering.purdue.edu/AAE/Courses/CoursePages/aae450/2005/spring/upload/weekly_presentations/Week_08_3.8.05/WK8_HF_J_Guttromson>

Bibliography

[1] Larson, Wiley J., and Linda K. Pranke. Human Spaceflight Mission Analysis and Design. New York: McGraw-Hill Companies, Inc. 1999.

1.4 Human Factors Probability and Risk Assessment Analysis

Author: Jayleen Guttromson

We compute the risk assessment by first outlining the possible failure modes for each of the human factors systems. Next, we rate each of these failure modes assuming zero redundancy of the systems on a scale from zero to five.

Table 1-3 Probability Scale Definitions

Probability Scale	
0	Rarely
1	Minor
2	Fair
3	Moderate
4	Severe
5	Loss of Crew

Then, we rate each of the failure modes assuming ten percent redundancy of the systems on the same scale. We calculate a failure mode probability by dividing the rating by the loss of crew rating. The subtotal for each subsystem originates from the product of the failure mode probabilities. These subtotals add together to obtain the bottom line human factors probability values. The bottom line human factors probability without redundancy is 2.672% and with ten percent redundant systems is 0.786%. The following tables outline the failure modes assumed for each subsystem and Table 1-11 includes a comparison for the subsystems and overall probabilities.

Table 1-4 Atmosphere System Failure Modes and Probability Ratings

System	Failure Mode	No Redundancy	Has Redundancy	Out of
Atmosphere System	Loss of ability to control atmosphere	3	1	5
	Loss of ability for atmosphere revitalization	3	2	5
	Cabin leaks	4	2	5
	Increase in trace contaminants	2	1	5
	Toxic spill in cabin	1	1	5
	CO2 removal system failure	3	2	5
	Power loss to fans / Loss of air cooling to avionics	3	1	5
	Loss of humidity control/condensation control	2	1	5
	System Bottom Line Assessment	0.332%	0.002%	

Table 1-5 Waste Management System Failure Modes and Probability Ratings

System	Failure Mode	No Redundancy	Has Redundancy	Out of
Waste Management	Inability to store/dump waste	2	1	5
	Waste storage tank leak/rupture	1	1	5
	Leaks/blockages in pipes	2	1	5
	Shower/laundry system	1	1	5
	Create more trash than allocated/assumed	2	2	5
	Inability to control odors	1	1	5
	System Bottom Line Assessment	0.051%	0.013%	

Table 1-6 Fire Suppressant System Modes and Probability Ratings

System	Failure Mode	No Redundancy	Has Redundancy	Out of
Fire Suppression	Fire Occurs	0.5	0.5	5
	Fire Detection	1	1	5
	Fire Suppression	3	1	5
	System Bottom Line Assessment	1.200%	0.400%	

Table 1-7 Maintenance System Failure Modes and Probability Ratings

System	Failure Mode	No Redundancy	Has Redundancy	Out of
Maintenance System	Incorrect tools to do on-orbit fixes	2	1	5
	Equipment/tool breaks	1	1	5
	Maintenance by-products contaminates atmosphere	1	1	5
	Run out of supplies	2	1	5
	Run out of replacement units	2	2	5
	No maintenance procedure/accessibility	2	2	5
	System Bottom Line Assessment	0.102%	0.026%	

Table 1-8 Water Management System Failure Modes and Probability Ratings

System	Failure Mode	No Redundancy	Has Redundancy	Out of
Water Management	Recycling equipment breaks	3	2	5
	Unable to recycle at assumed rates	3	1	5
	Water treatment facility failure	3	2	5
	Bacteria/fungi build up	1	1	5
	Tanks/filters/valves/piping leakage or blockage	2	1	5
	Crew consumes more than initially allotted	1	1	5
	System Bottom Line Assessment	0.346%	0.026%	

Table 1-9 Food Preparations System Failure Modes and Probability Ratings

System	Failure Mode	No Redundancy	Has Redundancy	Out of
Food Preparations	Food goes bad	2	2	5
	Equipment breaks – freezer, food warmer	2	1	5
	Crew overeats rations	1	1	5
	Contaminated drinking water	1	1	5
	System Bottom Line Assessment	0.640%	0.320%	

Table 1-10 Human Failure Modes and Probability Ratings

System	Failure Mode	No Redundancy	Has Redundancy	Out of
Humans	Crew has medical emergency	4	3	5
	Psychological break-down	2	2	5
	Human error during critical mission segments	3	2	5
	Exercise equipment breaks	3	2	5
	Vertigo	1	1	5
	AG fails, ~3yr continuous μg effects	1	1	5
	Radiation poisoning	2	2	5
	Environmental monitoring system fails	2	2	5
	Medicine expires	1	1	5
	Crew compatibility – teamwork disintegrates	1	1	5
	Incorrect procedures	2	2	5
	System Bottom Line Assessment	0.001%	0.0004%	

Table 1-11 Subsystem and Overall Probability Ratings

No Redundancy	Probability	10% System Redundancy	Probability
Atmosphere System	0.332%	Atmosphere System	0.002%
Humans	0.001%	Humans	0.0004%
Water Management System	0.346%	Water Management System	0.026%
Waste Management System	0.051%	Waste Management System	0.013%
Food Preparations System	0.640%	Food Preparations System	0.320%
Fire Suppression System	1.200%	Fire Suppression System	0.400%
Maintenance System	0.102%	Maintenance System	0.026%
Bottom Line Assessment	2.672%	*Bottom Line Assessment*	0.786%

1.5 Extravehicular Activity Research and Rationale for Omitting Capability

Author: Jayleen Guttromson

Human operations in a vacuum environment require a personal pressurized spacecraft to protect the crewmember during a spacewalk or Extravehicular Activity (EVA). Today, we have two suits available, Russia's Orlan and NASA's Extravehicular Mobility Unit (EMU). We propose EMUs for Project Legend because they offer improved mobility and dexterity. However, in an effort to save mass and reduce volume on the Crew Transport Vehicle (CTV), an engineering and management decision eliminated the airlock and spacesuits. The following information contains research leading up to the rationale concerning this decision.

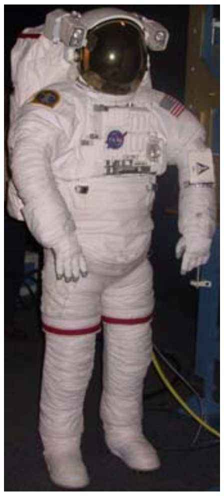


Figure 1-4 Extravehicular Mobility Unit (EMU) Suit Visual (Ref. [1])

An EMU weighs approximately 120 kilograms. It has a suit envelope of 1.92 x 1.04 x 0.76 meters and stows in an envelope of 1.1 x 0.71 x 0.66 meters [2]. Each EMU provides a pressurized enclosure, life support and temperature control system, and micrometeoroid protection during EVA. Each suit contains seven hours of expendables, a 30 minute oxygen reserve, and has a 15 year lifespan. EVA history over the course of 138 spacewalks demonstrates a less than one percent failure rate for an EMU failed start [3] [4]. A current estimate for building a new EMU from all new components is approximately $12 million dollars [4].

An EMU consists of a hard upper torso, lower torso assembly, gloves, helmet and visor assembly, a communications cap, and a liquid cooling and ventilation garment. The upper torso acts as a structural mount for the portable life support system, display and control module, helmet, and lower torso assembly or brief area, including the legs and boots. The portable life support system provides oxygen for breathing, ventilation, and pressurization as well as water for cooling. The EMU is nominally pressurized to 4.3 pounds per square inch.

We propose the CTV include an EMU for each crewmember. EVA capability necessitates incorporating an airlock into the CTV crew compartment configuration. An initial internal airlock estimate with minimal volume is approximately 4.3 cubic meters and weighs in the region of 1000 kilograms. In addition to adding an airlock, we assume a 150 kilogram mass for tools and workaids to fix or inspect the CTV. To support EVA operations, we use approximately 150 kilograms of CTV consumables and loose about ten percent of the airlock's free volume gas each cycle. These additional mass and volume suggestions for the CTV prompted a re-evaluation of having EVA capability during the mission. We were unable to develop a mass-effective solution to keep EVA capability, so we choose to accept the risk and eliminate EVA capability on the CTV.

Reference

[1] Guttromson, Jayleen and Bill Spenny. "EMU and Orlan Suits New Employee Orientation Pitch." Crew and Thermal Systems Division EVA and Spacesuit Systems Branch (EC5). 25 Sept. 2002.

[2] Dismukes, Kim. "Extravehicular Activity Mobility Units." NASA Johnson Space Center. 5 Feb. 2005. <http://spaceflight.nasa.gov/shuttle/reference/shutref/orbiter/eclss/emu.html>

[3] Von Puttkamer, Jesco, Dr. "Extravehicular Activities (EVA) Statistics." NASA. 7 Feb. 2005. <http://www.hq.nasa.gov/osf/EVA/EVA_totals_table.html>

[4] Blanco, Raul, EMU Subsystem Manager, NASA Johnson Space Center, Houston, TX. Phone correspondence. 7 Feb. 2005.

Bibliography

[1] "Advanced Life Support Baseline Values and Assumptions Document." CTSD-ADV-484 A. Crew and Thermal Systems Division. NASA Johnson Space Center. Houston, TX. 16 Aug. 2004. <http://advlifesupport.jsc.nasa.gov/documents/SIMADocs/CR_2004_208941.pdf>

[2] "Airlock Offers New Gateway to Space." STS-104 Shuttle Press Kit. NASA. 6 Feb 2005. <http://www.shuttlepresskit.com/STS-104/payload93.htm>

[3] Kozloski, Lillian. U.S. Space Gear. Washington: Smithsonian Institution Press, 1994.

[4] Larson, Wiley J., and Linda K. Pranke. Human Spaceflight Mission Analysis and Design. New York: McGraw-Hill Companies, Inc. 1999.

[5] NASA Extravehicular Mobility Unit (EMU) Life Support Subsystem (LSS) and Space Suit Assembly (SSA) Data Book. Hamilton Sundstrand. September 2003.

[6] "Space Suit Evolution From Custom Tailored To Off-The-Rack." ILC, Inc. and NASA. <http://history.nasa.gov/spacesuits.pdf>

18 Harrigan, Matt

The following code was written to analyze all loads on pressure vessels for the mission. It was originally written to size all components of the CTV crew quarters pressure vessel. With slight modification, it was used to analyze the MLV, ARV, and MHV. The loads analyzed were of course the pressure differential and also axial compression. A series of vertical spars and circumferential ribs were added to resist buckling under the axial compression. Also, the mass of the floor was calculated. The inputs are the material used, masses of the payload, and the dimensions of the structure. The outputs are the various thicknesses, and of course the total mass.

```
%Matt Harrigan

clc
clear all

%Assumptions
%Cylinder, ends have 2 times the radius
%

thick_alum = .01;
rho_alum = 2780;

poly_thick = .04
poly_rho = 1000

wall_mass = 500;

yield_stress = 1.56e9/4; %pa
E = 140e9/4; %pa
pressure = 101000; %pa
radius = 3.5; %m
length = 5; %m
safety_factor = 2;
shell_safety_factor = 2
rho = 1550 %kg/m3
total_mass = 100000; %total mass including stuff on the inside
launch_g = 6;

%Cylinder
thick_hoop = shell_safety_factor*pressure*radius/yield_stress %m

%Cap
thick_cap = shell_safety_factor*pressure*radius/yield_stress %m

%Mass
inner_shell_mass = (2*pi*radius*thick_hoop*length+2*41.2*thick_cap)*rho %kg
outer_shell_mass = (2*pi*radius*thick_alum*length+2*41.2*thick_alum)*rho_alum
poly_mass = (2*pi*radius*poly_thick*length+2*41.2*poly_thick)*poly_rho

%Stiffeners
%resists axial compression during launch
%assumes worst case load throughout length of cylinder

long_num = 100; %number of axial stiffeners
ring_num = 10; %number of ring stiffeners
stiff_width = poly_thick; %radial dimension of axial stiffeners
stiff_length = length/(ring_num+1); %effective buckling length of axial stiffeners
end_width = .01; %length of end of the C channel
```

```
spacing = 2*pi*radius/long_num; %m, distance between stiffeners
stiff_load = .5*total_mass*launch_g*9.8/long_num; %N

%failure due to column buckling, pinned ends
stiff_thick_1 = safety_factor*stiff_load*12*stiff_length^2/(pi^2*E*(stiff_width)^3) %m
%failure due to yielding
stiff_thick_2 = safety_factor*stiff_load/((stiff_width+2*end_width)*yield_stress) %m
%find bigger of thicknesses
stiff_thick = max([stiff_thick_1 stiff_thick_2]) %m

stiff_mass = stiff_thick*(2*end_width+stiff_width)*length*rho*long_num*2 %kg
ring_mass = 2*radius*pi*stiff_width*stiff_thick*rho*ring_num*2 %kg

%Floors
%floor is carbon fiber and fiberglass honeycomb sandwich, transfers load to
%equally spaced joists
%Assumes each joist carries equal load



mass_per_floor = 8000;
floor_pressure = mass_per_floor*launch_g*9.8/(pi*radius^2);
unsupported_dist = 3;
floor_thick = .02;

floor_sheet_thick = safety_factor*unsupported_dist*floor_pressure/(2*floor_thick*yield_stress)

mass_floor = (2*floor_sheet_thick*pi + rho/500*floor_thick)*(radius-1.5)^2*rho


struc_mass = (ring_mass + stiff_mass + inner_shell_mass + outer_shell_mass + 2*mass_floor + poly_mass
+ wall_mass)
```

The following code is the final code used to size the CTV main truss. The main inputs are the mass of the crew area, the main engine acceleration, and the truss length. The outputs are the diameters and masses.

```
%Solid Main Truss
%Matt Harrigan

clc
close all
clear all

crew_area_mass = 65000;
main_engine_accel = 7.7; %m/s2
length = 60;


radius_vector = (.001:.0001:1);

safety_factor = 2;
dist_between_axial_members = 1.75; %m
dist_between_diag_supports = 2*dist_between_axial_members; %assumes to be at 45 deg angles
E = 72e9;
yield_stress = 490e6;
rho = 2780;
joint_percent = .25

axial_element_load = main_engine_accel*crew_area_mass/3

%Failure mode 1 - yield from compression
radius(1) = sqrt(safety_factor*axial_element_load/(pi*yield_stress));

%Failure Mode 2 - Global Column Buckling
inertia_2 = 5/6*dist_between_axial_members^2*pi.*radius_vector.^2;
```

```
critical_load_2 = pi^2*E*inertia_2/(4*length^2);
[ignore,k] = min(abs(safety_factor*axial_element_load - critical_load_2));
radius(2) = radius_vector(k);

%Failure Mode 3 - Local Column Buckling
inertia_3 = pi.*radius_vector.^4/4;
critical_load_3 = pi^2*E*inertia_3/(4*dist_between_diag_supports^2);
[ignore,k] = min(abs(safety_factor*axial_element_load - critical_load_3));
radius(3) = radius_vector(k);

%Failure Mode 4 - Yielding from spinning tension
radius(4) = sqrt(safety_factor*crew_area_mass*9.8/(pi*yield_stress))

final_radius = max(radius)

%Mass
axial_element_volume = pi*final_radius^2*length;
axial_element_mass = rho*axial_element_volume;
total_mass = 5*axial_element_mass*(1 + joint_percent)
```

The following code is used to calculate the structural mass of a fuel tank. Density has a strong influence on the ratio of fuel mass to structural mass, so liquid hydrogen, liquid methane, liquid oxygen, and RP-1 were all analyzed. The inputs are the size of the tank, and which fuel. The outputs are the thicknesses and masses.

```
%Tank Mass
%Matt Harrigan


%Assumptions
%assumes worst case pressure, ie bottom, is constant everywhere


liquid = 1 %chooses liquid, 1=H2, 2=O2, 3=CH4, 4=RP1
material = 1 %chooses material 1=7075 T6 alum

liquid_density_vector = [70 1140 464 817]; %kg/m3
yield_stress_vector = [490]; %Gpa
wall_density_vector = [2780]; %kg/m3

length = 9.5; %m, length of cylinder section, ignores hemispherical endcaps
radius = 4.75; %m , radius of tank
tank_pressure = 1 %atm, tank pressure in orbit, not peak pressure during launch
launch_g = 10; %max g load during booster launch
safety_factor = 1.25; %constant for both cyl and sphere parts
n_mass = .3 %extra mass in percent for hoses, attachments, insulation, etc, based on paper________

liquid_density = liquid_density_vector(liquid); %kg/m^3
yield_stress =  yield_stress_vector(material); %mpa currently material is 7075-T6 aluminum
wall_density = wall_density_vector(material); %Kg/m^3 currently material is 7075-T6 aluminum

tank_volume = 4/3*pi*radius^3+pi*radius^2*length; %m^3
liquid_mass = liquid_density*tank_volume %kg, mass of hydrogen in tank
max_pressure = liquid_mass*9.8*launch_g/(pi*radius^2)+tank_pressure*101000 %pascals, pressure during
peak g load

end_thick = safety_factor*sqrt(.25*max_pressure^2*radius^2/((yield_stress*10^6)^2)) %m, thickness of
spherical end caps
end_mass = 4*pi*radius^2*end_thick*wall_density*2 %kg mass of both end caps

cyl_thick = safety_factor*sqrt(.75*max_pressure^2*radius^2/((yield_stress*10^6)^2)) %m
cyl_mass = pi*radius^2*cyl_thick*length*wall_density %kg

structural_mass = (1 + n_mass)*(cyl_mass + end_mass) %kg
```

```
total_mass = structural_mass + liquid_mass %kg
mass_fraction = structural_mass/total_mass

The following code estimates the thrust to spin up in a given amount of time, and the fuel required.
Two designs are analyzed are not the final design.  Choice 1 is the original design.  Choice 2 is an
early alternative, which is similar to the final design except that the masses on each end of the
truss are equal.
%Matt Harrigan
%Spin Up Thrust
%AAE 450 Spring 05

%last updated jan 22

clc

spin_radius = 25; %m, distance from spin axis to centroid of outer masses
spin_time = 2; %days, time to spin up or spin down
final_angular_rate = sqrt(9.8/spin_radius) %rad/s
angular_accel = final_angular_rate/(86400*spin_time) %rad/s^2
isp = 300 %isp of spin thrusters

%-----------Choice 1 - 2 outer masses with central core-----------

%outer masses are assumed to be cylinders of constant radius with axis of symmetry parallel to spin
axis
%core is assumed to be cylinder of constant radius
%mass is assumed to be evenly distributed
%thruster is assumed to be on edge of outer mass with constant thrust
%truss is assumed to be of negligible weight, for now atleast

outer_mass = 50; %metric tons
outer_radius = 4; %m, distance from centroid of outer mass to edge of outer mass
core_mass = 200; %metric tons
core_radius = 5; %m, distance from spin axis to edge of the core

core_rho = core_mass/(pi*core_radius^2); %metric ton/m^2, area density
outer_rho = outer_mass/(pi*outer_radius^2); %metric ton/m^2, area density
%inertia_1 = (pi*outer_radius^4/4*outer_rho + outer_mass*spin_radius^2)*2+pi*core_radius^4/4*core_rho
%metric ton*m^2
inertia_1 = outer_mass * 2 * spin_radius^2
thrust_1 = inertia_1*1000*angular_accel/spin_radius %N
fuel_1 = thrust_1*86400*spin_time/(9.8*isp) %kg mass of fuel


%-------------------------Choice 2, 2 equal masses at end of truss------------

%outer masses are assumed to be cylinders with axis of symmetry parallel to spin axis
%mass is assumed to be evenly distributed
%truss is assumed to be of negligible weight, for now atleast
%thruster is assumed to be on edge of outer mass with constant thrust
%outer mass is assumed to be cylinder of constant radius

outer_mass = 125; %metric tons, mass of each outer section, ie. half the vehicle mass
outer_radius = 5; %radius of outer masses
outer_rho = outer_mass/(pi*outer_radius^2); %metric ton/m^2, area density
inertia_2 = (pi*outer_radius^4/4*outer_rho + outer_mass*spin_radius^2)*2
thrust_2 = inertia_2*1000*angular_accel/spin_radius %N total thrust, actually 2 thurster on each outer
mass
fuel_2 = thrust_2*86400*spin_time/(9.8*isp)
```

The following code was used to size the CTV's main truss mass using the original 3 mass configuration. However, this is using a pressurized cylinder as the truss. At the time, we were planning for crew members to be able to traverse the truss to gain easy access to other sections of the vehicle.

```
%Matt Harrigan
%Analysis of truss for A.G., 2 small equal outer masses and large core
%AAE 450 Spring 05

%last updated jan 22

clc
%-------Assumptions--------
%vehicle is spinning at 1 g when main engine fires!!!!!!!
%Truss is a thin pressurized cylinder
%Bending and torsion loads are small and ignored, for now
%only analyzes spinning and pressure loads
%outer mass is a point mass at 1 g accel


pressure = 100000; %N/m^2 or pascal, internal pressure
length = 20; %m assumed to be less than spin radius for now, other structures comprise part of the 30
m spin radius
tube_radius = .5; %m
outer_mass = 25; %metric tons
thick = .0035; %m, constant thickness of wall
rho = 1 %density of material
cable_dist = 10; %m dist from cable attachment to truss
accel = 3 %m/s accel of vehichle do to main engine firing
cable_yield_stress = 350*10^6 %pascals

%pressure loads
hoop_stress = pressure*tube_radius/(thick)*6
long_stress = hoop_stress/2*6

%spinning loads
load = outer_mass * 9.8 * 1000;
area = 2*tube_radius*pi*(thick)
spin_stress = load/area*6

%Von Mises failure criteria
stress_x = hoop_stress; %stress in x direction, tangent to cylinder wall
stress_y = long_stress + spin_stress; %stress in y direction, along axis
%all other stress are zero given assumptions
critical_stress = sqrt((stress_x-stress_y)^2+stress_y^2+stress_x^2)/sqrt(2)/1000000 %Mpa
mass_cyl = thick*2*pi*tube_radius*20*2.78/1000*(100^3) %kg

%cable
theta = atan(30/cable_dist)
cable_load = accel*outer_mass*1000/cos(theta)
cable_area = cable_load/cable_yield_stress*2
cable_diameter = sqrt(cable_area/pi)
cable_length = sqrt(30^2+cable_dist^2)
cable_mass = cable_length*cable_area*2.78/1000*(100^3)

total_mass = cable_mass + mass_cyl*2
```

19 Hauser, Aaron

19.1.1.1 Vehicle Structure Appendix

Author: Aaron Hauser

Contributors: Matt Harrigan, Jeff Yoke

Code (1)

I helped Matt Harrigan write this code for pressurized composite vessels such as the HAB and the living quarters on the CTV. I then altered it further to be used for the HAB

```
%Adapted Matt Harrigans code
clc
clear all

%Assumptions
%Cylinder, ends have 2 times the radius
%

yield_stress = 1.56e9/4; %pa
E = 140e9/4; %pa
pressure = 101325; %pa
radius = 3.5; %m
length = 5.5; %m
safety_factor = 2;
shell_safety_factor = 3
rho = 1550 %kg/m3
total_mass = 50000; %total mass including stuff on the inside
launch_g = 6;
Surface_Area = 2*radius^2*pi+2*radius*pi*length
%Cylinder
thick_hoop = shell_safety_factor*pressure*radius/yield_stress %m

%Cap
thick_cap = shell_safety_factor*pressure*radius/yield_stress %m

%Mass
shell_mass= (2*pi*radius*thick_hoop*length+2*pi*radius^2*thick_cap*1.25)*rho*2 %kg
%double walled

shell_area = 2*pi*radius*length + 2*pi*radius^2*1.25; %m2
shielding_density = shell_mass/shell_area*.1 %g/cm2

%Stiffeners
%resists axial compression during launch
%assumes worst case load throughout length of cylinder

long_num = 30; %number of axial stiffeners
ring_num = 10; %number of ring stiffeners
stiff_width = .08; %radial dimension of axial stiffeners
stiff_length = length/(ring_num+1); %effective buckling length of axial stiffeners
end_width = .01; %length of end of the C channel

spacing = 2*pi*radius/long_num; %m, distance between stiffeners
stiff_load = total_mass*launch_g*9.8/long_num; %N

%failure due to column buckling, pinned ends
stiff_thick_1 = safety_factor*stiff_load*12*stiff_length^2/(pi^2*E*(stiff_width)^3) %m
%failure due to yielding
stiff_thick_2 = safety_factor*stiff_load/((stiff_width+2*end_width)*yield_stress) %m
%find bigger of thicknesses
stiff_thick = max([stiff_thick_1 stiff_thick_2]) %m
```

```
stiff_mass = stiff_thick*(2*end_width+stiff_width)*length*rho*long_num %kg
ring_mass = 2*radius*pi*stiff_width*stiff_thick*rho*ring_num %kg

%Floors
%floor is carbon fiber and fiberglass honeycomb sandwich, transfers load to
%equally spaced joists
%Assumes each joist carries equal load

floor_load = total_mass*9.8*.38; %N
joist_width = .1; %m
joist_height = .2; %m
num_joists = 6;
joist_load = floor_load/num_joists; %N
moment = joist_load*radius/2; %Nm
joist_cap_thick = safety_factor*joist_load*radius/(2*joist_width*joist_height*yield_stress)

joist_web_thick = safety_factor*joist_load*4/(joist_height*yield_stress)
mass_joist=(joist_cap_thick*joist_width*2+joist_web_thick*joist_height)*2*(1+2*sqrt(3)) *2*radius*rho

thick_floor = .05;
sheet_moment = joist_load*radius/(2*num_joists)
thick_sheet = safety_factor*sheet_moment/(thick_floor*yield_stress)

mass_sheet = pi*radius^2*thick_sheet*4*rho
mass_floor = mass_joist + mass_sheet

%raidation shielding
rad_area_density = 0

mass_rad = rad_area_density*Surface_Area


%total mass + 10% for secondary structures and fastners
struc_mass = (ring_mass + stiff_mass + shell_mass + 2*mass_floor + mass_rad)*1.1
```

Code(2)

I originally wrote a code just to calculate the mass of honeycomb needed to absorb the energy of an impact. Jeff Yoke then took it and changed it to include the mass of the entire legs on the lander. I then took his code and changed it to find the mass of entire legs on the HAB

```
clc
clear all

mb = 50000;          % kg mass of vehicle
v = [10];              % m/s velocity of descent
g = 9.8;             % m/s^2 gravtiy
gm = .38*g;          % m/s^2 martian gravity
Wb = mb*gm;          % N martian weight of craft craft
ho = 4;         % m height of center of mass
Ev = 1/2*mb*v.^2+Wb*ho  % J total energy dissipated

g_clear = 2         %ground clearance
theta = 60*pi/180          % angle formed by leg and ground
leg_length = g_clear/sin(theta)     %length of leg
leg_rad = .03       %radius of leg


% honeycomb in leg
Fc = 6894.757*8000;      % Pa Ultimate compressive stress
dh =  .75;                % m usable srtoke
he =  leg_length;                 %
we =  1/62.428*1000*44;  % density
Espec = Fc*dh/he/we    % m^2/s^2 specific energy
leg_hc_mass = Ev/Espec/gm  % kg mass of honeycomb
```

```
Fc = 6894.757*8000;      % Pa Ultimate compressive stress
dh =  .75;                % m usable srtoke
he =  .1;                 % m footpad thickness
we =  1/62.428*1000*44;  % density
Espec_pad = Fc*dh/he/we;    % m^2/s^2 specific energy
pad_hc_mass = Ev/Espec_pad/gm; ; % kg mass of honeycomb

leg_hc = leg_hc_mass+pad_hc_mass
honey_total = 6*leg_hc

% non honey
rho = 1550;
yield_stress = 1.56e9/4;
leg_thick = leg_rad-sqrt(leg_rad^2-2*Wb/(yield_stress*pi))
foot_stem_thick = leg_rad-leg_thick
foot_thick = .01
foot_rad = .25

leg_nonhc=rho*(leg_length*pi*(leg_rad^2-(leg_thick-leg_thick)^2)+(1-dh)*pi*foot_stem_thick^2
+foot_thick*pi*foot_rad^2)
nonhoney_total = 6*leg_nonhc

leg_mass = leg_hc+leg_nonhc
tot_leg_mass = 6*leg_mass
```

19.2 Internal Layout and Configuration Appendix

Author: Aaron Hauser

Contributors: Aaron Kottlowski

Figure 11 is a preliminary design for the HAB before we had any numbers for life support systems, food, and water. It's a cylinder with legs to keep it above the surface. Instead of just resizing it when we saw that our volumetric needs were much less, the whole idea of a cylinder on its side was scraped for the current squashed can shape. This preliminary design did not make the best use of the curved wall space. Landing a cylinder that is standing up is simpler than one that is lying on its side. One concept that is kept through out the design process is the idea of separating the living quarters from the life support equipment.

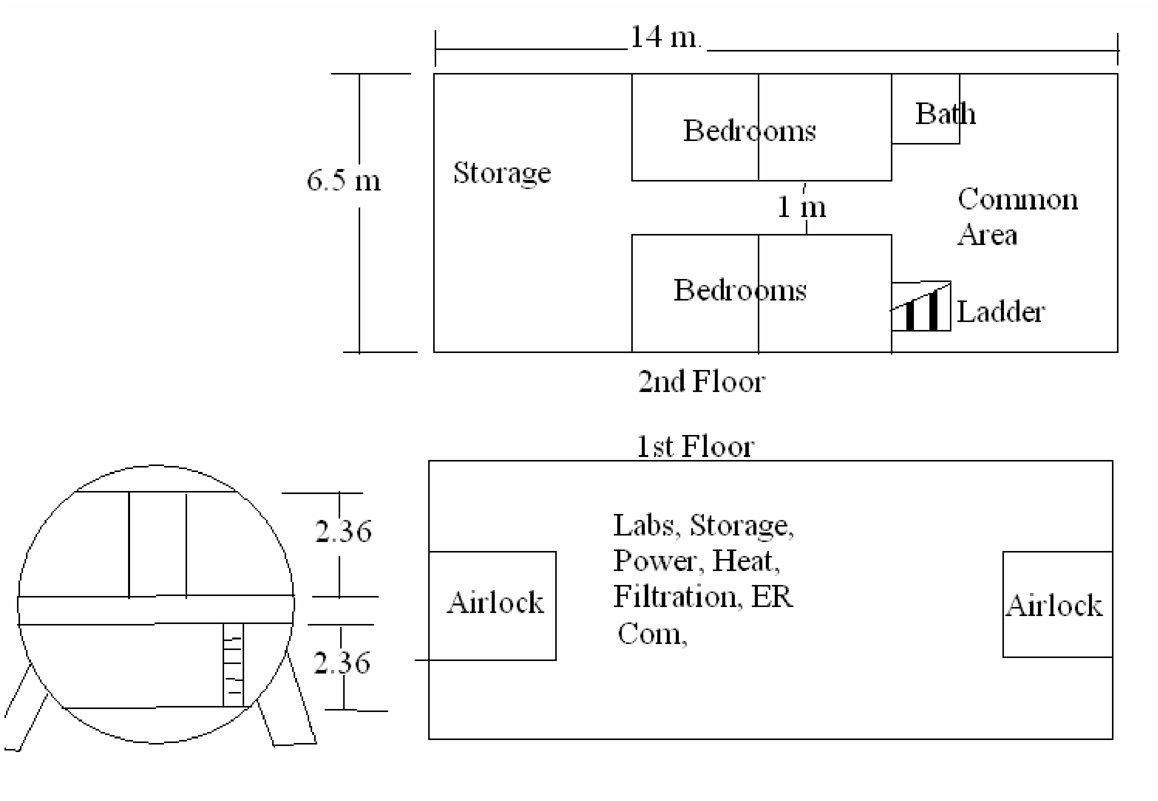


Figure 11 Preliminary HAB Design

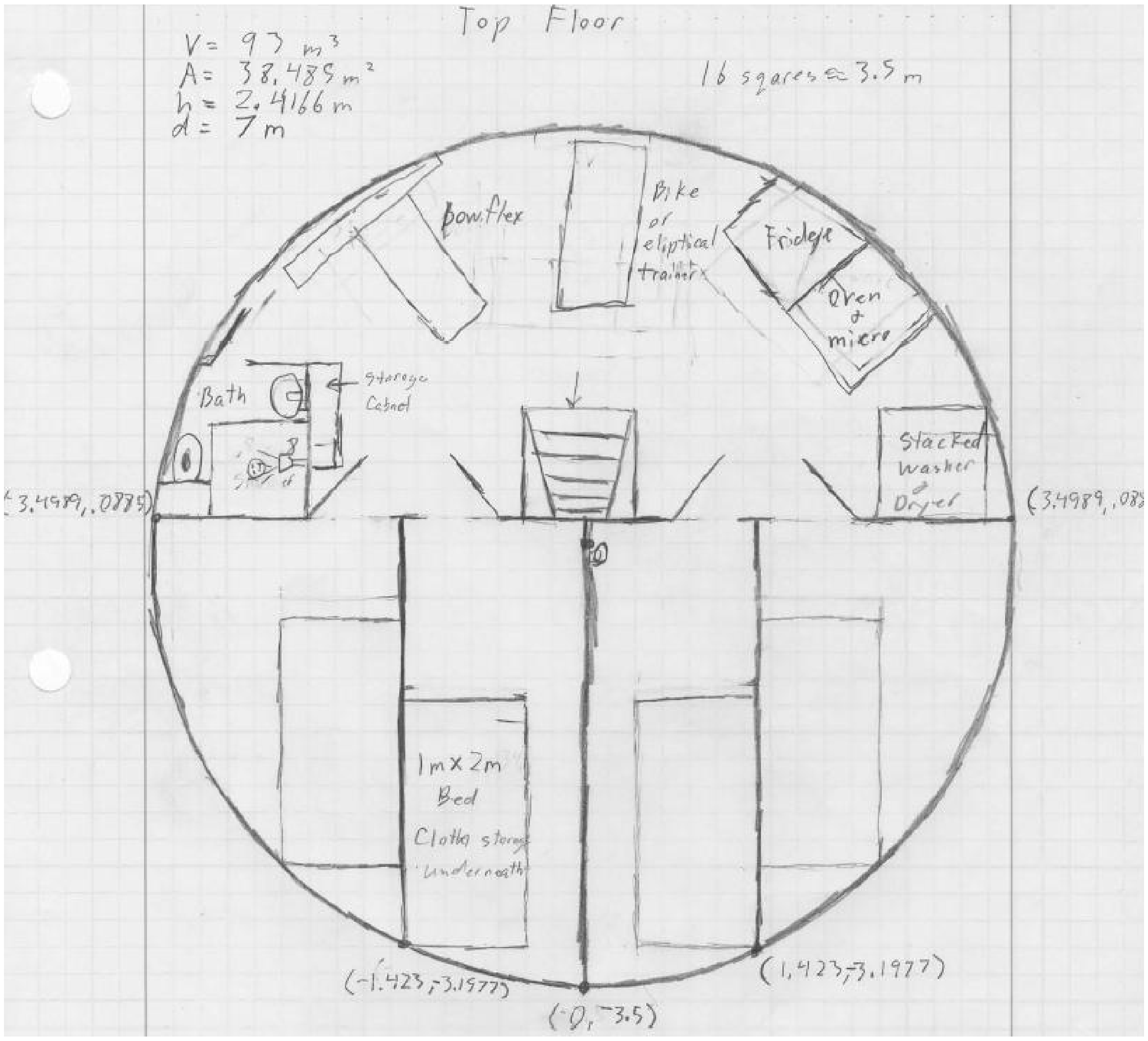


Figure 12 Early "Squashed Can" Top Floor

Figure 12 and Figure 13 are early internal designs for a cylinder that is standing up. Again we see that the living quarters are on the top floor while the bottom floor is reserved for the plants, food, water, and life support systems. While the basic dimensions were kept about the same (5.46 m to 5.5 m height, 7 m diameter), the layout was altered further.

The addition of the radiation shielded room changed the top floor. The rooms were made to be more radial instead of rectangular like in Figure 12 to make better use of the curved walls. The kitchen, laundry area, and bathroom were then placed in the remaining space.

The downstairs layout was most affected by the downsizing of food and plants. The two airlocks in Figure 13 are redundant so one was removed. The hydrogen tank was moved to the outside below the

HAB in order to alleviate the tightness of the downstairs. The plants were moved around so that the astronauts are able to reach them

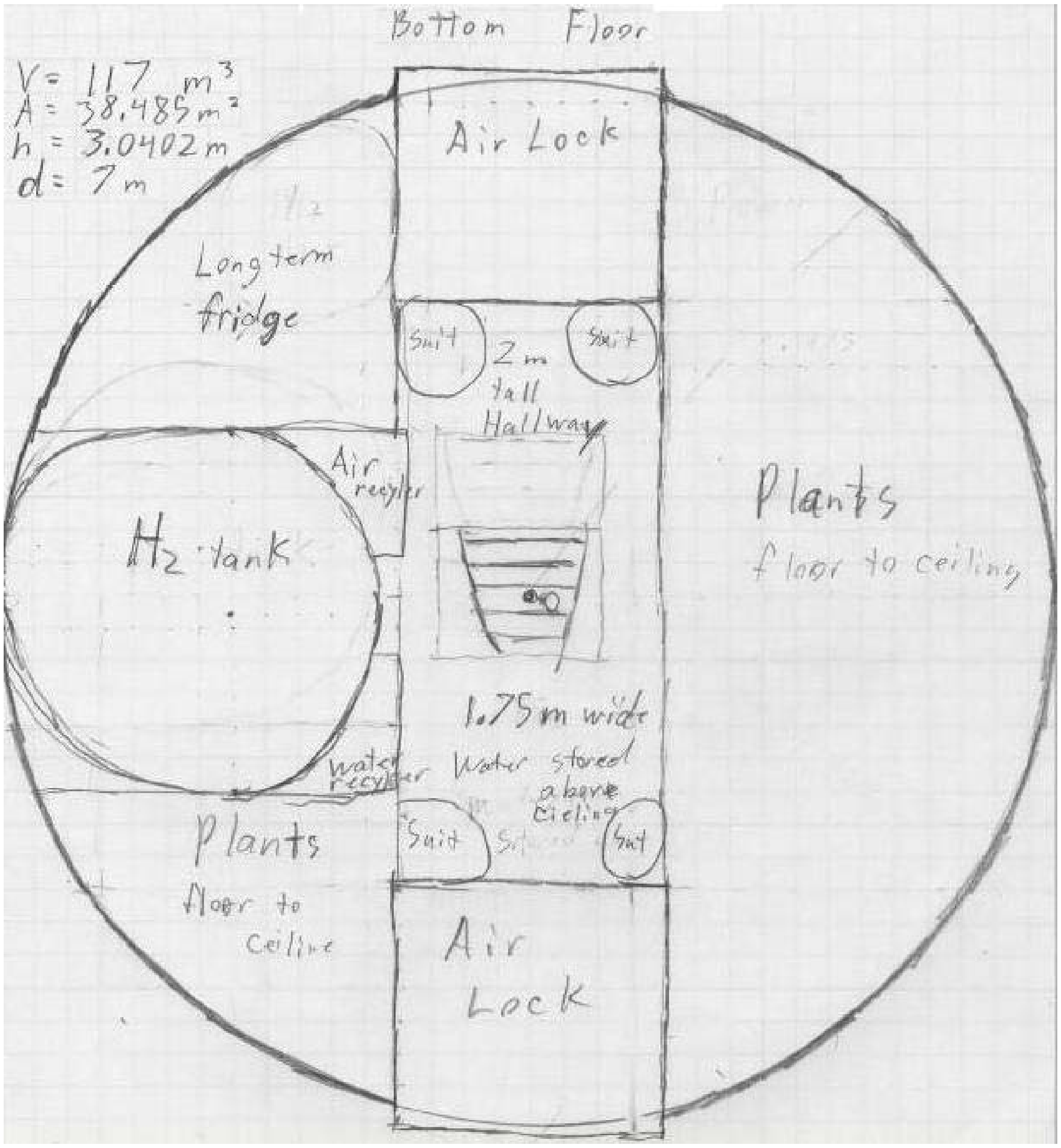


Figure 13 Early "Squashed can" Bottom Floor

20 Matt Heinemann Appendix

Author(s): Matt Heinemann

Contributor(s): Eric Gustafson, Kevin Kloster

20.1.1 Large Period Parking Orbit

From our analysis, the speed of the orbital velocity in the parking orbit around Earth greatly influences the mass of fuel required for the Crew Transport Vehicle (CTV) to escape. The Earth Launch Vehicle (ELV) is severely limited to low altitudes. An orbit meeting these requirements would have a long period, a high eccentricity, and most importantly, a high velocity at periapsis. Boosting the CTV to a long period, high eccentricity parking orbit was studied using several options. It was resource in-effective to carry the fuel required to alter the orbit around Earth. We found the requirements to be resource expensive at the beginning of its mission to change the orbit. Thus we decided on a single orbit, the construction/parking orbit.

20.1.1.1 Assumptions

With the Lambert transfer and parking orbit established (Earth Orbit Configuration) the CTV then has a payload which includes the CTV's dry weight (133 tons), the Ascent and Recovery Vehicle (ARV, 8 tons), the Mars Lander Vehicle (MLV) wet weight (38 tons), and the worst case fuel: the final Earth insertion burn (16 tons), the Mars escape burn (18 tons), the Mars insertion burn (33 tons), and all inclination and trajectory correction maneuvers (21 tons). This is a total payload of 267 tons to leave Earth orbit. The rocket equation:

$$\Delta V = -g\, I_{sp} \ln(1 - m_{dot}\, t_{burn}/m_{initial})$$

Equation **20-1** Rocket Equation

Where g is the constant 9.80665 m/s^2, I_{sp} is the specific impulse in seconds, m_{dot} is the rate of mass loss in kg/s, $m_{initial}$ the initial mass of the rocket in kg, and ΔV the change in velocity in m/s. The propulsion group assured us that the engine start and stop times are instantaneous. We also drop fuel tanks after each burn.

20.1.1.2 Analysis

To determine if by increasing the velocity before the escape burn and by effect decrease the fuel required to escape. To find out if 5 burns would be cheaper than 4. since we were still around Earth, we first considered an ion drive, similar to the XIPS system. However, because the thrust produced was so small and the mass of the CTV was so large, our first estimates put the first burn around Earth requiring 400 years to complete. We then combined 40 of the XIPS thrusters together, to raise the orbit in under 2 years. This required several more nuclear reactors and was therefore abandoned. We then looked at sending up an extra fuel tank which would be dumped after the first burn. This gave us the idea to dump the other tanks en-route. The code we developed to compare the boost to a Large Period Parking Orbit before escape versus immediately escaping from our LEO construction orbit.

20.1.1.3 LPPO versus LEO code

After all the delta-v's had been found, the only input required was the date and the amount of fuel burned in the LPPO boost.

```
% Kevin Kloster & Matt Heinemann
% AAE450
% CTV from LEO to LPPO
% Orients CTV LPPO to shoot CTV
% into transfer orbit to Mars
clc
clear all
close all
format long g
m0 = 269699+12000;    %kg
for k=1:10
mdot = 101.2;   %kg/s
Isp = 1007.6;   %s
g = 9.80665;    %m/s^2
m_prop = 16000;
m_tug = .1*m_prop;
dv = g*Isp*log((m_prop+m0)/m0)/1000;    % deltav from extra tank (LEO-->LPPO)
mue = 398600.4414;
Re = 6378.1;
rp_LEO = Re+200;
v_LEO = sqrt(mue/rp_LEO);
vp_LPPO = v_LEO+dv;
a = mue/(2*mue/rp_LEO-(vp_LPPO)^2);
e = 1-rp_LEO/a;
p = a*(1-e^2);
IP_LPPO = 2*pi*sqrt(a^3/mue);
ra_LPPO = a*(1+e);
va_LPPO = sqrt(2*mue/ra_LPPO-mue/a);
d = 6;  % 1-6 - launch dates
slip = 6; % 1-11, 6 is optimal day
[vxe inc vdx vdy vdz] = trajectory_vxe(d,slip,IP_LPPO);
a_hyp = mue/vxe^2;
e_hyp = rp_LEO/a_hyp+1;
theta_max = acos(-1/e_hyp);
vdepart = [vdx vdy vdz];
vd_hat = vdepart/norm(vdepart);
rp_hatz = vd_hat(3);
c = cos(theta_max)+vd_hat(3)^2;
rp_hatx = -(vd_hat(2)*sqrt((vd_hat(1)^2+vd_hat(2)^2)^2-c^2)-vd_hat(1)*c)/(vd_hat(1)^2+vd_hat(2)^2);
rp_haty = sqrt(vd_hat(1)^2+vd_hat(2)^2-rp_hatx^2);
rp_hat = [rp_hatx rp_haty rp_hatz];
rp_LEO_3D = rp_LEO*rp_hat;
vp_LPPO_3D = vp_LPPO*[-rp_haty rp_hatx rp_hatz];
h_hat = cross(rp_LEO_3D,vp_LPPO_3D)/norm(cross(rp_LEO_3D,vp_LPPO_3D));
inc2 = acos(h_hat(3))
% Quad Check
if h_hat(1)/sin(inc2)>=0 & -h_hat(2)/sin(inc2)>=0 Omega = acos(-h_hat(2)/sin(inc2));
elseif h_hat(1)/sin(inc2)>=0 & -h_hat(2)/sin(inc2)<=0 Omega = acos(-h_hat(2)/sin(inc2));
elseif h_hat(1)/sin(inc2)<=0 & -h_hat(2)/sin(inc2)<=0 Omega = -acos(-h_hat(2)/sin(inc2));
elseif h_hat(1)/sin(inc2)<=0 & -h_hat(2)/sin(inc2)>=0 Omega = -acos(-h_hat(2)/sin(inc2));
end
theta_hat = cross(h_hat,rp_hat);
```

```
if rp_hat(3)/sin(inc2)>=0 & theta_hat(3)/sin(inc2)>=0 theta = acos(rp_hat(3)/sin(inc2));
elseif rp_hat(3)/sin(inc2)>=0 & theta_hat(3)/sin(inc2)<=0 theta = acos(rp_hat(3)/sin(inc2));
elseif rp_hat(3)/sin(inc2)<=0 & theta_hat(3)/sin(inc2)<=0 theta = -acos(rp_hat(3)/sin(inc2));
elseif rp_hat(3)/sin(inc2)<=0 & theta_hat(3)/sin(inc2)>=0 theta = -acos(rp_hat(3)/sin(inc2));
end
w = theta;
rnode_LPPO = p/(1+e*cos(Omega));
vnode_LPPO = sqrt(2*mue/rnode_LPPO-mue/a);
dv_inc = 2*vnode_LPPO*sin(5*pi/180-inc2);
dvtot = dv+dv_inc;
dv_escape_LPPO = sqrt(vxe^2+2*mue/rp_LEO)-vp_LPPO;
dv_LPPO_tot = dvtot+dv_escape_LPPO;
m_prop_LEO_LPPO = exp((dvtot*1000+100)/(Isp*g))*(m0-m_tug)-(m0-m_tug);
[m_tot_LPPO_transfer, m_prop_LPPO_transfer] = burn_fun(dv_escape_LPPO*1000);
m_prop_LPPO = m_prop_LEO_LPPO+m_prop_LPPO_transfer;
m_tot_LPPO = m_tot_LPPO_transfer+m_prop_LEO_LPPO;
m0 = m_tot_LPPO;
end
total_mass = m_tot_LPPO;
propellant_mass = m_prop_LPPO;
dv_escape_LEO = sqrt(vxe^2+2*mue/rp_LEO)-v_LEO
[m_tot_LEO, m_prop_LEO] = burn_fun(dv_escape_LEO*1000)

function [vxe, inc, vdx, vdy, vdz] = trajectory_vxe(d,slip,IP_LPPO)
mue = 398600.4418;
Re = 6378.1;
mum = 42828.2869;
Rm = 3397;
mu = 132712200000;
Rs = 695990;
ae = 149597886.5;
ee = .01671022;
pe = ae*(1-ee^2);
ne = sqrt(mu/ae^3);
am = 227936636;
nm = sqrt(mu/am^3);
em = .09341233;
bm = am*sqrt(1-em^2);
pm = am*(1-em^2);
% Ephemeris (from STK 6.0)
Load_Data
dates = [('December 30, 2015');
         ('March 7, 2018    ');
         ('June 26, 2020    ');
         ('August 3, 2022   ');
         ('October 17, 2024 ');
         ('October 27, 2026 ')];
r_Eo = e_pos(d,1:3);
v_Eo = e_pos(d,4:6);
r_Mo = m_pos(d,1:3);
v_Mo = m_pos(d,4:6);
% Earth parking orbit
rp_LEO = Re+200;
IP_LEO = IP_LPPO;
a_LEO = ((IP_LEO/(2*pi))^2*mue)^(1/3);
e_LEO = 1-rp_LEO/a_LEO;
v_LEO = sqrt(2*mue/rp_LEO-mue/a_LEO);
% Mars parking orbit
rp_LMO = Rm+350;
IP_LMO = 1*60*60*24;
a_LMO = ((IP_LMO/(2*pi))^2*mum)^(1/3);
e_LMO = 1-rp_LMO/a_LMO;
v_LMO = sqrt(2*mum/rp_LMO-mum/a_LMO);
% Launch slip
t_slip = 24*60*60*[-10 -7 -5 -3 -1 0 1 3 5 7 10];
Ee = 0;
for k = 1:length(t_slip)
    Ee = Ee-(Ee-ee*sin(Ee)-ne*t_slip)/(1-ee*cos(Ee));
end
fe = 1-ae/norm(r_Eo)*(1-cos(Ee));
ge = t_slip-1/ne*(Ee-sin(Ee));
```

```
Em = 0;
for k = 1:length(t_slip)
    Em = Em-(Em-em*sin(Em)-nm*t_slip)/(1-em*cos(Em));
end
fm = 1-am/norm(r_Mo)*(1-cos(Em));
gm = t_slip-1/nm*(Em-sin(Em));
for j = 1:length(t_slip)
    r_E(j,:) = fe(j)*r_Eo+ge(j)*v_Eo;
    fedot(j) = -sqrt(mu*ae)/(norm(r_E(j,:))*norm(r_Eo))*sin(Ee(j));
    gedot(j) = 1-ae/norm(r_E(j,:))*(1-cos(Ee(j)));
    v_E(j,:) = fedot(j)*r_Eo+gedot(j)*v_Eo;
    r_M(j,:) = fm(j)*r_Mo+gm(j)*v_Mo;
    fmdot(j) = -sqrt(mu*am)/(norm(r_M(j,:))*norm(r_Mo))*sin(Em(j));
    gmdot(j) = 1-am/norm(r_M(j,:))*(1-cos(Em(j)));
    v_M(j,:) = fmdot(j)*r_Mo+gmdot(j)*v_Mo;
    % Transfer ellipse geometry
    rp = norm(r_E(j,:));
    IP = 1.5*365.25*24*60*60;
    a = ((IP/(2*pi))^2*mu)^(1/3);
    e = 1-rp/a;
    p = a*(1-e^2);
    ra = a*(1+e);
    n = sqrt(mu/a^3);
    TA = acos(dot(r_E(j,:),r_M(j,:))/(norm(r_E(j,:))*norm(r_M(j,:))));
    E = 2*atan(tan(TA/2)/sqrt((1+e)/(1-e)));
    t(j) = 1/n*(E-e*sin(E));
    f(j) = 1-a/norm(r_E(j,:))*(1-cos(E));
    g(j) = t(j)-1/n*(E-sin(E));
    fdot(j) = -sqrt(mu*a)/(norm(r_M(j,:))*norm(r_E(j,:)))*sin(E);
    gdot(j) = 1-a/norm(r_M(j,:))*(1-cos(E));
    v_D(j,:) = (r_M(j,:)-f(j)*r_E(j,:))/g(j);
    v_xe(j,:) = v_D(j,:)-v_E(j,:);
    dvi(j) = sqrt(norm(v_xe(j,:))^2+2*mue/rp_LEO)-v_LEO;
    v_A(j,:) = fdot(j)*r_E(j,:)+gdot(j)*v_D(j,:);
    v_xm(j,:) = v_A(j,:)-v_M(j,:);
    dvf(j) = sqrt(norm(v_xm(j,:))^2+2*mum/rp_LMO)-v_LMO;

    h_hat(j,:) = cross(r_E(j,:),v_D(j,:))/norm(cross(r_E(j,:),v_D(j,:)));
    inc(j) = acos(h_hat(j,3));
    Omega(j) = asin(h_hat(j,1)/sin(inc(j)));
    Omegachk(j) = acos(-h_hat(j,2)/sin(inc(j)));
    r_hat(j,:) = r_E(j,:)/norm(r_E(j,:));
    theta_hat(j,:) = cross(h_hat(j,:),r_hat(j,:));
    theta(j) = asin(r_hat(j,3)/sin(inc(j)));
    thetachk(j) = acos(theta_hat(j,3)/sin(inc(j)));
    w(j) = theta(j);   % thetastar is always zero initially for this orbit
end
dvi = dvi';
dvf = dvf';
dvtot = dvi+dvf;
[dv day] = min(dvtot);
best_deltav = dv;
best_day = t_slip(day)/24/60/60;
TOF = t'/24/60/60;
date = dates(d,:);
depart_velocity = v_D;
vxe = norm(v_xe(slip,:));
inc = inc(slip);
vdx = v_D(slip,1);
vdy = v_D(slip,2);
vdz = v_D(slip,3);
return

% propellant mass calculator
function [m_tot, m_prop] = burn_fun(dv1)
format long g
mdot = 101.2/3;   % kg/s
x = 3;           % number of engines
Isp = 1007.6;    % s
g0 = 9.80665;
max_acc = 5;     % m/s^2
```

```
m_ARV = 2112.479;   % kg
m_MLV = 23904.32;   % kg
m_CTV = 143881.6;     % kg
m_humans = 280;    % kg
m_empty = m_CTV-m_humans;   % kg
d = 3;      % mission opportunity
dv =  [956 4519  830  886;
        787 1187 1353  871;
       1020 1016 1916 1220;
       1247 1349 1120  786;
        936  861  929  845;
        999 1918  933  966];
dve_A = dv(d,4);  % m/s
m_prop4 = exp(dve_A/(Isp*g0))*m_empty-m_empty;    % kg
t_inc = .1;
m4 = 0;
t4 = 0;
k = 1;
while m4 <= m_prop4
    t4 = t4+t_inc;
    m4 = m4+mdot*x*t_inc;
    m_acc4 = m_empty+m_prop4-m4;
    F = mdot*x*g0*Isp;
    acc4(k) = F/m_acc4;        % acceleration
    if acc4(k) > max_acc
        x = x-1;
    end
    if x == 0
        x = 1;
    end
    k=k+1;
end
x4 = x;
t_4 = linspace(0,t4,length(acc4));
max_acc4 = max(acc4(10:length(acc4)));
mf3 = m_empty+m_prop4+m_humans+m_ARV;   % kg
dvm_D = dv(d,3);    % m/s
m_prop3 = exp(dvm_D/(Isp*g0))*mf3-mf3;    % kg
m3 = 0;
t3 = 0;
x = 3;
k = 1;
while m3 <= m_prop3
    t3 = t3+t_inc;
    m3 = m3+mdot*x*t_inc;
    m_acc3 = mf3+m_prop3-m3;
    F = mdot*x*g0*Isp;
    acc3(k) = F/m_acc3;        % acceleration
    if acc3(k) > max_acc
        x = x-1;
    end
    if x == 0
        x = 1;
    end
    k=k+1;
end
x3 = x;
t_3 = linspace(0,t3,length(acc3));
max_acc3 = max(acc3(10:length(acc3)));
mf2 = mf3+m_prop3+m_MLV;
dvm_A = dv(d,2);    % m/s
m_prop2 = exp(dvm_A/(Isp*g0))*mf2-mf2;    % kg
m2 = 0;
t2 = 0;
x = 3;
k = 1;
while m2 <= m_prop2
    t2 = t2+t_inc;
    m2 = m2+mdot*x*t_inc;
    m_acc2 = mf2+m_prop2-m2;
    F = mdot*x*g0*Isp;
```

```
        acc2(k) = F/m_acc2;         % acceleration
        if acc2(k) > max_acc
            x = x-1;
        end
        if x == 0
            x = 1;
        end
        k=k+1;
    end
    x2 = x;
    t_2 = linspace(0,t2,length(acc2));
    max_acc2 = max(acc2(10:length(acc2)));
    mf1 = mf2+m_prop2;
    dve_D = dv1;        % m/s
    dv_TCM = 100;       % m/s
    m_prop1 = exp((dve_D+dv_TCM)/(Isp*g0))*mf1-mf1;    % kg
    m1 = 0;
    t1 = 0;
    x = 3;
    k = 1;
    while m1 <= m_prop1
        t1 = t1+t_inc;
        m1 = m1+mdot*x*t_inc;
        m_acc1 = mf1+m_prop1-m1;
        F = mdot*x*g0*Isp;
        acc1(k) = F/m_acc1;         % acceleration
        if acc1(k) > max_acc
            x = x-1;
        end
        if x == 0
            x = 1;
        end
        k=k+1;
    end
    x1 = x;
    t_1 = linspace(0,t1,length(acc1));
    max_acc1 = max(acc1(10:length(acc1)));
    m_tot = mf1+m_prop1;
    m_prop = m_prop1+m_prop2+m_prop3+m_prop4;
```

After checking the range from minimum burn to max burn without leaving orbit it was determined that the LPPO boost almost never reduced the total fuel mass below that required for an escape from LEO.

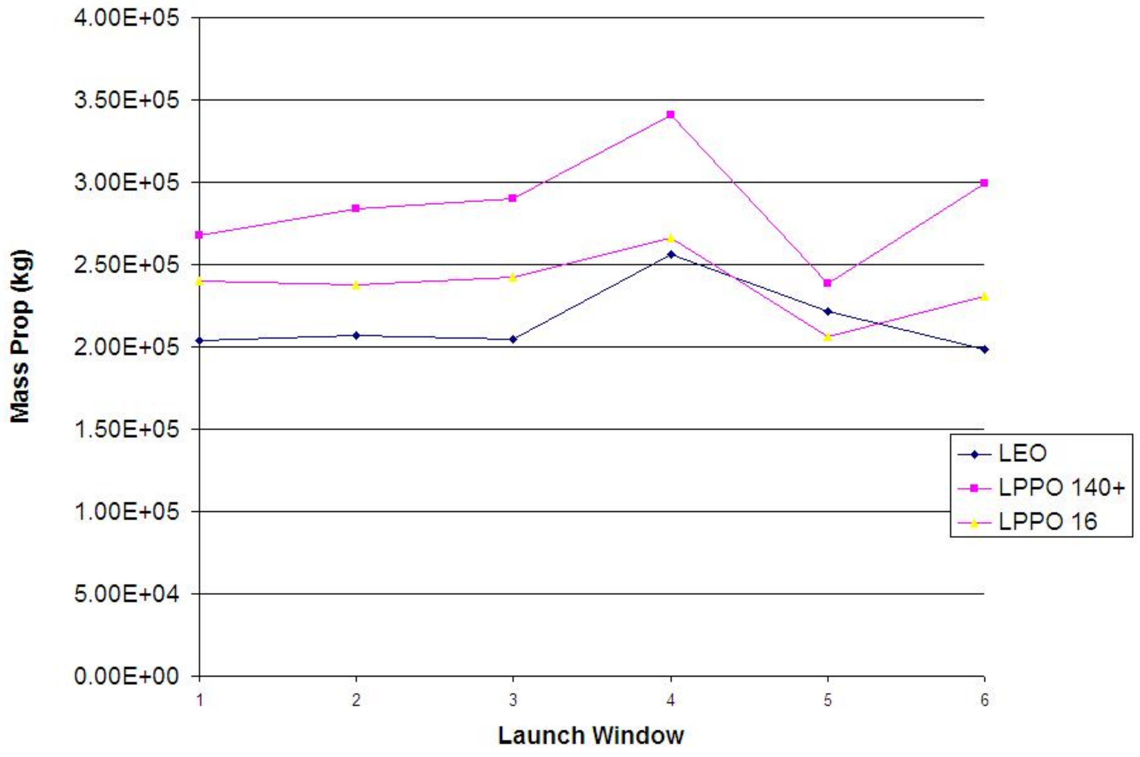


Figure 20-1 LPPO mass range vs LEO mass

The two upper lines show the range in propellant mass using the LPPO boost. Only the 5th window actually saved fuel using a vary small boost. After deciding on a single escape burn from LEO the code was rewritten and can be found in Kevin Kloster's Appendix, called simple_burn2.

20.1.2 Moon Windows

The Moon crosses the CTV's exit vector periodically when the CTV is performing its escape. We set up a box around the Moon to make sure that its gravity did not affect the CTV significantly. If the box crossed our exit trajectory on a launch date we would not launch. The box required that the Earth be the closest gravitational body to the CTV. Worst case, the Moon will only turn the CTV half a degree.

20.1.2.1 Assumptions

The Earth and the Moon were modeled using a two-body method. We treated any gravitational influence from the Moon as a hyperbolic flyby. By limiting the size of the effective delta-v from the Moon, we created a wall of influence the CTV could not enter.

20.1.2.2 Analysis

We decided to keep the Moon's influence from turning the CTV more than ½ a degree at most. $\delta = 0.5$.

$$\sin(\delta/2) = 1/e$$

Equation **20-2** Turn Angle

This gives us a hyperbolic eccentricity of 229.184.

The velocity of the Moon:

$$V_{Moon}^2/2 = \mu_{Earth}/r_{Moon} - \mu_{Earth}/2a_{Moon}$$

Equation **20-3** Lunar Velocity

This gives a velocity of 0.1129 km/s around Earth. The velocity of the CTV near Earth is 33.2 km/s. The cosine law gives us:

$$V_{inf}^2 = V_{Moon}^2 + V_{CTV}^2 - V_{Moon}\, V_{CTV} \cos\gamma$$

Equation **20-4** Cosine Law

This gives a V_{inf} of 44.60 km/s towards the Moon. From this we get the hyperbolic semi-major axis:

$$a_{Hyp} = \mu_{Moon}/ V_{inf}^2$$

Equation **20-5** Hyperbolic semi-major axis

Combining this with out hyperbolic eccentricity:

$$r_{pHyp} = a_{Hyp}\,(e-1)$$

Equation **20-6** Radius at periapsis

This gives a radius of effect of 562 km or a 3 hour wall blocking the CTV during exit.

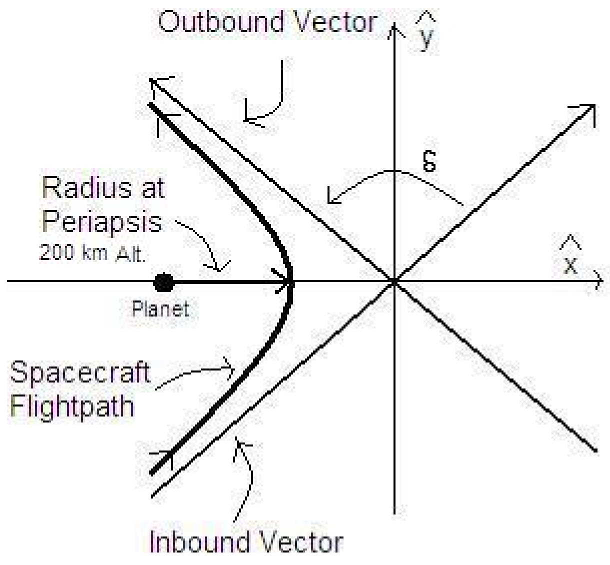


Figure **20-2** Flyby

Since the CTV is in a 90 minute parking orbit, this means at most, there will only be a 2 orbit delay, before escape. Using STK, we matched the exit vectors during each window date to see if the Moon was in the general area. Luckily the Moon was only in the way of the Mission 7 Window. However, we switched to the non-free return trajectory, which changed the launch date to a date when the Moon was not close enough to influence the CTV.

21 Hoff, Phil

21.1 LES Rocket Code

Author: Phil Hoff

21.1.1 Modeling System Behavior

In order to model the behavior of the Launch Escape System (LES), we wrote a MatLab code using a series of equations and approximations. We assumed system equilibrium for chamber conditions. This process is modeled in the flow charts on the following pages. The LES configuration was designed to be similar to the LES used in the Apollo missions. What makes this system different from the Apollo LES is that it is required to lift a much heavier Crew Module than the Apollo LES. The larger mass of the Crew Module is due to two fundamental differences in mission requirements for the two programs. The first difference is that we are required to launch four astronauts in the crew module instead of the three that the Apollo program was required to launch. The second difference is that our Crew Module is required to withstand reentry into the Earth's atmosphere from a much higher velocity. Due to the higher reentry velocity, we have to include a much thicker and heavier heat shield than the Apollo mission employed. The added heat shield mass hinders the LES in pulling the Crew Module from the launch vehicle, but it could also have some benefits in protecting the crew in the event that the Crew Module is still relatively close to the launch vehicle during a catastrophic failure.

The propellant that we select for the three rocket motors is TP-H-3304 which consists of 18% Al, 71% AP, and 11% HTPB. [1] This propellant has been used in the Star 48 and Star 37 rocket motors. We also considered propellant TP-H-1202 (used in the Star 63D rocket motor) for its higher density and c* value but we deemed it to be too unsafe due to its 12% composition of HMX (Her Majesty's Explosive).

Inputs:

- n = number of points in propellant star configuration
- *wedge_angle_factor* = angle of each wedge of propellant wrt the angle of the sector that encompasses it
- Rp = initial port radius
- L = length of the propellant grain
- ε = nozzle expansion ratio
- PR = initial port to throat ratio (must be at least 2)

Initialize propellant properties (data from SPAD) [1]
getPropellantProperties.m

- ρ_P= Propellant Density
- c* = Burn Efficiency
- a_b=Burn Rate Coefficient
- n_b = Burn Rate Exponent

Initial Port Geometry

$$\phi = 2\pi/n$$ (Sector Angle)

$$b = 2R_P \sin(\phi/2)$$

(Base of triangle forming propellant wedge)

(Also Base of triangle filling sector of the port)

$$h_{wedge} = \frac{b}{2}\tan(\theta/2)$$

(Height of triangle forming propellant wedge)

$$h_{\sec} = \sqrt{R_P^2 - \left(b/2\right)^2}$$

(Height of triangle filling a sector of the port)

$$t = S/2$$

(web thickness outside the star)

$$\theta = \phi * wedge\_angle\_factor$$

(Wedge Angle)

$$A_{Sec} = \pi R_P^2 / n$$ (Area of a sector)

$$A_{\sec\_tri} = \tfrac{1}{2} bh$$

(Area of triangle filling a sector)

$$A_{Sliver} = A_{Sec} - A_{\sec\_tri}$$

(Area of propellant btw base of triangle and sector arc)

$$A_{wedge} = \tfrac{1}{2} bh_{wedge}$$

(Area of propellant in wedge triangle)

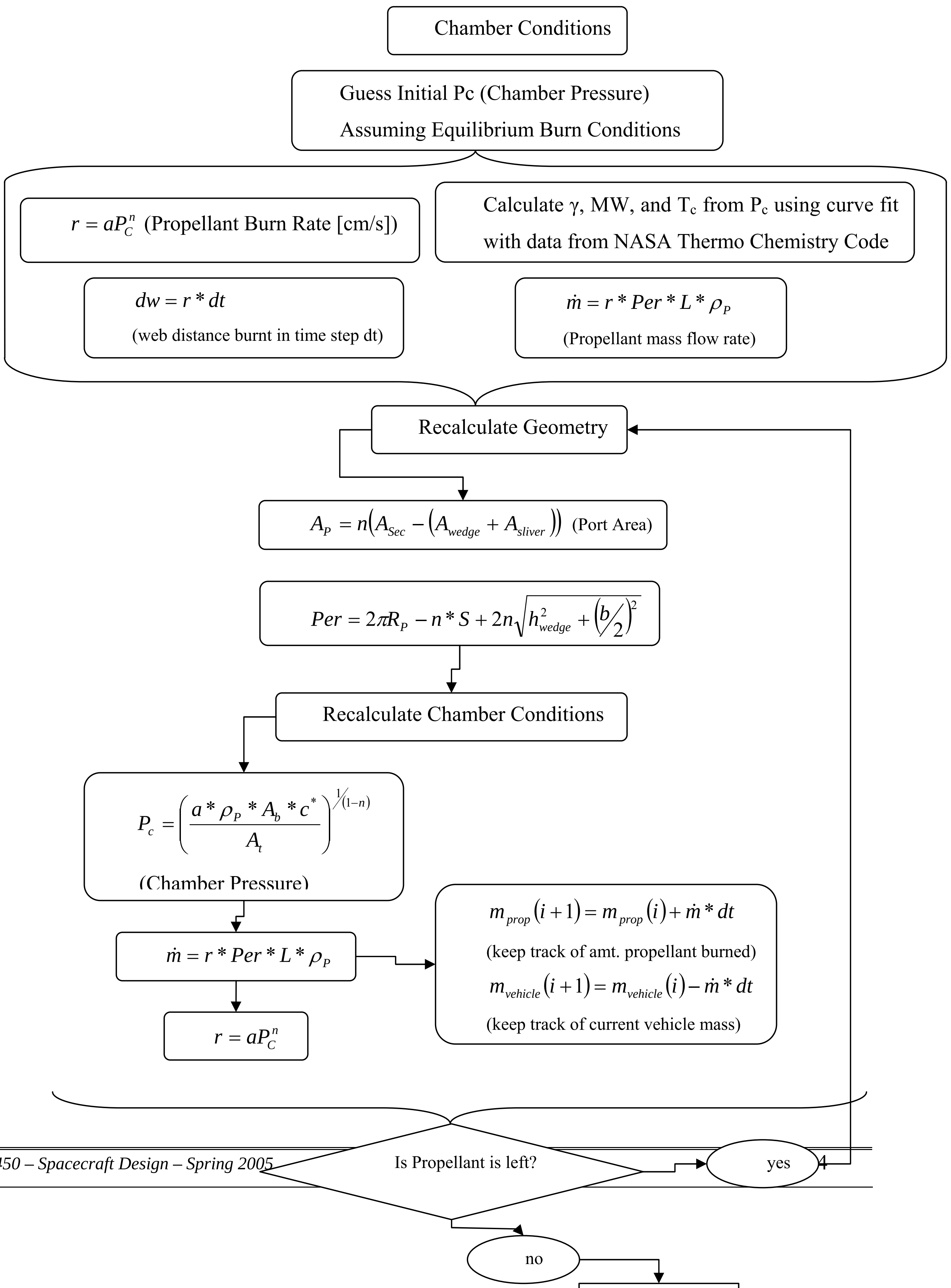
$$A_P = n\left(A_{Sec} - \left(A_{wedge} + A_{sliver}\right)\right)$$ (Port Area)

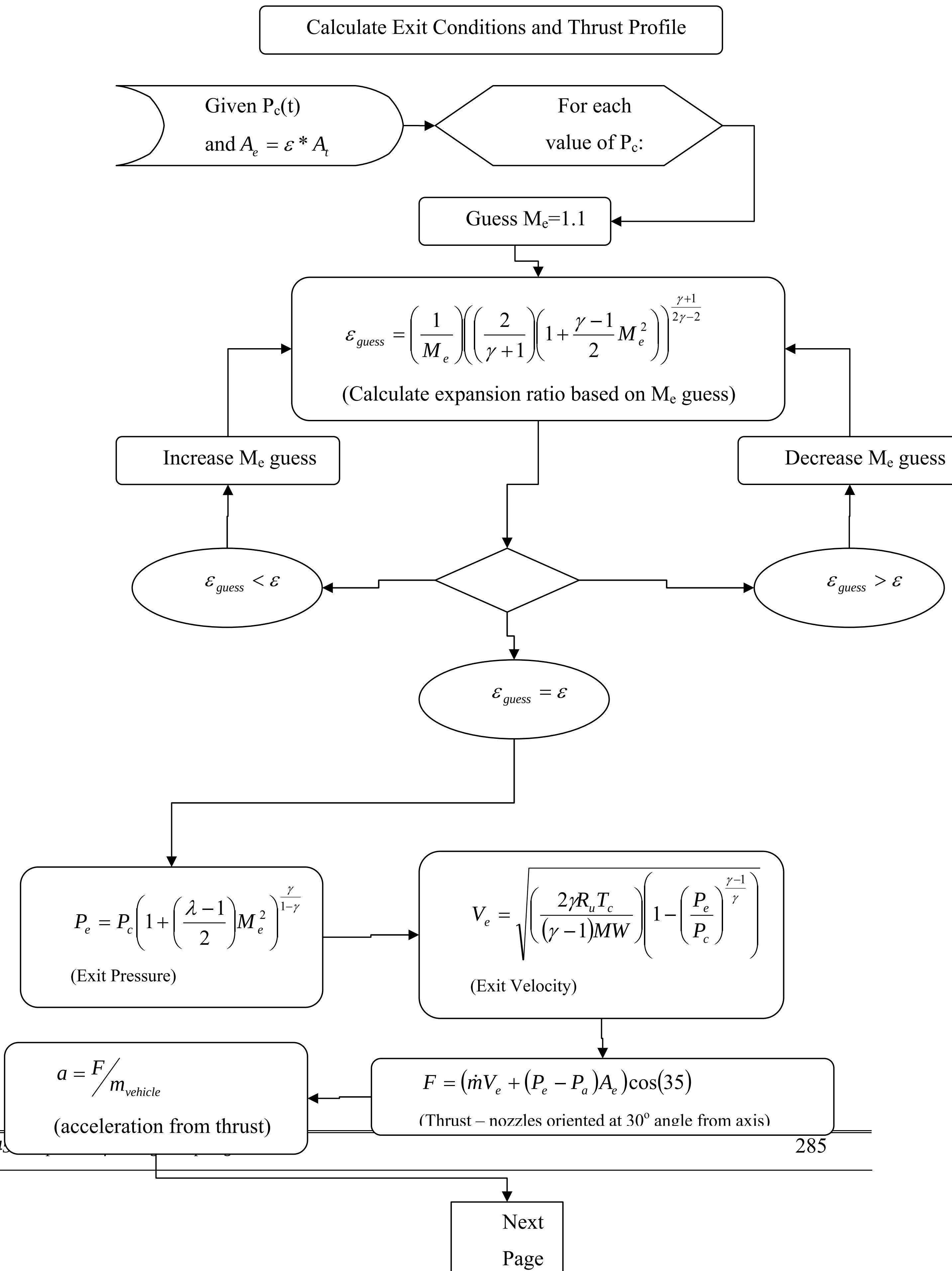
$$A_t = A_P / PR$$ (Throat Area)

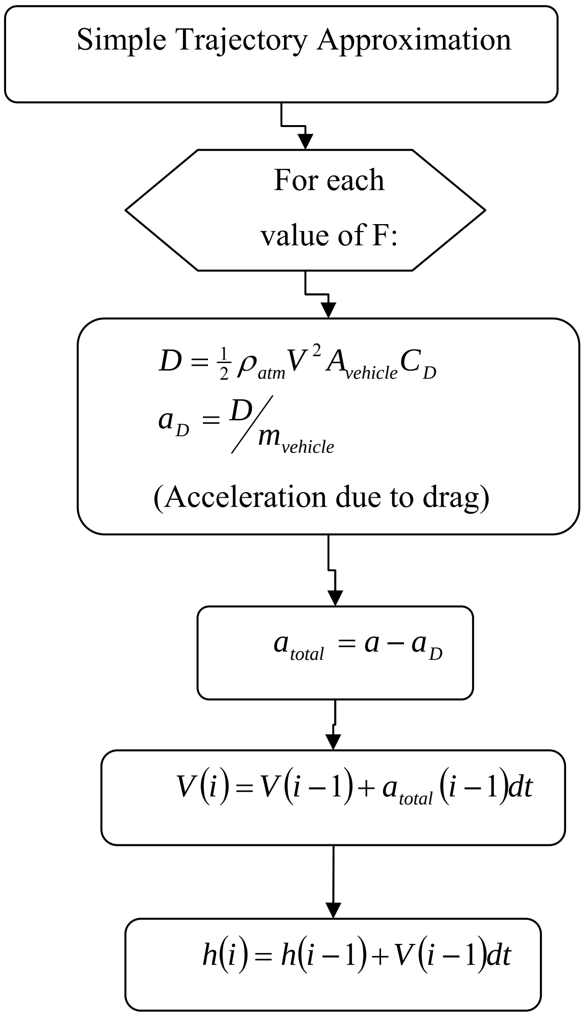
$$Per = 2n\sqrt{h_{wedge}^2 + \left(b/2\right)^2}$$ (Perimeter of port area)

$$A_b = Per * L$$ (Burn Area)

Next Page


Chamber Conditions
Guess Initial Pc (Chamber Pressure) Assuming Equilibrium Burn Conditions
$r = aP_C^n$ (Propellant Burn Rate [cm/s])
Calculate γ, MW, and T_c from P_c using curve fit with data from NASA Thermo Chemistry Code
$dw = r * dt$ (web distance burnt in time step dt)
$\dot{m} = r * Per * L * \rho_P$ (Propellant mass flow rate)
Recalculate Geometry
$A_P = n\left(A_{Sec} - \left(A_{wedge} + A_{sliver}\right)\right)$ (Port Area)
$Per = 2\pi R_P - n * S + 2n\sqrt{h_{wedge}^2 + \left(b/2\right)^2}$
Recalculate Chamber Conditions
$P_c = \left(\frac{a * \rho_P * A_b * c^*}{A_t}\right)^{1/(1-n)}$ (Chamber Pressure)
$\dot{m} = r * Per * L * \rho_P$
$r = aP_C^n$
$m_{prop}(i+1) = m_{prop}(i) + \dot{m} * dt$ (keep track of amt. propellant burned)
$m_{vehicle}(i+1) = m_{vehicle}(i) - \dot{m} * dt$ (keep track of current vehicle mass)
Is Propellant is left?
yes
no


Calculate Exit Conditions and Thrust Profile
Given $P_c(t)$ and $A_e = \varepsilon * A_t$
For each value of P_c:
Guess M_e=1.1
$\varepsilon_{guess} = \left(\frac{1}{M_e}\right)\left(\left(\frac{2}{\gamma+1}\right)\left(1+\frac{\gamma-1}{2}M_e^2\right)\right)^{\frac{\gamma+1}{2\gamma-2}}$
(Calculate expansion ratio based on M_e guess)
Increase M_e guess
Decrease M_e guess
$\varepsilon_{guess} < \varepsilon$
$\varepsilon_{guess} > \varepsilon$
$\varepsilon_{guess} = \varepsilon$
$P_e = P_c\left(1+\left(\frac{\lambda-1}{2}\right)M_e^2\right)^{\frac{\gamma}{1-\gamma}}$
(Exit Pressure)
$V_e = \sqrt{\left(\frac{2\gamma R_u T_c}{(\gamma-1)MW}\right)\left(1-\left(\frac{P_e}{P_c}\right)^{\frac{\gamma-1}{\gamma}}\right)}$
(Exit Velocity)
$F = (\dot{m}V_e + (P_e - P_a)A_e)\cos(35)$
(Thrust – nozzles oriented at 30° angle from axis)
$a = F / m_{vehicle}$
(acceleration from thrust)
Next Page


Simple Trajectory Approximation
For each value of F:
$D = \frac{1}{2}\rho_{atm}V^2 A_{vehicle}C_D$
$a_D = D/m_{vehicle}$
(Acceleration due to drag)
$a_{total} = a - a_D$
$V(i) = V(i-1) + a_{total}(i-1)dt$
$h(i) = h(i-1) + V(i-1)dt$

21.1.2 Results

The following are the results of the rocket code for each of the three rocket motors in the Launch Escape System:

21.1.2.1 Launch Escape Motor

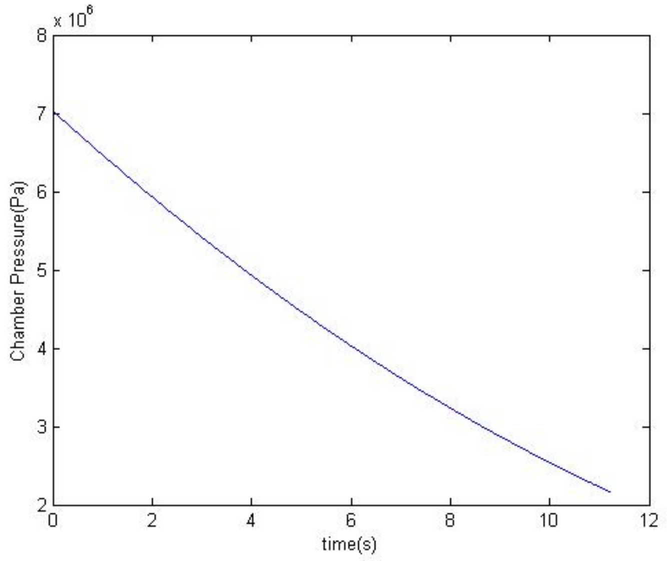


Figure 3 LEM Chamber Pressure

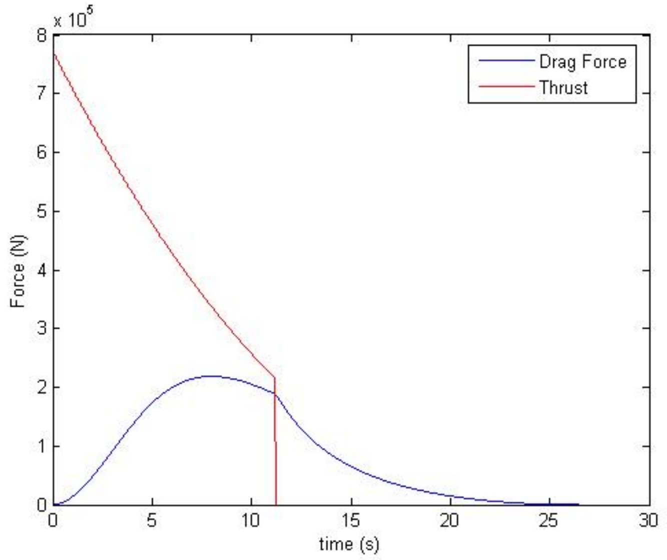


Figure 4 Thrust & Drag Forces on Vehicle

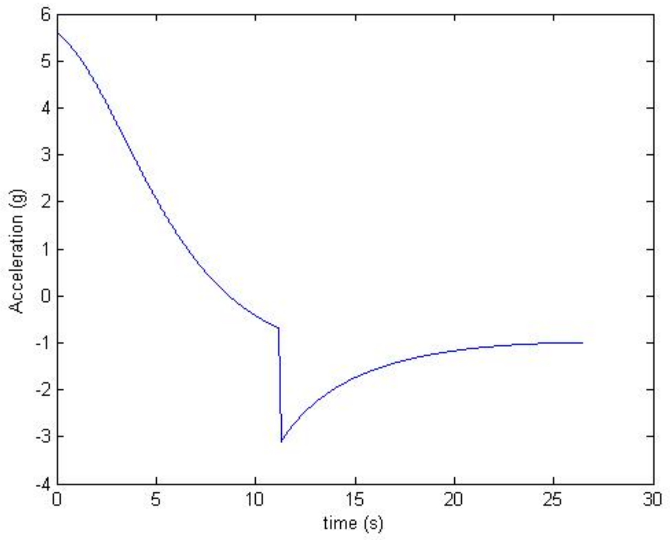


Figure 5 Vehicle Acceleration During Launch Escape

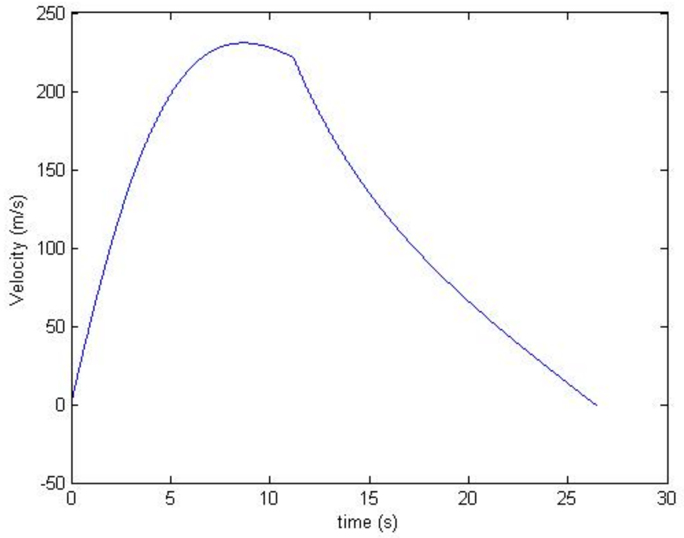


Figure 6 Velocity During Launch Escape

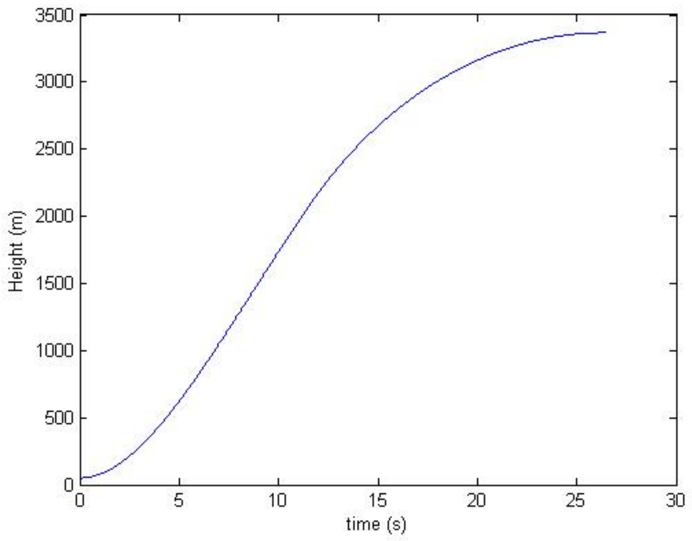


Figure 7 Vehicle Trajectory

21.1.2.2 Tower Jettison Motor

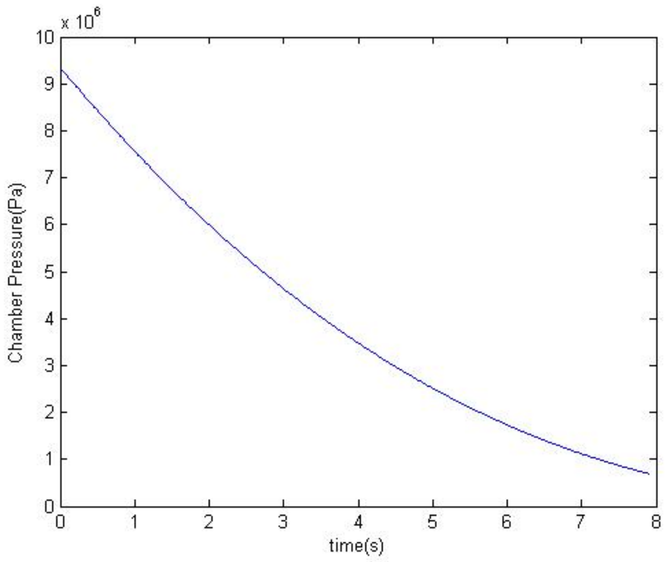


Figure 8 TJM Chamber Pressure

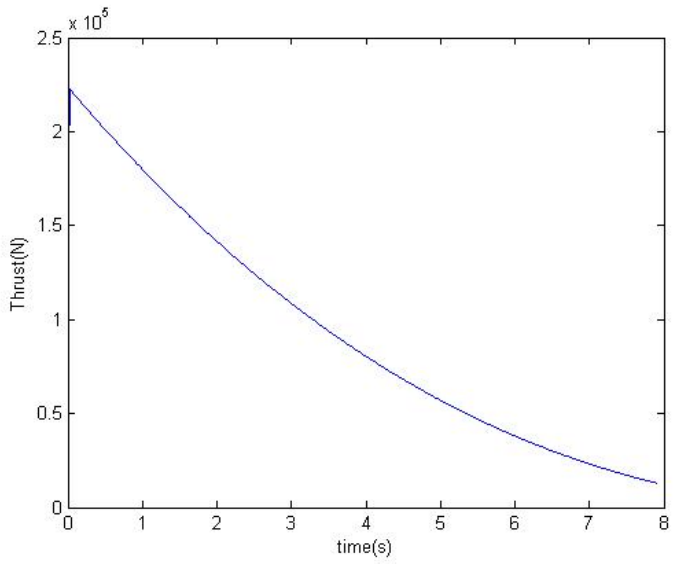


Figure 9 TJM Thrust

21.1.2.3 Pitch Control Motor

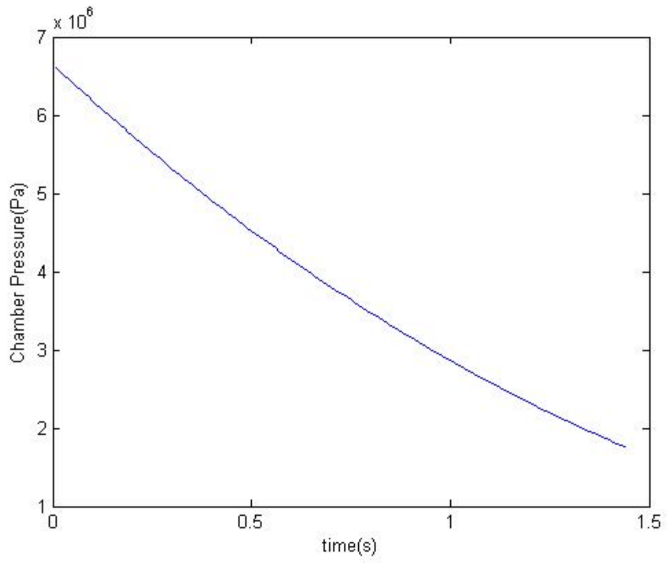


Figure 10 PCM Chamber Pressure

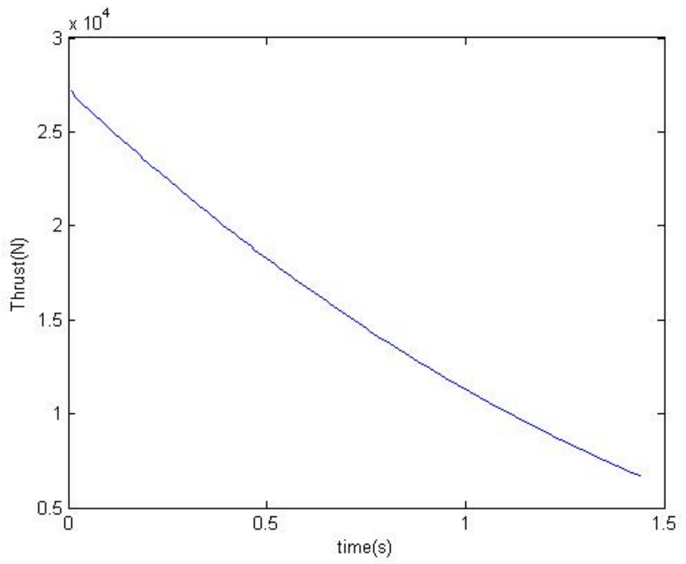


Figure 11 PCM Thrust

21.1.3 Lessons Learned

We chose the internal burning star configuration for the propellant grain shape when we designed the Launch Escape System and we modeled the arms of the star as triangles, sometimes with sharp points at the ends. In reality, our model may not be as accurate as it could be since during burning, the sharp points may be broken off. It might be wise to instead, design the arms as trapezoids as that would eliminate the sharp edges and also keep the burn area closer to constant.

The LES would also benefit from a more detailed trajectory model that takes into account the moment applied by the Pitch Control Motor. For this project, we modeled a simple approximation of a vertical ascent trajectory ignoring the effect of the Pitch Control Motor. Vertical ascent is obviously not an acceptable actual trajectory as it would either leave the Crew Module falling back onto the launch vehicle or (at low altitude) the launch vehicle could catch back up to the escaping Crew Module. The reason the launch vehicle could catch back up to the Crew Module if the event occurs at low enough altitude is that at that low altitude, engine shutoff would increase the risk of the launch vehicle falling into a populated area.

21.1.4 Rocket Code

The following is a listing of the code that we use for modeling the Launch Escape Motor in the Launch Escape System. We use the same code to model the tower jettison and pitch control motors but with different values input for the variables.

21.1.4.1 getPropellantProperties.m

This code returns values for properties of the chosen propellant. These values are from Space Propulsion Analysis and Design [1] table 6.9.

```
function [rho,cstar,Tc,ab,nb] = getPropellantProperties
rho = 1800; %kg/m^3
cstar = 1527; %m/s
Tc = 3396; %K
ab = 0.399;
nb = 0.30;
return
```

21.1.4.2 run.m

This is the primary code for calculating chamber conditions and also calls run2.m.

```
clc
clear all
close all

want_traj = 0;            % set equal to 1 if you want to run traj.m, 0 if not

g = 9.81; %m/s^2

wedge_angle_factor=1.9; % used to adjust the size of each point-Must be greater than 1
n = 16;                 %number of points in star configuration
Rp = .35; %m            %initial interior port radius
L = 5.5; %m             %Chamber Length
Ru = 8.31447;
ER = 3.85;
PR = 2;                 %Port to Throat Area Ratio (must be at least 2)
m_vehicle = 11900; %kg
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Returns Properties of TP-H-3340 Propellant
%   18%Al, 71%Ap, 11%HTPB
%   rhoP = solid propellant density
%   cstar= Cstar value taken from Space Propulsion Analysis & Design
%   Tc   = average chamber temperature also taken from SPAD (pg 330)
%   ab   = burn rate coefficient from SPAD
%   nb   = burn rate exponent from SPAD
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[rhoP,cstar,Tc,ab,nb] = getPropellantProperties;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Finds initial geometry of port X-section
%   phi = angle of sector enclosing each point of star
%   th  = angle of wedge of propellant making up the point of star
%   b   = length of the base of the triangle describing a wedge of propellant
%   S   = arclength of the arc between under a wedge of propellant
%   Ap  = Port Area
%   At  = Throat Area
%   Per = Perimeter of port used to get burn area
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
phi = 2*pi/n;
b = 2*Rp*sin(phi/2);
th = phi*wedge_angle_factor;

S = phi*Rp;
```

```
t = S/2;

Asec = (pi*Rp^2)/n;
h_sectri = sqrt(Rp^2-(b/2)^2);
AsecTri = 0.5*b*h_sectri;
Asliver = Asec-AsecTri;

h_wedge = b/(2*tan(th/2));
Awedge = 0.5*b*h_wedge;

Ap = n*(Asec-(Awedge+Asliver));
At = Ap/PR;

Per = 2*n*sqrt(h_wedge^2 + (b/2)^2);

Ab = Per*L;

Pc = 4.2947e+006; %Pa
r_cm = ab*(Pc/(10^6));
r = r_cm/100;
dt = 0.01;%s
dw = 0;
time = 0;
wedges = 1;
m = 0;
i = 1;
mass_prop = 0;
gam = 0.0046820381*log(Pc/10^6) + 1.1211390929;
MW = (0.3894458877*log(Pc/10^6) + 28.9075842531)/1000;
Tc = 113.5739568711*log(Pc/10^6) + 3372.3328543337;
mdot_total = r*Per*L*rhoP;

while S(i)>0
    time(i+1) = time(i)+dt;
    r_cm = ab*(Pc(i)/(10^6))^nb;
    r(i+1) = r_cm/100;%m/s
    dw(i) = r(i)*dt;
    Rp(i+1)= dw(i)+Rp(i);
    if wedges
        S(i+1) = S(i) - 2*dw(i);
        if S(i+1) < 0
            wedges = 0;
            S(i+1) = 0;
        end %if
        phi(i+1) = S(i+1)/Rp(i+1);
        b(i+1) = 2*Rp(i+1)*sin(phi(i+1)/2);
        if wedges
            N_new = 2*pi/phi(i+1); %for wedge size only, number of wedges is same
            Asec_new = (pi*Rp(i+1)^2)/N_new;
            h_sectri_new = sqrt(Rp(i+1)^2-(b(i+1)/2)^2);
            Asectri_new = 0.5*b(i+1)*h_sectri_new;
            Asliver_new = Asec_new-Asectri_new;

            h_wedge(i+1) = b(i+1)/(2*tan(th/2));
            Awedge_new = 0.5*b(i+1)*h_wedge(i+1);
        else
            Awedge_new = 0;
            Asliver_new = 0;
            S(i+1) = 0;
            h_wedge(i+1) = 0;
            b(i+1) = 0;
        end %if

        Ap(i+1) = pi*Rp(i+1)^2-n*(Awedge_new + Asliver_new);
        Per(i+1) = 2*pi*Rp(i+1) - n*S(i+1) + 2*n*sqrt(h_wedge(i+1)^2 + (b(i+1)/2)^2);
    else
        S(i+1) = 0;
        h_wedge(i+1) = 0;
        b(i+1) = 0;
        Ap(i+1) = pi*Rp(i+1)^2;
        Per(i+1) = 2*pi*Rp(i+1);
    end %if

    Ab(i+1)=Per(i+1)*L;

    mdot_total(i+1) = r(i+1)*Per(i+1)*L*rhoP;
    mass_prop = mass_prop + r(i)*dt*Per(i)*L*rhoP;
    m_vehicle(i+1) = m_vehicle(i) - mdot_total(i+1)*dt;

    Pc(i+1) = (10^6)*((ab/100)*rhoP*Ab(i+1)*cstar*(1/10^6)/At)^(1/(1-nb));

    gam(i+1) = 0.0046820381*log(Pc(i+1)) + 1.1211390929;
    MW(i+1) = (0.3894458877*log(Pc(i+1)) + 28.9075842531)/1000;
    Tc(i+1) = 113.5739568711*log(Pc(i+1)/10^6) + 3372.3328543337;
    i = i+1;
end %while

mass_prop
dmavg = mean(mdot_total);
figure(1)
plot(time,Pc)
xlabel('time(s)')
ylabel('Chamber Pressure(Pa)')

figure;plot(h_wedge)
run2
```

21.1.4.3 run2.m

This code uses the values calculated in run.m to calculate nozzle exit conditions and thrust attained by the motor. If the variable want_traj is set to a number other than zero, run2.m will also call traj.m.

```
%Gets Exit Conditions after run.m has been run
clc
close all
Ae = ER*At;
Pa = 101325;%Pa
for i = 1:length(Pc)
    i=300;
    Ru = 8.31441;

    Me2(i) = 1.1;
    guess=(1/Me2(i))*((2/(gam(i)+1))*(1+((gam(i)-1)/2)*Me2(i)^2))^((gam(i)+1)/ (2*gam(i)-2));
    count = 0;

    while abs(guess-ER)>.0001
        if abs(guess-ER)>.1
            if guess<ER
                Me2(i) = Me2(i)+.01;
                count = count+1;
            else
                Me2(i) = Me2(i)-.01;
                count = count+1;
            end %if
        elseif abs(guess-ER)>.01
            if guess<ER
                Me2(i) = Me2(i)+.001;
                count = count+1;
            else
                Me2(i) = Me2(i)-.001;
                count = count+1;
            end %if
        elseif abs(guess-ER)>.001
            if guess<ER
                Me2(i) = Me2(i)+.0001;
                count = count+1;
            else
                Me2(i) = Me2(i)-.0001;
                count = count+1;
            end %if
        else
            if guess<ER
                Me2(i) = Me2(i)+.00001;
                count = count+1;
            else
                Me2(i) = Me2(i)-.00001;
                count = count+1;
            end %if
        end
        guess=(1/Me2(i))*((2/(gam(i)+1))*(1+((gam(i)-1)/2)*Me2(i)^2))^((gam(i)+1)/ (2*gam(i)-2));
    end %while

    Pe2(i) = Pc(i)*(1+((gam(i)-1)/2)*Me2(i)^2)^(gam(i)/(1-gam(i)));

    Ve2(i) = sqrt((2*gam(i)*Ru*Tc(i)/((gam(i)-1)*MW(i))) * (1-(Pe2(i)/Pc(i))^((gam(i)-1)/gam(i))));
    F(i) = (mdot_total(i)*Ve2(i)+(Pe2(i)-Pa)*Ae)*cos(35*pi/180);
    accel(i) = (F(i)/m_vehicle(i))/9.81;
end
At;
Pe2(end)-Pa
m_vehicle(end)
mass_prop
plot(time(2:end),F(2:end))
xlabel('time(s)')
ylabel('Thrust(N)')

figure
plot(time(2:end),Pc(2:end))
xlabel('time(s)')
ylabel('Chamber Pressure(Pa)')

figure
plot(time(2:end),accel(2:end))
xlabel('time(s)')
ylabel('Acceleration (g)')

if want_traj
    traj
end
```

21.1.4.4 traj.m

This program calculates a very basic approximation of the LES trajectory assuming vertical ascent.

```
clc
close all

% specific heat ratio
gam_atm = 1.4;
% specific gas constant
R_atm = 287; %J/(kg*K)

rad_vehicle = 2.5; %m
A_vehicle = pi*rad_vehicle^2; %m^2

h = 50; %m
v(1) = 0;


aF = F./m_vehicle;


for i = 1:length(aF)
    if i ~= 1
        v(i) = v(i-1) + a_total(i-1)*dt;
        h(i) = h(i-1) + v(i-1)*dt;
    end
    [rho_atm,tK,Press_atm]= atmosMetric(h(i));

    % Speed of Sound at altitude
    a_atm = sqrt(gam_atm*R_atm*tK);
    % Mach Number at given altitude and velocity
    Mach(i) = norm(v(i))/a_atm;
    % Drag Coefficient
    CD = dragRocket(Mach(i));
    Drag(i) = .5*rho_atm*v(i)^2*A_vehicle*CD;
    aD(i) = Drag(i)/m_vehicle(i);

    a_total(i) = aF(i)-aD(i)-g; %m/s^2

end %for
while v(i-1)>0
    [rho_atm,tK,Press_atm]= atmosMetric(h(i-1));

    % Speed of Sound at altitude
    a_atm = sqrt(gam_atm*R_atm*tK);
    % Mach Number at given altitude and velocity
    Mach(i) = norm(v(i-1))/a_atm;
    % Drag Coefficient
    CD = dragRocket(Mach(i));
    Drag(i) = .5*rho_atm*v(i-1)^2*A_vehicle*CD;
    m_vehicle(i) = m_vehicle(i-1);
    aD(i) = Drag(i)/m_vehicle(i);

    a_total(i) = -aD(i)-g; %m/s^2
    time(i) = time(i-1)+dt;
    aF(i) = 0;
    F(i) = 0;
    v(i) = v(i-1) + a_total(i-1)*dt;
    h(i) = h(i-1) + v(i-1)*dt;
    i = i +1;
end


plot(time(2:end),Drag(2:end))
hold on
plot(time(2:end),F(2:end),'-r')
xlabel('time (s)')
ylabel('Force (N)')
legend('Drag Force', 'Thrust')

figure
plot(time(2:end),a_total(2:end)/g)
xlabel('time (s)')
ylabel('Acceleration (g)')

figure
plot(time(2:end),v(2:end))
xlabel('time (s)')
ylabel('Velocity (m/s)')

figure
plot(time(2:end),h(2:end))
xlabel('time (s)')
ylabel('Height (m)')
```

21.1.4.5 Other subroutines used

Also used in this code were the functions atmosMetric.m and dragRocket.m, both originally written for the design of the ELV by Phil Spindler. These functions were used to calculate atmospheric conditions and the drag coefficient respectively.

atmosMetric.m:

```
function [densM,tK,pressureM]= atmosMetric(altMeter)
%
alt=altMeter*3.281;
%% convert altitude to feet
%         results = atmos(altitude,output)
```

```
%
%        Units are Calculated in English and converted to metric at the end
%
%         density      =  slugs / ft^3
%
%              temperature  =  degrees Rankine
%
%         pressure     =  density * R * Temp = lb / ft^2
%
%
%                        R = 1716
%

    n = length(alt);
    t = zeros(1,n);
            dens = zeros(size(t));

            r = 1716.0;
            gc = 32.174;

    for i = 1:n

            if alt(i) < 35000

                    a       = -0.003566;
                    t(i)    = a * alt(i) + 518.69;
                    dens(i) = (6.38e-15) .* t(i) .^ (-(1 + (gc / (a * r))));

                    elseif alt(i) < 65578

                    t(i)    = 390.0;
                    dens(i) = (0.004) .* exp(-(gc * alt(i) ./ (r * t(i))));

                    elseif alt(i) < 104924

                    a       = 0.0005208;
                    t(i)    = a * alt(i) + 357.6;
                    dens(i) = 0.004491 .* exp(-alt(i) ./ 20369.0);

                    elseif alt(i) < 154107

                    a       = 0.001629;
                    t(i)    = a * alt(i) + 245.26;
                    dens(i) = (1.572e+28) .* t(i) .^ (-(1.0 + (gc / (a * r))));

                    elseif alt(i) < 170502

                    t(i)    = 487.8;
                    dens(i) = (0.001075) .* exp(-(gc * alt(i) ./ (r * t(i))));

                    elseif alt(i) < 235700

                    a       = -0.001591;
                    t(i)    = a * alt(i) + 749.89;
                    dens(i) = (1.825e-35) .* t(i) .^ (-(1 + (gc / (a * r))));

                    elseif alt(i) < 278705

                    a       = -0.001;
                    t(i)    = a * alt(i) + 611.7;
                    dens(i) = 0.02542 .* exp(-alt(i) ./ 19311.0);

                    elseif alt(i) < 295099

                    t(i)    = 321;
                    dens(i) = 0.18485 .* exp(-(gc * alt(i)) ./ (r * t(i)));

                    elseif alt(i) < 327888

                    a       = 0.000884;
                    t(i)    = a * alt(i) + 60.13;
                    dens(i) = (3.09e+47) .* t(i) .^ (-(1.0 + (gc / (a * r))));

                    elseif alt(i) < 360777

                    a       = 0.0025;
                    t(i)    = a * alt(i) - 469.72;
                    dens(i) = (4.2335e12) .* t(i) .^ (-(1.0 + (gc / (a * r))));

                    elseif alt < 400000

                    a       = 0.00659;
                    t(i)    = a * alt(i) - 1944.86;
                    dens(i) = 2.2788 .* t(i) .^ (-(1.0 + (gc / (a * r))));

                    else

                    dens(i) = 0;
                    t(i)    = 0;

                    end

    end

            pressure = r.*dens.*t;



densM=dens*515.4;
% density in kg/m^3
tK=t*273/492;
%
pressureM=pressure*47.88;
% pressure in n/m^2
```

dragRocket.m:

function Cd=dragRocket(M)

```
% calculate the drag of a rocket based on frontal area

    if M <= .8
         Cd = .4;
    elseif M <= 1.5
         Cd = .8570.*M - .2857;
    elseif  M > 1.5
        Cd = .55 + .45.*exp( - .9.*(M - 1.5));
    end
```

[1] Humble, Henry, and Larson. Space Propulsion Analysis and Design. New York: The McGraw-Hill Companies, Inc., 1995.

22 Husein, Ezdehar

22.1 Artificial Gravity

Author: Ezdehar Husein

22.1.1 Topic Description

One method of producing artificial gravity in a spacecraft involves constantly rotating the vehicle about an axis at a fixed speed. This results in a radial gravitational force similar to that experienced on earth. A range of gravitational forces can be produced by varying different parameters discussed in this section.

22.1.2 Theory and Assumptions

This section produces calculations based on design decisions that were made at the time this analysis took place. When the proposition of creating artificial gravity on the Crew Transport Vehicle (CTV) first came up, the CTV was proposed as a "fly-swatter" design. As such, crew quarters at a distance *r* from an axis of rotation would have a counterweight equal in mass on the opposite end of the vehicle.

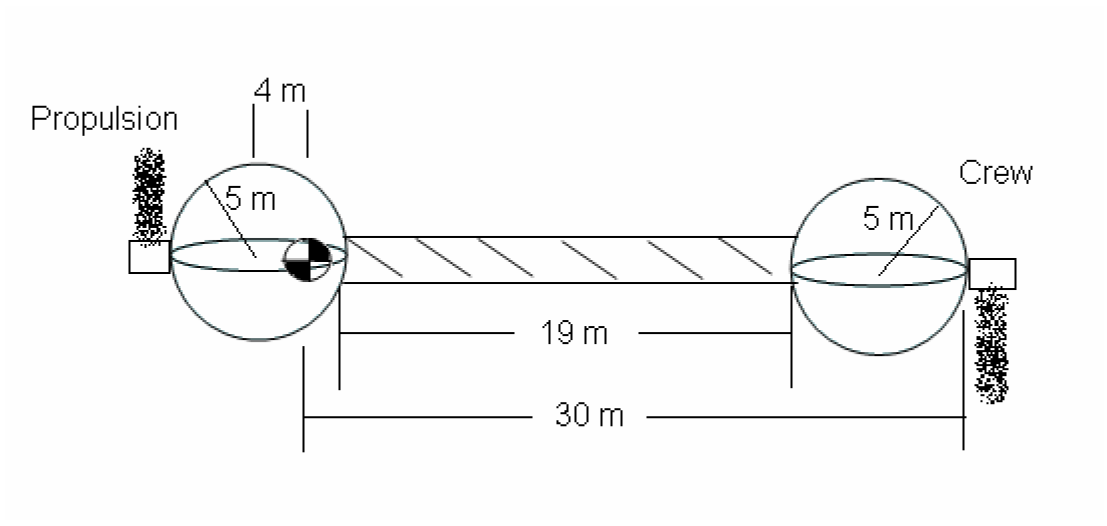


Figure 1-1: Crew Transport Vehicle "Fly-Swatter" Design (Ben Parkinson)

22.1.3 Analysis

Assuming that a finite material for the axis constrains us to a maximum of 45 meters and that we must produce a 1-g gravitational force, we begin the analysis to find an optimum combination of the following parameters:

Parameters:

r : radial position of crew quarters to center of rotation (meters)

Ω: angular rate of rotation of spacecraft (rev/ min)

V_t: tangential speed of spacecraft (meters/sec)

V_p: Walking speed of crewmembers (meters/sec)

22.1.3.1 Mobility Constraints for Expected Range of Motion on CTV

Based on tolerances outlined in Theodore Hall's *Artificial Gravity and The Architecture of Orbital Habitat [1]*, we performed an analysis of what constraints existed for expected motions on the CTV. This is dependent on the configuration of the crew quarters and any other activities the crew was expected to perform while in transport.

22.1.3.2 Motion Parallel to Axis of Rotation

For any motions that were parallel to the axis of rotation, such as walking along the floor of the spacecraft, we sought to minimize the ratio of Coriolis Acceleration to Centripetal acceleration. We aimed to obtain a ratio of less than 20% to mitigate nausea, dizziness, and the potential for blackouts of crewmembers.

Parameters:

$\Omega = 5.44$ rev/min

r = 15 to 45 meters

$V_t = 17.1$ meters/sec

$A_{cent} = \Omega^2 r$ (1)

$A_{cor} = 2*V_p$ (2)

$A_{cent} / A_{cor} = 2 * V_p / V_t < 20\ \%$ (3)

22.1.3.3 Climbing

Based on Hall's recommendations, we aimed to keep the value of A_{cor} to less than 30% of A_{cent} for all climbing motions of the crew. Climbing was needed to move from multiple floors of the crew quarters on the CTV.

$A_{cor} < .3 * A_{cent}$ (4)

22.1.3.4 Lifting

Based on Hall's recommendations, we aimed to keep the value of A_{cor} to less than 25% of A_{cent} for all lifting motions of the crew.

$A_{cor} < .25 * A_{cent}$ (5)

22.1.4 Conclusions

Using a range of possible radius values from 15 to 45, we produced the following table.

Results of Expected Range of Motion for CTV

r (m)	V_t (m/s)	v (m/s) walking	v (m/s) Climbing	v (m/s) Lifting
15	8.55	.855	1.283	1.07
20	11.4	1.14	1.71	1.425
25	14.25	1.425	2.138	1.78
30	17.1	1.71	2.565	2.13
35	19.95	1.995	2.993	2.494
40	22.8	2.28	3.42	2.85
45	25.62	2.565	3.843	3.203

Table 1

For the requirements of Project Legend, the following criteria were set on the expected range of motion for crewmembers on the CTV:

Optimum Tolerances for Motion Range

r (m)	Ω (rev/min)	V_t (m/s)	v (m/s) walking	v (m/s) Climbing	v (m/s) Lifting
30	5.44	17.1	1.7	2.55	2.13

Table 2

22.2 Radiation Effects on Plant Life

22.2.1 Topic Description

Project Legend requires that a Mars Habitat Vehicle (MHV) be sent to the surface of Mars before a manned spacecraft arrives. We propose the MHV to either 1) have food shipped as payload with the MHV or 2) have food stored for only the time of the manned surface-stay. The difference in both options is the necessary shelf-life of any food used for human consumption. Option (1) requires a shelf-life of at least 10 years, while option (2) was limited to a 500 day surface stay at the time this analysis took place.

22.2.2 Theory and Assumptions

For the purpose of this analysis, the mission on the Martian surface will be 500 days in duration. Also, only galactic cosmic rays are considered while solar flares are disregarded. We assume to use 20 cm of Polyethylene shielding on the MHV and base all calculations on an average dose equivalent, H, of 12 REM/year. We will thus be using a quality factor, Q, equal to 2.9 that reflects the galactic cosmic rays and energy type.

22.2.3 Analysis

We perform the following calculations using the following radiation units:

Absorbed Dose (D) : SI Unit – Gray (Gy) = J/Kg = 100 rad

Dose Equivalent(H) : SI Unit – Sievert (Sv) = 100 REM

Quality Factor (Q) : based on radiation type and its energy

Based On: 500 day Mission on Martian Surface

Dose Equivalent (H) = 12 REM/year = .12 Sv/yr

Quality Factor (Q) = 2.9

Relation: H = Q * D (1)

Absorbed Dose (D) = 4.14 rad/yr (2)

Total Absorbed Dose over 500 days on Martian Surface: **5.67 rad**

This calculation demonstrates that for a 500 day period on the Martian surface, we are only subjecting plant and animal life to approximately 5.67 rad. To gain a perspective on what this means, the following table summarizes tolerance levels of various plant types. The LD – 50 tells us what magnitude of absorbed dosages would be necessary to kill 50% of the grown crops.

Summary of Plants and Radiation Tolerances

Plant	LD-50 in RADS
Cabbage, Spinach	7000
Onions	1000
Oats	1650
Barley, rye, wheat, corn	2150
Potatoes	6000
Tomatoes	7500

Table 1

We can easily see that the magnitude of absorbed radiation based on galactic cosmic rays is not a major factor on the MHV for plant life. The tolerances of plant life far exceed the expected radiation on the Martian surface for a 500 day period.

References

1. Hall, Theodore W., “Artificial Gravity and The Architecture of Orbital Habitats,”, 20 March 1997.
2. Mars radiation: http://helios.ecn.purdue.edu/~tatjanaj/NUCL497_2002/Report 12.pdf
3. Radiation Units: http://www.ccohs.ca/oshanswers/phys_agents/ionizing.html

23 Jaron, Jackie

23.1.1 Jaron – Appendix – Ascent and Recovery Vehicle Heat Shield Design Process

Author: Jackie Jaron

Contributors: Jackie Jaron & Hafid Long

23.1.1.1 Purpose

The purpose of this appendix is to illustrate the method we use to size the Ascent and Recovery Vehicle (ARV) heat shield for atmospheric re-entry.

23.1.1.2 Methods

We use the ballistic re-entry velocity profile supplied by Angela Long of the Aerodynamics Group. This profile consists of a density and velocity at each discrete point. This array loads into ARV_Heating.m and the program calculates the heating rate of the vehicle as it re-enters the Earth. The heat flux values are then saved as an array to a .qw file for later use by testsoddit.m (see **Error! Reference source not found.** for flow chart).

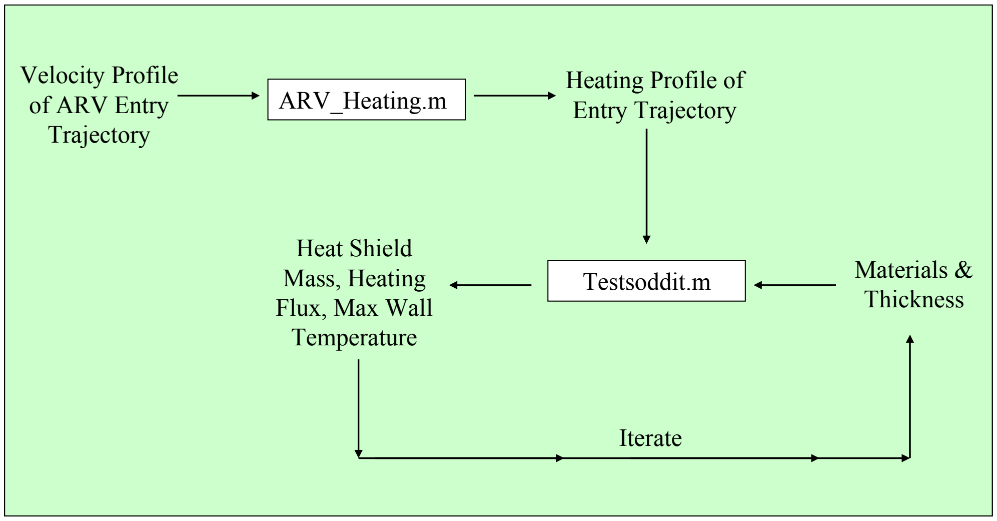

Velocity Profile of ARV Entry Trajectory
ARV_Heating.m
Heating Profile of Entry Trajectory
Heat Shield Mass, Heating Flux, Max Wall Temperature
Testsoddit.m
Materials & Thickness
Iterate

Figure 23-1 Flowchart of Sizing Process

The MATLAB code, testsoddit.m, creates an input file for SODDIT to use, runs the SODDIT program and then plots the output data. Testsoddit.m requires the user to input the TPS materials, their respective thicknesses and the heat flux profile generated by ARV_Heating.m. Both programs, testsoddit.m and SODDIT, do not optimize the heat shield size or mass and thus it is the user's job to vary the thickness of each material to obtain the lowest mass heat shield for a given heating profile.

23.1.1.3 ARV Stagnation Heating Code

23.1.1.3.1 Description/Inputs/Outputs

The ARV_Heating.m code outputs a heating profile and preliminary mass estimate for the ARV assuming an Earth ballistic re-entry. The code requires that a density and velocity profile (profile.txt) be loaded into the MATLAB workspace. The diameter (*d*) of the ARV leading blunt end is also an input to the code.

The ARV_Heating.m file assumes the entire blunt end experiences stagnation heating during re-entry. The following plot is a heating profile for the ARV during the worst case re-entry scenario (launch window 2).

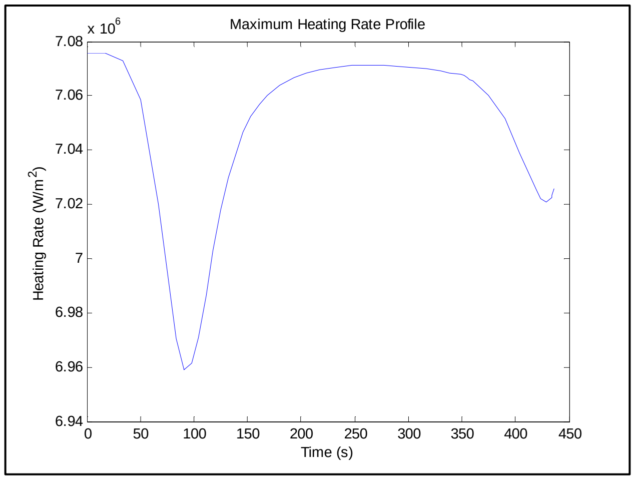


Figure 23-2 Heating Rate Profile of ARV Worst Case Entry

23.1.1.3.2 ARV_Heating.m MATLAB Code

```
% AAE 450
% By: Hafid A. Long
% Modified by J. Jaron 02/28/2005
% Code for weight estimation of heat shield for ARV.
%
%   Blunt body configuration is used for the Ascent & Return Vehicle.
%   For approximation purposes, the part of vehicle that will be
%   exposed to a majority of the atmospheric heating will be considered
%   as a flat plate. A fully turbulent flow and a large angles of attack
%   will be assumed for conservative approximation of the heat transfer.
%   The high-performing and commonly used thermal protection material of
%   SLA-561 (ablators) will be used.
echo off
clear all; clc; %close all; clc;

% Define general constant
N = 0.8;        % Exponent for heat transfer correlation
M1 = 3.37;      % Exponent for heat transfer correlation [V <= 3962 m/s]
```

```
M2 = 3.7;       % Exponent for heat transfer correlation [V > 3962 m/s]
k1 = 3.35e-8;   % Constant for heat transfer correlation
k2 = 2.20e-9;   % Constant for heat transfer correlation
SBC = 5.6696e-8;% Stefan-Boltzmann constant (W/(m^2-deg. K^-4))

% Define vehicle geometry
d = 4.5;         % diameter of vehicle (m)
rn = d/2;      % Nose radius of vehicle (m)

% Define flight condition
AOA = 0*pi/180;% Angle of attack (rad.)

% Define material constant (SLA-561)
d_m = 14.5/2.205/0.3043^3;      % Density (kg/m^3)
hv = 54.1e6;                    % Effective heat of ablation (J/kg)
E = 0.70;                       % Emissitivity

% Load external trajectory data
load profile.txt;
t = profile(:,1);              % Trajectory time (s)
alt = profile(:,2);            % Altitude (m)
V = profile(:,3);               % Trajectory velocity (m/s)
d_a = profile(:,7);            % Atmospheric density along trajectory (kg/m^3)

%  Material properties (SLA-561)
load AP.txt;
temp = AP(:,1);
Cp = AP(:,2);

% plot trajectory profile
figure; plot(t,alt);
xlabel('Time (s)'); ylabel('Altitude (m)');
title('ARV Trajectory Profile');
grid on

% Plot trajectory velocity profile
figure; plot(t,V);
xlabel('Time (s)'); ylabel('Velocity (m/s)');
title('Trajectory Velocity Profile');

% Determine location of maximum heating rate,max heating at center of heat shield
% Compute stagnation point heating rate at nose of Mars Lander Vehicle

Tw = 500*9/5*ones(length(t),1);
Tw_test = 0;
for n = 1:1:length(t)
    while abs(Tw_test-Tw(n)) >= 5
        Tw_test = Tw(n);
        Cpw = interp1(temp,Cp,Tw,'spline');
        qdot = (1.83e-8).*sqrt(d_a(n)./(d./2)).*V(n).^3.*(1-(Cpw.*Tw(n)./...
            (0.5.*V(n).^2)));
        qdot = qdot*100^2;
        Tw(n,1) = (qdot(n,1)/(E*SBC))^(1/4);
    end
end

% Plot results
figure; plot(t,qdot);
xlabel('Time (s)'); ylabel('Heating Rate (W/m^2)');
title('Heating Rate Profile At Nose Of Mars Lander Vehicle');

figure; plot(t,Tw);
xlabel('Time (s)'); ylabel('Wall Temperature (^oK)');
title('Wall Temperature Profile At Nose of Mars Lander Vehicle');

% Compute maximum total heat load (J)
A = pi*4^2/4.5;     % Surface Area of a hemisphere

% A is the total area of the ablative material
Q = trapz(t,qdot)*A % computes integral of qdot wrt t
```

```
% Compute mass of heat shield for ARV
m = Q/hv

% Compute thickness of heat shield for ARV
b = Q/(d_m*hv*A)

% Compute maximum heating rate of the ARV
Tw_max = 1060*9/5*ones(length(qdot),1);

% % Plot results
figure; plot(t,qdot);
xlabel('Time (s)'); ylabel('Heating Rate (W/m^2)');
title('Maximum Heating Rate Profile');

figure; plot(t,Tw_max);
xlabel('Time (s)'); ylabel('Wall Temperature (^oK)');
title('Maximum Wall Temperature Profile');

% Approximate mass to be 60% of total heat shield mass
m_structure = 0.6*m

% Compute total mass of thermal protection system
mtotal = m + m_structure

fid = fopen('jj.qw','w');
for i = 1:length(t)
    fprintf(fid, '%5.2f\t\t%6.4e\n',t(i),qdot(i));
end
fclose(fid);
```

23.1.1.4 SODDIT Input File, SODDIT Execution and Output Code

23.1.1.4.1 Description/Inputs/Outputs

The testsoddit.m MATLAB file is a function and creates an input file for SODDIT to use, allows the user to execute the SODDIT FORTRAN program and generates plots from the output data.

The testsoddit.m MATLAB function requires the following inputs (for an example input line, type 'help testsoddit' in the MATLAB command window) :

- *Materials*: The names of the materials used for each layer of the TPS in the order they appear (ie. outer most layer to inner most layer). The input material name must match the naming convention used in the matprop.txt file.
- *Thickness:* An array of each material thickness. The user iterates on the thicknesses to optimize the heat shield mass.
- *Filename:* The file name, which contains the heating data calculated in ARV_Heating.m. This filename should have a *.qw* file extension.

The testsoddit.m file outputs the following parameters:

- *Density* of each material

- *Mass* of the each material
- *Maximum wall temperature (Tmaxwall)* is the maximum temperature any one material experiences during the entire re-entry phase.
- *Heat dissipated (heatout)* is the total amount of heat dissipated by the heat shield.

In addition, testsoddit.m generates two plots, which are beneficial for the user to analyze for shield sizing and optimization.

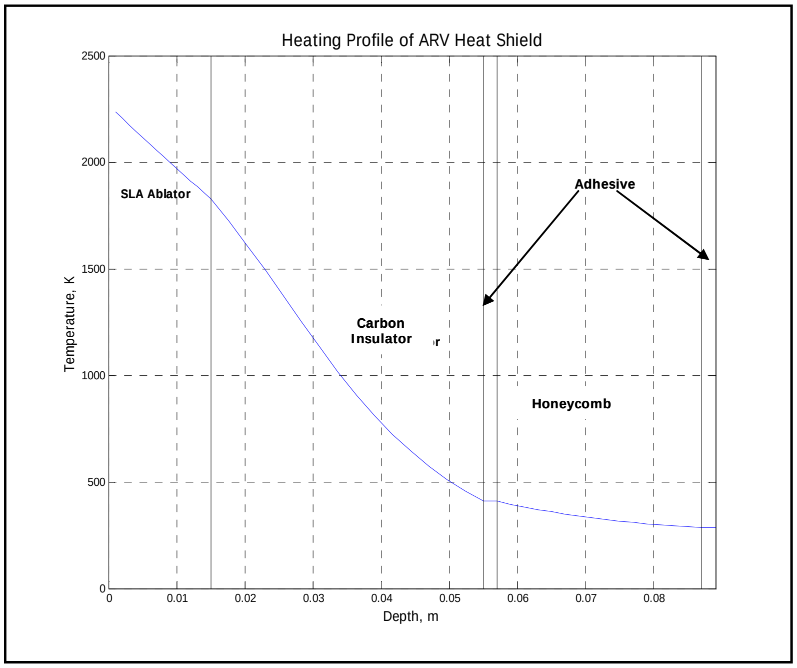


Figure 23-3 Heating Profile of ARV Heat Shield

Figure 23-3 shows the heating profile of the heat shield while Figure 23-4 allows the user to see if the heating of each material remains below the maximum allowable temperature.

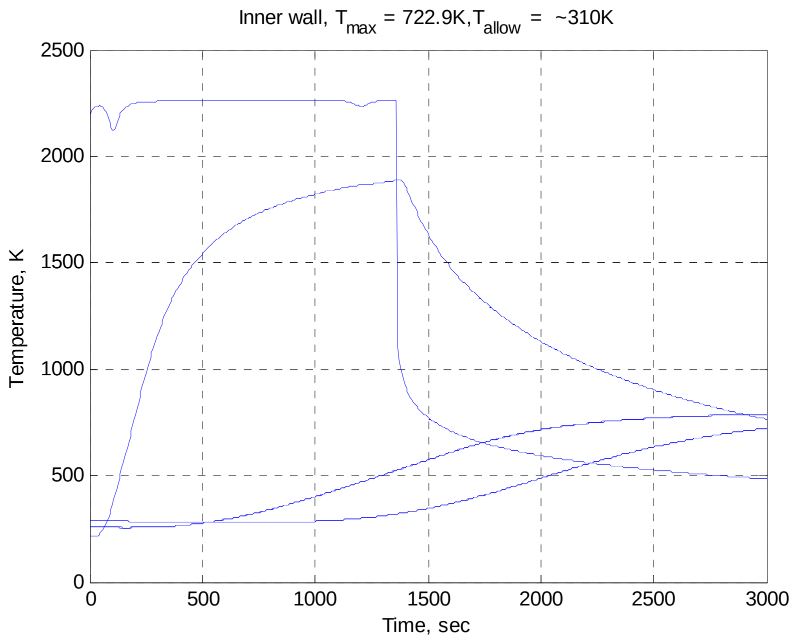


Figure 23-4 Inner Wall Temperature of Each Heat Shield Material for the Duration of Re-Entry

23.1.1.4.2 Testsoddit.m MATLAB Code

```
function [density, masses, Tmaxwall,heatout] = testsoddit(mat,thick,filename,hindex)
% testsoddit({'Teflon','Carbon Insulator','Composite'},[1e-3 4e-2 2e-3],'test',1)
% mat = {'Teflon','Carbon Insulator','Composite'};
% thick = [1e-3 4e-2 2e-3];
% filename = 'Test';
% hindex = 1;

comp = 1;

soddit_path = '/home/bfrc/a/aae450s/soddit/';
current_directory = '/home/bfrc/a/jaron/WTF/';

%If top layer is an ablator, then "Ablator" must be part of its name to model ablation

% Problem constants
n = 15;                         % Number of elements per material
timemax = 3000;                 % Maximum time of simulation
n_output = 500;                 % Number of output time values

% Geometry
nmat = length(thick);           % Number of layers modelled
node = nmat*n+1;                % Total number of "nodes"
chunk = sum(thick);             % Total thickness
dx = thick/n;                   % Thickness of element per material

% Form vector of temperature distribution across thickness
tinit = [200 290];                      % 0 & thickness initial temp, linearized across length
delT = thick/chunk*(tinit(1)-tinit(2)); % Change in temperature across each individual layer
Tn = tinit(1)-cumsum(delT);             % Temperature at the bottom of each layer
Tn = [tinit(1) Tn(1:nmat-1)];           % Temperature at the top of each layer
Tdist = tinit(1);                       % Initialize Temperature distribution vector
for mc = 1:nmat
    Tdist_mc = Tn(mc)-delT(mc)/n*[1:n]; % Find temperature at each node for a given layer
    Tdist = [Tdist Tdist_mc];           % Append to the vector for use in Block six
```

```
end

disp('Creating input file');

% Create input file
inputfname = [filename '.inp'];
infile = fopen(inputfname,'w');

% Set flags
prt=1;                          % 1 to write to print file, 0 not to

%  If top layer is an ablator, specify the proper flag, and create FRACT for use in block 4
if findstr('Ablator',mat{1});
    abl = 3;
else;
    abl = 0;
end
if abl == 3;
    fract = [.001   0.05];
else;
    fract = [];
end

flags=zeros(1,80);              % All other flags are zero
flags(4)=abl;
flags(5)=prt;
flags(16)=1;
flags(2)=3;

% Get heating data
heat = load([filename '.qw']);                                                  % Load heating
data from file
heat = [heat(:,1) zeros(size(heat,1),1) heat(:,1+hindex) zeros(size(heat,1),1)];    % Format for use
in SODDIT
heatout(1) = max(heat(:,3));
heatout(2) = trapz(heat(:,1),heat(:,3));

%  Condense heating data by removing long series of zeros (SODDIT will interpolate)
i = 2;                          % Initialize

while i< length(heat)-1
    % If the heating value is zero for three successive points then remove the middle one.
    % This leaves the beginning and end of long stretches of zeros
    if (heat(i-1,3) == 0 & heat(i,3) == 0 & heat(i+1,3) == 0)
        heat(i,:)=[];
        i=i-1;
    end
    i = i+1;
end

if heat(end,1)>timemax
    timemax=heat(end,1);                        % Run for longer of user specified and file
specified time values
else
    % Add q=0 at end to allow for equilibrium
    heat = [heat;heat(end,1)+1 0 0 0;timemax 0 0 0];    % Append
end

mp=fopen('matprop');            % Material property file

% Write to input file

% Block 1
fprintf(infile,'%s \n',...
    '**block 1 control flags',...
    'c23456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789',...
    sprintf('%1i',flags));

% Block 2

%  Write material names and thicknesses
```

```
fprintf(infile,'  %s, ',mat{:});fprintf(infile,'\n');
fprintf(infile,'  %f, ',thick);
fprintf(infile,'\n  Heating with known time dependant flux\n');

% Block 3
fprintf(infile,'%s \n',...
    '**block 3 print times and print intervals, maximum of 20 and 19 resp',...
    ['  0.0       '  num2str(timemax) ],...
    ['  ' num2str(ceil(timemax/n_output))] );           % Ceil - round off to infinity
% Keep number of time steps to under 300

% Block 4
fprintf(infile,'%s \n',...
    '**block 4 general problem constants',...
    '  dtmin      dtmax    theta    dtempm    sigma    radius     expn   fract1 fract2',...
    ['  0.000001   0.5      1.0        10.  5.66961e-8  0.0       0.0    ' num2str(fract)]);

% Block 5
fprintf(infile,'%s\n','**block 5**********material property data ');
for matc = 1:length(mat)
    fprintf(infile,'%s \n','**rho    dhf   treff   qstar   tabl     matnam (start past col 41)');

    % Find material
    findline(mat{matc},mp);                             % See function below
    l1 = str2num(fgetl(mp));                            % First line of property for given material
    rho(matc) = l1(1);                                  % Density
    Tm(matc) = l1(2);                                   % Melting temperature
    numcheck = str2num(fgetl(mp));                      % Get number of columns
    data = fscanf(mp,'%f',[length(numcheck) inf]);
    data = data';
    data = [numcheck;data];                             % Total data set
    % If ablation then write ablation data
    % Also, if no emmisivity data is given for the top layer, assume its .8 to run SODDIT (get real
data later)
    if matc == 1
        if abl == 3;
            qabl = [l1(3) l1(2)];
        else
            qabl = [];
        end
        if size(data,2) < 4
            data(:,4)=0.8*ones(size(data,1),1)
        end
        % Set emmisivity to zero of lower layers
    else
        data(:,4) = zeros(size(data,1),1);
        qabl = [];
    end
    % Write material property to input file
    fprintf(infile,'%s \n',...
        ['  ' num2str(l1(1)) ' 0.0    298. ' sprintf('%.4E ',qabl) '                          '
mat{matc}],...
        '**temp             cp          cond    emit  absrp');
    fprintf(infile,'  %8.2f    %8.2f     %8.3f%8.2f 1\n',data(1:end-1,:)');
    fprintf(infile,'-1%8.2f    %8.2f     %8.3f%8.2f 1\n',data(end,:)');
    frewind(mp);                                        % Go back to beginning to find next material
end

fprintf(infile,'%s \n','/end of block 5 material property tables');

% Block 6
fprintf(infile,'%s \n',...
    '**block 6 user defined node generation data',...
    '**initial temperature',['**  ' num2str(tinit)],...
    '**(node no) (mat) (dxi) (Temperature) (Area) (Vol) )(NGAP)');
mat_dist = ceil([1:node]/n);
mat_dist(end) = mat_dist(end)-1;    % Find which material corresponds to each node
thick_dist = thick(mat_dist)/n;     % Thickness of each element
A_dist = ones(n*nmat+1,1)';         % Relative area of each element (area into the page)
V_dist = thick_dist.*A_dist;        % Relative volume of each element
```

```
nodedat = [ [1:n*nmat+1]' mat_dist' thick_dist' Tdist' A_dist' V_dist'  zeros(n*nmat+1,1)];     % Make
matrix
fprintf(infile,'%i %4i %10.8f %10.2f %10.8f %10.8f %5i \n',nodedat');                           %
Format & write
fprintf(infile,'/end of Block 6 Node generation\n');

% Block 7

%  Get heating data
fprintf(infile,'%s \n',...
    '**block 7 front face boundary condition data',...
    '**ibcty1 ifmt1 trad1',...
    ['  '  num2str([1 0 20])],...
    '**time   rec enthapy rad input   C sub h           P',...
    '** sec    J/Kg         W/m^2     Kg/m^2-s         atm');

%  Write qdot data to input file
fprintf(infile,'  %i            %i       %.4E        %i      \n',heat');
fprintf(infile,'%s \n',...
    '/end of block 7 aero heating boundary conditions',...
    '**block 8 back face boundary condition data',...
    '**ibctyn ifmtn tradn (perfectly insulated back face boundary condition)',num2str(0));

% Block 9 - pseudo atmosphere data (not used but necessary to fool SODDIT, hahaha)
fprintf(infile,'%s\n',...
    ' 1.0E-03          1.0E-9   1000.               -1720.44         MARS NO ABLATE ',...
    ' 1.0E-03          1.0E-9    300.               -1929.71         MARS NO ABLATE 1 ',...
    ' 1.0E+01          1.0E-9   1000.               -1720.44         MARS NO ABLATE ',...
    ' 1.0E+01          1.0E-9    300.               -1922.91         MARS NO ABLATE 3 ');

disp(['Created ' inputfname]);

ext = findstr('.inp',inputfname);               % Find index of .inp extension

% Put input filename into a file for use in soddit

fprintf(fopen('inf','w'),[current_directory inputfname '\n']);

% Delete old output files, and run soddit
fprintf(fopen('s','w'),['rm ' inputfname(1:ext-1) '_out.txt ' inputfname(1:ext-1) '_plt.txt\n'
soddit_path 'soddit.exe < inf\n']);
fclose('all');

% Tell user to run soddit and wait until finished
disp('Type "s" in a terminal to run SODDIT.  Hit any button once completed.')
pause

% Delete files used to run soddit, but contain no data
if comp==1
    !chmod u+x s
    !s
    !rm inf
end

datfile = fopen([inputfname(1:ext-1) '_plt.txt']);  % Open plot file based on input filename
disp(['Opened ' filename '_plt.txt']);

% Find temperature data
%  Find heading of temperature histories
findline('**t/c temperatures**',datfile);           % See function below
fgetl(datfile);                                     % Skip 1 line to get to start of data
data = fscanf(datfile,'%f',[length(mat)+2 inf]);    % Read in matrix
data=data';

% Place data into appropriate vectors and find times of maximum temps
t = data(:,1);                                      % Time
Twall = data(:,2:length(mat)+2);                    % Temperature
T = Twall;                                          % Don't want to screw things up
Tmaxwall = [max(Twall) data(end,length(mat)+2)];    % Max and final temperatures

% Find times at maximum surface and inner wall temps
```

```
for Tc = 1:length(mat)+1
    iTmax(Tc) = min(find(Twall(:,Tc) == Tmaxwall(Tc)));
end

tmaxwall = [0 t(iTmax)' t(end)];                    % Time vector, contains zero for use in following
for loop

figure(1),clf

for figc = 1:1                                      % Length(Tmaxwall)
    tindex = [2:length(Tmaxwall)+1];                % Indices of timevalues in tmaxwall that you want
to plot.

    % tmaxwall is [inital time, time of max temp at top of each layer, t maxtemp of inner wall, final
time]
    pc = tindex(figc);
    if comp == 2
        a = sprintf('time = %13.5E',tmaxwall(pc));
        ai = findstr('E+0',a);
        a([ai+2]) = [];
    else
        a = sprintf('time = %12.5E',tmaxwall(pc));
    end

    disp(a);
    findline(a,datfile);                            % Find heading of each desired time value
    fgetl(datfile); fgetl(datfile);                 % Skip 2 lines
    data = fscanf(datfile,'%f',[3 inf]);            % Read in temperature profile
    data=data';
    xp(:,1)=data(:,2);Tp(:,1)=data(:,3);            % Assign to proper variables

    % Plot'em up
    figure(1),subplot(1,1,1),plot(xp,Tp);grid on;hold on;
    xlines=cumsum(thick);
    ylines=get(gca,'Ylim');
    plot([xlines;xlines],ylines','k')
    set(gca,'Xlim',[0 xlines(end)]);

    if pc == 1
        dmat = '(Initial Distribution)';
        dtemp = num2str(Tmaxwall(1));
    elseif pc == nmat+2
        dmat = ['(Final Distribution)'];
        dtemp = num2str(Tmaxwall(pc-1));
    else
        dmat = [' (T_m_a_x for ' mat{pc-1} ')'];
        dtemp=num2str(Tmaxwall(pc-1));
    end
    title(['t = ' sprintf('%i',tmaxwall(pc)) 'sec,' dmat])
    xlabel('Depth, m');ylabel('Temperature, K')
end

disp('Read temperature data')

% Display max temperature
figure(2);
clf;

for f2c = 1:nmat+1
    mindex = [1:nmat+1];                            % Columns of temperature history output to plot

    %Columns are:  top of each layer, innerwall
    matc = mindex(f2c);
    figure(2),subplot(1,1,1),plot(t,T(:,matc));
    if f2c == nmat+1
        dmat = 'Inner wall';
        dtemp = '~310';
    else
        dmat = [mat{matc}];
        dtemp = num2str(Tm(matc));
    end
```

```
        title([dmat ', T_m_a_x = ' sprintf('%5.1f',Tmaxwall(matc)) 'K,T_a_l_l_o_w =  ' dtemp 'K'])
        xlabel('Time, sec')
        ylabel('Temperature, K')
        grid on
        hold on
    end

    % Plot heating input
    heat = load([filename '.qw']);
    figure(6); plot(heat(:,1),heat(:,1+hindex));
    title(['Heating Data, ' filename '.qw']);
    xlabel('Time, sec');ylabel('q, W/m^2')
    grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %used to find a given string in a file
    function findline(str,fid)
    %str is desired string, fid is file identifier
    b1=1;
    while b1==1
        b1=isempty(findstr(fgetl(fid),str));
    end
    return
```

23.1.2 Jaron – Appendix – Preliminary Mars Lander Vehicle to Crew Transport Vehicle Rendezvous Study

Author: Jackie Jaron

Contributors: Jackie Jaron

23.1.2.1 Purpose

We recognize the need for the Mars Lander Vehicle (MLV) to rendezvous safely with the Crew Transport Vehicle (CTV) after launch from the Martian surface.

23.1.2.2 Methods

The analysis assumes the MLV launches to Low Martian Orbit (LMO) of 120 km. At this altitude the Martian density is negligible and the first rendzvous burn is performed. We assume the MLV land 180 degrees out of phase of the CTV. During launch, we account for Mars rotation and the down range distance, this difference perturbs the argument of periapsis (*w*) of the MLV orbit with respect to the CTV orbit. To minimize the delta-v rendezvous occurs at the apoapsis of the CTV orbit.

The period of the CTV orbit and MLV orbit are equal to one Martian Day, this allows for multiple opportunities to rendezvous. This analysis method does not look into timing of the two vehicles; Matt Verbeke continues the analysis to address the vehicle phasing issue.

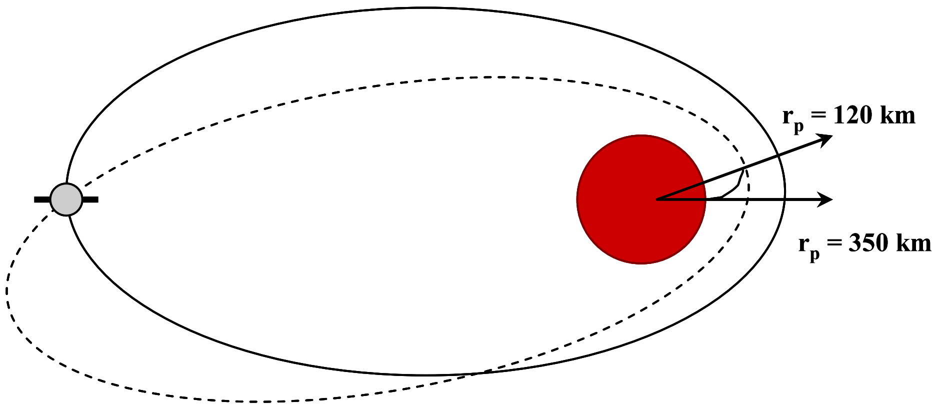


Figure 23-5 MLV Rendezvous to CTV Diagram

23.1.2.3 Inputs & Ouputs

The MATLAB code, MLVrndz.m uses the following inputs:

- *MLV Height (h)* the first rendezvous burn occurs at.
- *CTV Periapsis Height (rp_ctv)*

- *Period (p_ctv)* of CTV orbit
- *Down Range (range)* from the launch site when MLV reaches *MLV Height*. This values is obtained from John Stalbaum's MLV Ascent Code.

The MLVrndz.m outputs the following figure, which describes the MLV rendezvous orbit when the first burn occurs at an altitude of 120 km. We calculate the delta-v values as seen in Table 4 below.

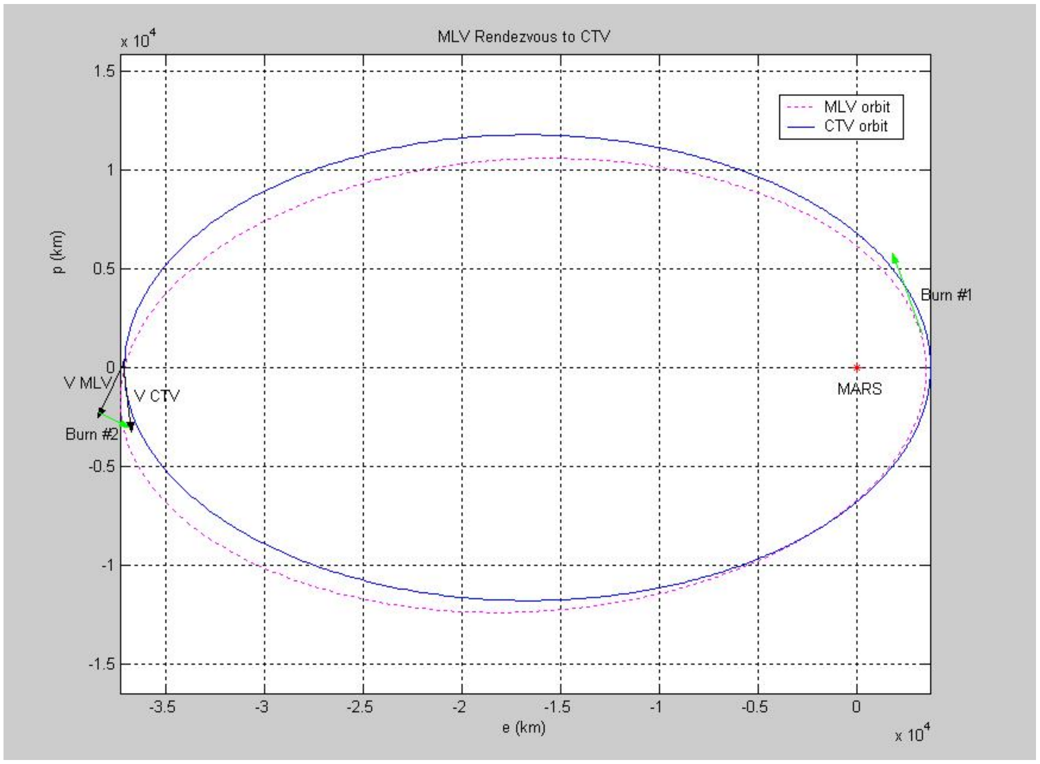


Figure 23-6 MLV Rendezvous & CTV Orbits

Table 4 MLV Rendezvous Delta-V

Burn	Delta-V (km/s)	Description
1	1.07	Burn to insert into MLV Transfer Elipse
2	0.11	Burn to rendezvous with CTV
Total	1.18	

23.1.2.4 MLVrndz.m MATLAB Code

```
% MLVrndz.m
% Written By J.Jaron
% Last modified 02/22/2005
% This matlab code calculates the delta-v necessary for the MLV to
% rendezvous with the CTV when the CTV is in a 1 Martian Day Period orbit
% All inputs are in meters then converted into km
close all;clc;clear all;

% Orbit Parameters
rmars = 3397;
muMars = 42828.2868534;     % [km^3/s^2]
Period = 88646.4;           % Period of parking orbit 1 MARS day


% List of conditions from ascent code
h = 120;                    % altitude above Martian Sea Level [km]%
rp = h + rmars;
v = sqrt(muMars/rp)         % using velocity of circular orbit at alt = 120km
range = 7.5082e2;           % range from trajectory (update) [km]
CircMars = 2*pi*rmars;      % Circumference of Mars [km]
a = ((Period/2/pi)^2*muMars)^(1/3);  % [km]

% CTV orbit paramters
rp_ctv = 350+rmars;
e_ctv = 1 - rp_ctv/a
ra_ctv = a*(1+e_ctv);
va_ctv = sqrt(muMars*(2/ra_ctv-1/a));
p_ctv = rp_ctv*(1+e_ctv);
gammaa_ctv = 0;

% MLV transfer ellipse
wt = 0.1064;                % angular displacement due to Mars rotation [rad]
deltaw =  wt/2;  %deltaw
thetastar = pi - deltaw;  % not accurate enough since MLVascent not
e = 1-rp/a                % assummed a was same as CTV parking orbit
p = a*(1-e^2);
ra = a*(1+e)
pagain = a*(1-e^2)
% delta v stuff
gamma_tx_point = acos(sqrt(muMars*p)/ra_ctv/va_ctv); % verify this with someone
v_tx_insert = sqrt(muMars*(2/rp_ctv-1/a)) - v
v_tx_exit = sqrt(muMars*(2/ra-1/a));
deltav = sqrt(v_tx_exit^2+va_ctv^2-2*v_tx_exit*va_ctv*cos(gamma_tx_point-gammaa_ctv))
totalDeltaV = deltav+v_tx_insert

% rtx = p/(1+e*cos(thetastar));
% working
% thetastar = acos(1/e*(p/ra_ctv-1));

% for plots
theta = linspace(0,2*pi,1000);
rtx = p./(1+e.*cos(theta-deltaw));
x = rtx.*cos(theta);
y = rtx.*sin(theta);
plot(x,y,'m:')
hold on
grid on
r2= p_ctv./(1+e_ctv.*cos(theta));
x2= r2.*cos(theta);
y2= r2.*sin(theta);
plot(x2,y2,'b')
title('MLV Rendezvous to CTV')
xlabel('e (km)')
ylabel('p (km)')
plot(0,0,'r*')
legend('MLV orbit', 'CTV orbit')
axis square
axis equal
```

23.1.3 Jaron – Appendix – Crew Transport Vehicle Free Return Martian Launch Windows

Author: Jackie Jaron

Contributors: Meredith Evans & Jackie Jaron

23.1.3.1 Purpose

To ensure the crew safely completes their mission to Mars and back, the Crew Transport Vehicle (CTV) embarks on a free return trajectory to Mars. In the event of an engine system failure during transit to Mars, the CTV can return to Earth without performing a significant maneuver.

23.1.3.2 Methods

To calculate the Mars free-return launch windows, a program called STOUR optimizes the free-return trajectory. Since we know the synodic cycle between Mars and Earth repeats it self approximately every two years, a free return launch window occurs at least once each synodic period. For the reference date, we use the year of the synodic period and run twelve cases (1 per month). The free return duration and date of Earth departure are the free variables, STOUR uses the reference date to converge on a launch window that provides the lowest delta-v. It is important to note there multiple local minima for a given year and thus it is necessary to run multiple cases for a given year.

23.1.3.3 STOUR Inputs

The following input file (FR2014_1.inp) is an example of a test case we use to generate possible free-return launch window dates (for an explanation of variables, see comments in code):

23.1.3.3.1 FR2014_1.inp STOUR Input File

```
HEAD='Mars Free Return'   % File Heading
SHOTA='EARTH'             % Departure Planet
BULSI='EARTH'             % Arrival Planet
BODY='Mars'               % Flyby Planet
JDL=2014,01,01            % Reference Date in Julian Time
nda=1
ndb=2
rn(2)=-1
ALTB=350                  % Flyby Altitude
re=200                    % Departure Body Altitude
jdate=0.0                 % Number of Julian Days from Reference Date
tpb=150                   % Number of Days to Reach First Flyby Planet
adate=1100                % Number of Days of Entire Trip
varyi='adate','tpb','jdate'       % Which parameters to vary
loops=50, $               % Maximum Number of Iterations
```

23.1.3.4 STOUR Outputs

Once STOUR runs the input file, it converges on a date which provides a free-return trajectory from Earth to Mars. The important outputs to note are the following:

- Earth Departure Date

- Mars Flyby Date
- Earth Arrival Date
- Delta-V necessary for departure at Earth (in km/s)
- $V_{infinity,}$ which is the hyperbolic excess velocity during planet approach (km/s)
- $C^3{}_{,}$ which energy at departure in km^2/s^2

The following table was compiled for each launch window and displays important parameters for each launch window:

Table 5 Mars Free-Return Launch Windows and Associated Delta-V Costs

Phase	Year	Month	Day	Hour	Second	Delta_V (km/s)	V_infinity (km/s)	C^3 (km^2/s^2)
Launch Window 1: 2013-2014								
Departure 1	2013	12	29	15	3	3.7		10.91
Mars fly by	2014	7	1	12	28		6.9	
Arrival 1	2016	12	27	19	12		3.31	
Departure 2	2014	4	3	9	2	3.82		13.56
Mars fly by	2015	5	10	6	27		7.68	
Arrival 2	2017	5	30	19	19		5.75	
Launch Window 2: 2016								
Departure 1	2016	2	21	9	34	3.74		11.69
Mars fly by	2016	7	21	22	22	0	7.71	
Arrival 1	2019	2	20	16	21	0.51	3.42	
Departure 2	2016	2	27	19	24	3.43		4.75
Mars fly by	2017	2	11	11	42		2.81	
Arrival 2	2019	6	6	0	44		3	
Departure 3	2016	7	26	1	9	5.71		61.0925
Mars fly by	2017	6	22	18	55		7.27	
Arrival 3	2016	8	14	22	45		8.18	
Departure 4	2016	5	14	6	36	3.83		13.8039
Mars fly by	2017	5	10	16	38		6.08	
Arrival 4	2019	7	1	6	54		5.8	
Launch Window 3: 2018								
Departure 1	2018	5	5	9	44	3.77		12.4
Mars fly by	2018	9	15	16	8	0	6.09	
Arrival 1	2021	5	3	8	38	0.54	3.52	
Departure 2	2018	6	2	23	36	3.92	15.99	
Mars fly by	2019	4	20	0	50		4.24	
Arrival 2	2021	7	14	18	1		4.97	
Departure 3	2018	5	20	21	17	3.57		7.8627

Mars fly by	2019	1	1	7	47	0	3.18	
Arrival 3	2021	6	16	1	41	0.46	3.25	
Launch Window 4: 2020								
Departure 1	2020	8	24	6	42	3.94		16.5
Mars fly by	2021	10	9	16	0		3.8	
Arrival 1	2023	11	20	3	46		3.23	
Departure 2	2020	7	18	21	35	3.8		13.1797
Mars fly by	2021	1	27	17	52	0	2.86	
Arrival 1	2023	7	3	16	22	0.63	3.81	
Launch Window 5: 2022								
Departure	2022	9	8	16	14	4.03		18.4904
Mars fly by	2023	4	5	23	21	0	3.46	
Arrival	2025	8	18	19	27	1.32	5.59	
Launch Window 6: 2024								
Departure 1	2024	10	16	12	52	4.14		21.1816
Mars fly by	2025	6	30	7	52	0	2.97	
Arrival 1	2027	11	3	3	18	0.99	4.81	
Departure 2	2024	10	13	5	49	3.99		17.72
Mars fly by	2025	5	20	23	0		4.11	
Arrival 2	2027	8	1	0	26		7.58	
Departure 3	2024	10	5	5	46	3.71		11.1896
Mars fly by	2025	9	15	3	13	0	2.54	
Arrival 3	2021	9	26	16	30	0.53	3.47	
Launch Window 7: 2026								
Departure 1	2026	11	28	5	45	5.21		47.9
Mars fly by	2027	3	30	2	54		12.99	
Arrival 1	2029	11	26	10	6		6.93	
Departure 2	2026	11	10	3	2	3.78		12.6478
Mars fly by	2027	6	27	3	49	0	4.36	
Arrival 2	2029	11	4	20	44	0.58	3.63	
Departure 3	2026	10	13	10	21	4		17.7741
Mars fly by	2027	7	1	12	33		2.22	
Arrival 3	2029	11	6	7	8		3.58	
Launch Window 8: 2028								
Departure	2028	12	12	5	20	3.7		10.84.53
Mars fly by	2029	6	27	14	39		6.27	
Arrival	2031	12	10	6	32		3.3	
Launch Window 9: 2030								
Departure 1	2030	12	24	21	48	3.72		11.2485
Mars fly by	2031	9	26	3	22		3.5	
Maneuver	2033	9	2	6	46	1.26		
Arrival 1	2034	4	11	7	5		2.72	
Departure 2	2031	1	29	15	38	3.72		11.3116
Mars fly by	2031	7	12	7	0		7.57	

Arrival 2	2034	1	28	16	28		3.36

23.1.3.5 FR2014_1.ssps STOUR Output File

```
Mars Free Return                                                                                      Feb 19, 2005    23:24:19
 Epoch= 2014  1  1  0  0  0       2456658.500    Earth Ecliptic and Equinox of 2000 and Body Equator and Equinox of Date
       Earth               Earth               Mars
 Set no= 1

    nt= 1.01    iter= 38   kgo= 8  flags  3  0  0  nm= 0  nmt= 3  ndl=  2   nda=  1   ndb=  2
   veq=   3.7122     grad=  .023263      dvt=  3.7122    dvmt=   .0000    dvpl=   .0000    tend= 1110.56     fty=  3.0405
   hca=   .40 -1.64

 jdate=    -5.0    2013 12 27  0 13      dvl=  3.7122      c3= 11.1699     dla=  -8.615     rla= 187.252      re=  6618.1
   tpb=   180.0    2014  6 30  0 25      dvb=   .0000     vhi=   6.992     vho=   6.992    bend=    .590     rcb=  49.823
 adate=  1105.6    2017  1 10 13 36      dva=   .5564     vhp=  3.5708     dap=  -2.001     rap= 175.637      rp=  6378.1

    nt= 1.02    iter= 27   kgo= 2  flags  0  0  0  nm= 0  nmt= 0  ndl=  2   nda=  1   ndb= -0
   veq=   3.7008     grad=  .000000      dvt=  3.7008    dvmt=   .0000    dvpl=   .0000    tend= 1094.17     fty=  2.9957
   hca=   .39 -1.60

 jdate=    -2.4    2013 12 29 15  3      dvl=  3.7008      c3= 10.9087     dla=  -7.867     rla= 184.872      re=  6618.1
   tpb=   181.5    2014  7  1 12 28      dvb=   .0000     vhi=   6.898    betb=  98.472    bend=    .079     rcb=  386.36
 adate=  1091.8    2016 12 27 19 12      dva=   .4786     vhp=  3.3060     dap=  -8.603     rap= 186.096      rp=  6378.1

Input File: FR2014_1.inp        Cumulative run time= 0.04 sec                                          Step= .100E-01
```

23.1.3.6 References

Landau, Damon. “To Mars and Back.” Presentation to Purdue University AAE 450. 15 Feb 2005.

Longuski, James. “Optimization in Aerospace Engineering (AAE 508) Course Packet.” Prepared for Purdue University. 2005.

23.1.4 Jaron – Appendix – Crew Transport Vehicle Free Return Martian Launch Windows

Author: Jackie Jaron

Contributors: Meredith Evans & Jackie Jaron

23.1.4.1 Purpose

To ensure the crew safely completes their mission to Mars and back, the Crew Transport Vehicle (CTV) embarks on a free return trajectory to Mars. In the event of an engine system failure during transit to Mars, the CTV can return to Earth without performing a significant maneuver.

23.1.4.2 Methods

To calculate the Mars free-return launch windows, a program called STOUR optimizes the free-return trajectory. Since we know the synodic cycle between Mars and Earth repeats it self approximately every two years, a free return launch window occurs at least once each synodic period. For the reference date, we use the year of the synodic period and run twelve cases (1 per month). The free return duration and date of Earth departure are the free variables, STOUR uses the reference date to converge on a launch window that provides the lowest delta-v. It is important to note there multiple local minima for a given year and thus it is necessary to run multiple cases for a given year.

23.1.4.3 STOUR Inputs

The following input file (FR2014_1.inp) is an example of a test case we use to generate possible free-return launch window dates (for an explanation of variables, see comments in code):

23.1.4.3.1 FR2014_1.inp STOUR Input File

```
HEAD='Mars Free Return'   % File Heading
SHOTA='EARTH'             % Departure Planet
BULSI='EARTH'             % Arrival Planet
BODY='Mars'               % Flyby Planet
JDL=2014,01,01            % Reference Date in Julian Time
nda=1
ndb=2
rn(2)=-1
ALTB=350                  % Flyby Altitude
re=200                    % Departure Body Altitude
jdate=0.0                 % Number of Julian Days from Reference Date
tpb=150                   % Number of Days to Reach First Flyby Planet
adate=1100                % Number of Days of Entire Trip
varyi='adate','tpb','jdate'       % Which parameters to vary
loops=50, $               % Maximum Number of Iterations
```

23.1.4.4 STOUR Outputs

Once STOUR runs the input file, it converges on a date which provides a free-return trajectory from Earth to Mars. The important outputs to note are the following:

- Earth Departure Date
- Mars Flyby Date

- Earth Arrival Date
- Delta-V necessary for departure at Earth (in km/s)
- $V_{infinity,}$ which is the hyperbolic excess velocity during planet approach (km/s)
- $C^3{}_{,}$ which energy at departure in km^2/s^2

The following table was compiled for each launch window and displays important parameters for each launch window:

Table 6 Mars Free-Return Launch Windows and Associated Delta-V Costs

Phase	Year	Month	Day	Hour	Second	Delta_V (km/s)	V_infinity (km/s)	C^3 (km^2/s^2)
Launch Window 1: 2013-2014								
Departure 1	2013	12	29	15	3	3.7		10.91
Mars fly by	2014	7	1	12	28		6.9	
Arrival 1	2016	12	27	19	12		3.31	
Departure 2	2014	4	3	9	2	3.82		13.56
Mars fly by	2015	5	10	6	27		7.68	
Arrival 2	2017	5	30	19	19		5.75	
Launch Window 2: 2016								
Departure 1	2016	2	21	9	34	3.74		11.69
Mars fly by	2016	7	21	22	22	0	7.71	
Arrival 1	2019	2	20	16	21	0.51	3.42	
Departure 2	2016	2	27	19	24	3.43		4.75
Mars fly by	2017	2	11	11	42		2.81	
Arrival 2	2019	6	6	0	44		3	
Departure 3	2016	7	26	1	9	5.71		61.0925
Mars fly by	2017	6	22	18	55		7.27	
Arrival 3	2016	8	14	22	45		8.18	
Departure 4	2016	5	14	6	36	3.83		13.8039
Mars fly by	2017	5	10	16	38		6.08	
Arrival 4	2019	7	1	6	54		5.8	
Launch Window 3: 2018								
Departure 1	2018	5	5	9	44	3.77		12.4
Mars fly by	2018	9	15	16	8	0	6.09	
Arrival 1	2021	5	3	8	38	0.54	3.52	
Departure 2	2018	6	2	23	36	3.92	15.99	
Mars fly by	2019	4	20	0	50		4.24	
Arrival 2	2021	7	14	18	1		4.97	
Departure 3	2018	5	20	21	17	3.57		7.8627
Mars fly by	2019	1	1	7	47	0	3.18	
Arrival 3	2021	6	16	1	41	0.46	3.25	

Launch Window 4: 2020								
Departure 1	2020	8	24	6	42	3.94		16.5
Mars fly by	2021	10	9	16	0		3.8	
Arrival 1	2023	11	20	3	46		3.23	
Departure 2	2020	7	18	21	35	3.8		13.1797
Mars fly by	2021	1	27	17	52	0	2.86	
Arrival 1	2023	7	3	16	22	0.63	3.81	
Launch Window 5: 2022								
Departure	2022	9	8	16	14	4.03		18.4904
Mars fly by	2023	4	5	23	21	0	3.46	
Arrival	2025	8	18	19	27	1.32	5.59	
Launch Window 6: 2024								
Departure 1	2024	10	16	12	52	4.14		21.1816
Mars fly by	2025	6	30	7	52	0	2.97	
Arrival 1	2027	11	3	3	18	0.99	4.81	
Departure 2	2024	10	13	5	49	3.99		17.72
Mars fly by	2025	5	20	23	0		4.11	
Arrival 2	2027	8	1	0	26		7.58	
Departure 3	2024	10	5	5	46	3.71		11.1896
Mars fly by	2025	9	15	3	13	0	2.54	
Arrival 3	2021	9	26	16	30	0.53	3.47	
Launch Window 7: 2026								
Departure 1	2026	11	28	5	45	5.21		47.9
Mars fly by	2027	3	30	2	54		12.99	
Arrival 1	2029	11	26	10	6		6.93	
Departure 2	2026	11	10	3	2	3.78		12.6478
Mars fly by	2027	6	27	3	49	0	4.36	
Arrival 2	2029	11	4	20	44	0.58	3.63	
Departure 3	2026	10	13	10	21	4		17.7741
Mars fly by	2027	7	1	12	33		2.22	
Arrival 3	2029	11	6	7	8		3.58	
Launch Window 8: 2028								
Departure	2028	12	12	5	20	3.7		10.84.53
Mars fly by	2029	6	27	14	39		6.27	
Arrival	2031	12	10	6	32		3.3	
Launch Window 9: 2030								
Departure 1	2030	12	24	21	48	3.72		11.2485
Mars fly by	2031	9	26	3	22		3.5	
Maneuver	2033	9	2	6	46	1.26		
Arrival 1	2034	4	11	7	5		2.72	
Departure 2	2031	1	29	15	38	3.72		11.3116
Mars fly by	2031	7	12	7	0		7.57	
Arrival 2	2034	1	28	16	28		3.36	

23.1.4.5 FR2014_1.ssps STOUR Output File

```
Mars Free Return                                                                                    Feb 19, 2005    23:24:19
 Epoch= 2014  1  1  0  0  0        2456658.500    Earth Ecliptic and Equinox of 2000 and Body Equator and Equinox of Date
       Earth               Earth               Mars
 Set no= 1

    nt= 1.01    iter= 38   kgo= 8  flags  3  0  0  nm= 0  nmt= 3  ndl=  2   nda=  1   ndb=  2
   veq=   3.7122     grad=  .023263      dvt=  3.7122    dvmt=   .0000    dvpl=   .0000    tend= 1110.56     fty=  3.0405
   hca=   .40 -1.64

 jdate=    -5.0    2013 12 27  0 13      dvl=  3.7122      c3= 11.1699     dla=  -8.615     rla= 187.252      re=  6618.1
   tpb=   180.0    2014  6 30  0 25      dvb=   .0000     vhi=   6.992     vho=   6.992    bend=    .590     rcb=  49.823
 adate=  1105.6    2017  1 10 13 36      dva=   .5564     vhp=  3.5708     dap=  -2.001     rap= 175.637      rp=  6378.1

    nt= 1.02    iter= 27   kgo= 2  flags  0  0  0  nm= 0  nmt= 0  ndl=  2   nda=  1   ndb= -0
   veq=   3.7008     grad=  .000000      dvt=  3.7008    dvmt=   .0000    dvpl=   .0000    tend= 1094.17     fty=  2.9957
   hca=   .39 -1.60

 jdate=    -2.4    2013 12 29 15  3      dvl=  3.7008      c3= 10.9087     dla=  -7.867     rla= 184.872      re=  6618.1
   tpb=   181.5    2014  7  1 12 28      dvb=   .0000     vhi=   6.898    betb=  98.472    bend=    .079     rcb=  386.36
 adate=  1091.8    2016 12 27 19 12      dva=   .4786     vhp=  3.3060     dap=  -8.603     rap= 186.096      rp=  6378.1

Input File: FR2014_1.inp       Cumulative run time= 0.04 sec                                             Step= .100E-01
```

23.1.4.6 References

Landau, Damon. “To Mars and Back.” Presentation to Purdue University AAE 450. 15 Feb 2005.

Longuski, James. “Optimization in Aerospace Engineering (AAE 508) Course Packet.” Prepared for Purdue University. 2005

24 Kallimani, James G.

24.1.1.1 Radiation Background

Author: James Kallimani

Earth has supported life for billions of years. Life has flourished on Earth because of the conditions that this planet offers. Gravity, air, and water are all taken for granted on Earth, and those conditions do not exist in space. The rigor of no gravity and no air leads us to create artificial atmosphere and gravity, similar to the conditions found on Earth. The Earth, however, also provides what may be one of the most important features of our survival. The Earth's electromagnetic (EM) field shields us from the high radiation levels in space.

Without this EM field, the surface of the Earth would be unable to support life. It would be bombarded by radiation that occurs naturally in space. There are two main types of space radiation that we are protected from here on Earth; Galactic Cosmic Radiation (GCR), and Solar Particle Events (SPE).

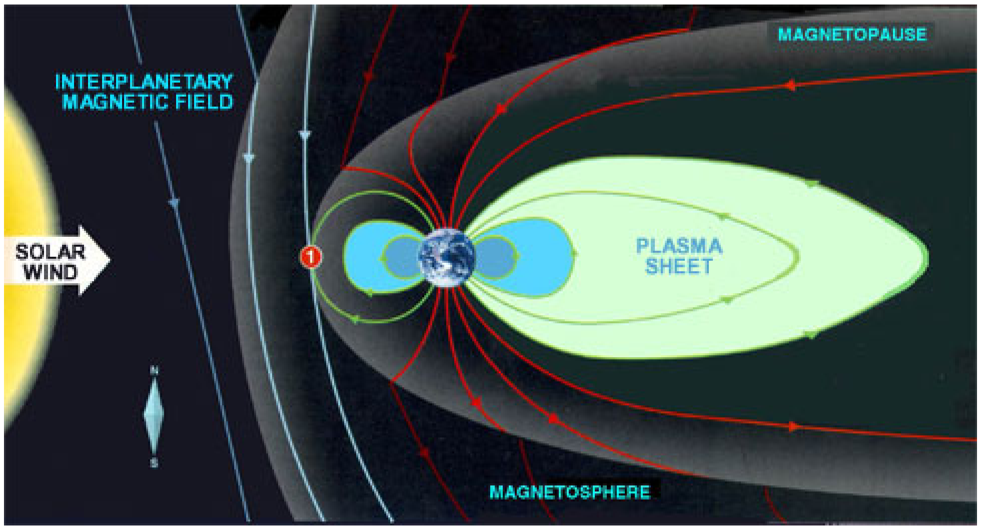


Figure 1. Earth's Protective EM Field (Adapted from Popular Science, 2001)

24.1.1.1.1 Solar Particle Events and Solar Flares

SPE has been observed on Earth for millions of years. The Aurora Borealis, also known as the Northern Lights, is nothing more than solar particles flying through the Earth's EM field. These particles come from the Sun. Without the EM field, this radiation would make it to the surface with deadly results. A more intense form of SPE is called a solar flare. The Sun is on an 11 year cycle. We estimate the largest solar events happening at the same point during that cycle. These solar flares are spikes in the radiation release from the sun, and they have significant effects. On the surface, the Northern Lights are more brilliant during a period of solar maximum. In space, these spikes wreak havoc on the equipment that we have come to rely on. This equipment survives only because it is still within the EM field. However, solar flares can destroy the electronics of satellites and other orbiting equipment. A noticeable effect on every day life is the degrading effect of SPE on satellite TV. This occurs when solar radiation is disrupting the transmission from the satellites to the receiving dishes on Earth.

Figure 2. Solar Radiation seen as the Northern Lights (http://www.kazuhiro-yamada.com/astro/aurora/00-025-22.html)

24.1.1.1.2 Galactic Cosmic Radiation

The second form of space radiation, GCR, gained attention during the Apollo missions of the late 60s and early 70s. It was during these missions to the moon that we first experienced GCR. It was first noticed by the astronauts when they closed their eyes. They would see bright flashes of light. Camera equipment also recorded these bright flashes. An investigation conducted during the latter missions showed that the bright flashes experienced by the astronauts were actually microscopic particles, nuclei of atoms, bombarding their bodies. These nuclei (ranging from helium atoms to uranium atoms), are created by distant stars and exist everywhere in space. When these high energy particles hit the retina of the eye, it was translated as a flash. This GCR effects both crew and equipment. The minimal shielding of the Apollo space craft was enough to shield the crew and equipment for short missions, but it would be very inadequate for longer missions. Studies show that the energy levels for GCR can reach well above 100 GeV. The most common energy level is 1 GeV, or 1000 MeV. This is compared to the 2 MeV level in the reactors.

Particles	Rate
Hydrogen nuclei (protons)	85%
Helium nuclei (alpha particles)	12.5%
Heavier atom nuclei	1%
electrons	1.5%

Table 1. Galactic Cosmic Radiation Breakdown

24.1.1.1.3 Neutron Radiation

The third form of radiation that is experienced by the crew is generated by the power and propulsion reactors. The power devices used in the CTV and the MHV are nuclear fission reactors. Nuclear fission reactors use uranium as an energy carrier. A free neutron traveling at speeds near that of light strikes the uranium atom and splits it in two pieces. This generates more neutrons. All these neutrons create a cloud around the ship. This cloud travels through space with the CTV, irradiating every unshielded surface. This is also a problem with the engines that are used by the CTV and the MHV. The nuclear thermal engines expel radioactive material to produce thrust. Without shielding, this material would harm the ship and the crew. Like the reactors, this expelling of materials creates a neutron cloud.

24.1.1.2 Radiation Shielding and Power Systems Appendix

Author: James Kallimani

24.1.1.2.1 Radiation Shielding

Designing a radiation shield is a very hard task, primarily because there is no really good data on which to base our design. For example, we know that Earth and Mars pass each other every 2.17 years. Using this data we calculate accurate orbit trajectories. Nothing like this can be said about the radiation the astronauts will experience on the way to Mars. The energy levels of radiation that we must shield against have never been properly recorded and documented. This area of design has not been well published. Therefore, we are unable to base our calculations on anything that we find in a book. In "The Case for Mars," Dr. Zubrin uses out of date short term low energy radiation data in his calculations. A mission to Mars will encounter quite the opposite; long term high energy radiation. We do not have long term data on human reactions to GCR and SPE, nor do we have long term data on how a nuclear reactor will function in the high radiation environment of space. To estimate the reactions that crew and equipment will have to the radiation, we use programs developed by national nuclear laboratories like Oak Ridge and Lawrence Livermore. Monty Carlo methods (MCNP) and COG are two of the programs designed to calculate material interactions. We must not use programs like Matlab to determine shielding sizes, because Matlab is not capable of calculating the complex interactions that occur between the radiation and the materials. Problems occur when using MCNP, because all users are required to obtain a license from Oak Ridge National Lab. Therefore, inputs written for MCNP can not be shared. COG, however, is a different story. We are able to include a sample shielding input that was modified from one available from Lawrence Livermore Nuclear Lab's website [1]. These two programs are essentially random number generators. When dealing with radiation, we look at millions and millions of particles entering a given material, like polyethylene. Each particle can react in different ways. The particle can scatter, or reflect off. It can be absorbed by the material. It can pass straight through the material. Or, it can hit other atoms, and cause them to become radiation. COG can follow each of the millions of particles through the material and calculate the odds of each thing happening. It also calculates the radiation that is created by the particles. These are the type of calculations that Matlab is incapable of performing.

```
basic
  neutron

mix
$USE ENDL90 Neutron Library
  NLIB= ENDL90
  mat 1 u235 17.7368  u238 1.012
  mat 2 polyethy .94

geometry
  sector 1 uranium -1
  sector 2 polyethylene   -2

  picture cs m color -36. 0. 12.  -36. 0. -36.  36. 0. -36.

surfaces
  1 cyl 0.4375 -0.4375 0.4375  tr -20.32 0 0 -20.32 0 -120      $ src
  2 box 8 1.5 20         tr 0 0 0        $ polyethylene block
  $Cylinder has radius 3.5 cm, from point 0 to 20 cm (20 cm long)

detector
  number 1
  point 20. 0. 0.
  drf-e neutron      $ gm tube sens for al cathode
    0.1 0.      0.2 8.0e-4  0.3 1.0e-3  0.4 1.6e-3
    0.5 2.1e-3  0.6 2.6e-3  0.7 3.3e-3  0.8 4.0e-3
    0.9 4.5e-3  1.0 5.1e-3  1.1 5.9e-3  1.2 6.6e-3
    1.3 7.2e-3  1.4 8.0e-3
  bin e neutron
    16 [0. i 1.5]

source
  npart 50000
  define p 1
    cylinder -24. 0. -0.4375  -24.  0.  0.4375  0.4375
  define t 1
    steady
  define e 1
    neutron line 14000 1.
  inc 7.4e7  e 1  p 1  t 1

walk-collision
  neutron region all energy 0. to 0.1 0.

end
```

Table 1. COG Shielding Input

This input describes a 14 MeV neutron source traveling through a block of polyethylene, with a detector placed after the material. By changing the thickness of the polyethylene, we can get an idea of what is required to shield from radiation in space.

24.1.1.2.2 Electromagnetic Radiation Shielding

Institutions like NASA, Brookhaven National Lab, and Purdue University are all working to develop the necessary hardware to ensure the safety of a mission beyond our planet. Because of this, we had no definite method to use. Polyethylene is a material of interest because of its high hydrogen content, similar to water, and light weight. Unlike water, other materials can be mixed in with polyethylene.

Currently at Purdue University, testing is underway using different types of polyethylene, including polyethylene with 5% Boron and 7.5% lithium mixed into the plastic. These materials may be the key to safe efficient travel outside of the Earth's electromagnetic (EM) field. Similarly, other institutions like the NASA Institute for Advanced Concepts (NAIC) [2], are studying the possibility of creating an artificial EM field around an interplanetary space craft. Dr. Jeffery Hoffman at the NIAC is leading this charge to use superconducting magnetic coils.

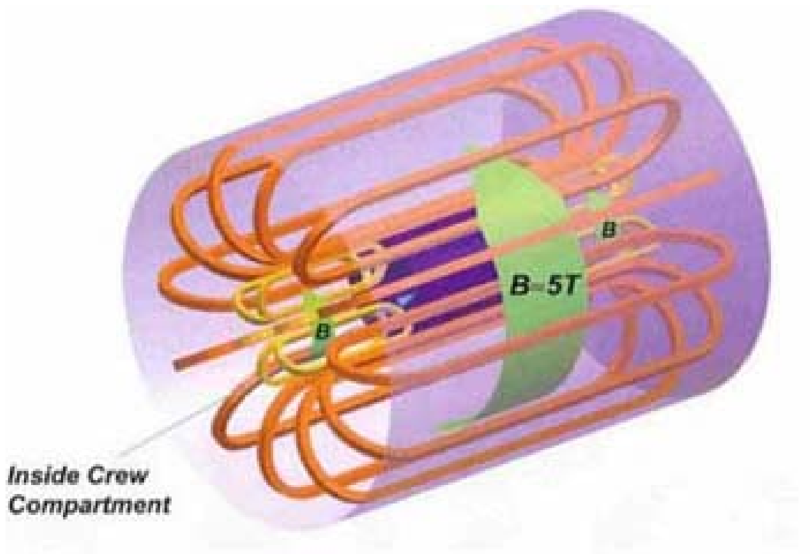


Figure 1. EM Field as described by NAIC (Image from NIAC)

This method was considered for Project Legend after initial calculations showed high shield masses. The EM shielding would use lightweight superconducting coils, but require large amounts of power. The power systems on the CTV are inadequate to supply the needed power, and increasing the system to be able to handle the increased power drain proved to increase the mass of reactor components and thermal equipment to an infeasible level.

24.1.1.2.3 Nuclear Fusion Reactors

To make this shielding feasible, an alternate power system could be used. Nuclear fusion power technology will be common in the near future. This technology will be based on hydrogen as a power source, not uranium. Hydrogen is not radioactive, and the by-products of fusion are primarily helium. The power device is called a dense plasma focus (DPF). This DPF is being developed at facilities like Lawrenceville Plasma Physics Lab [3], and The University of Mexico. This is the safest and most efficient power source for use on a space craft. A DPF with the same dimensions as the fission device that we are using would produce around 2 MW, instead of the 100 kW that the current device does.

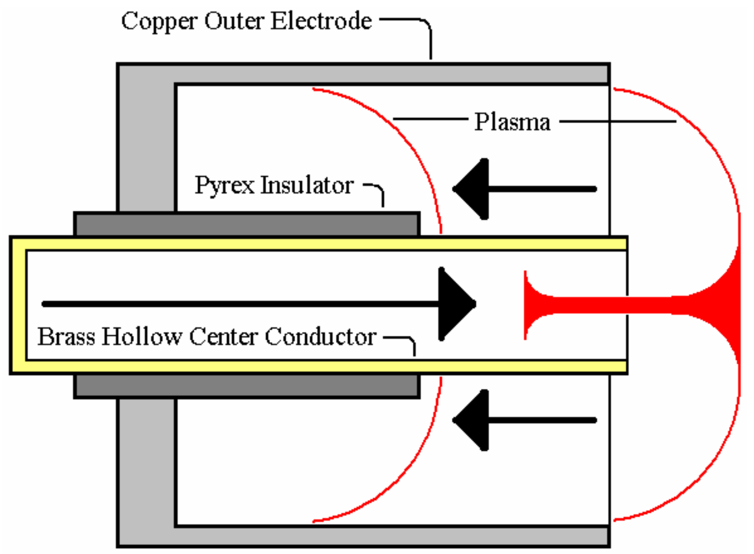


Figure 2. Dense Plasma Focus (Based on design from Lawrenceville Plasma Physics)

We would get 20 times more power for the mass and space. This makes the EM shielding possible. However, the technology has yet to be proven, and we decided against using it in our design. Perhaps this will be more feasible in the near future when another Mars mission is on the drawing board.

24.1.1.2.4 Reasons for Physical Shielding

For these reasons, we decided to use the physical shielding and not the EM shielding. We looked at two main materials for shielding: aluminum and polyethylene. These are two of the materials we are currently working with in conjunction with the Nuclear Engineering Department at Purdue University [4]. Aluminum acts as a reflector, as well as a good structural material. It reflects some of the radiation particles away before they ever enter the shield. Polyethylene has a very close chemical makeup to water. This is used to absorb radiation two different ways. It can absorb particles by making them heavier with extra neutrons, and it can absorb the particles simply by slowing them. The polyethylene will be the thickest part of the shielding to perform both of these functions.

Radiation shielding was met with skepticism from the majority of the engineers involved with this project, mostly because they had never heard of it before, and because there is a lack of information for them to read. Plus, the mass needs for the shielding were driving the cost of the project. However, by the end, it was clear that radiation shielding is needed.

The method that we use to design a radiation shield is a very sound one. We look at the energy passing through each section of the shield, and make it so that the interior of the space craft is safe for the mission duration. The calculated masses of the shielding of both the CTV and the MHV are estimates. As we described earlier, there is a lack of data in this area, so we can not be positive of any design until we can actually test it. However, the shielding described in this report is accurate in the sense we have the correct materials and the correct method. With more time we can develop the shielding further. Radiation shielding will help drive the design of future space craft, and Project Legend shows that shielding from radiation is needed, but is also very feasible.

24.1.1.2.5 Sources

[1] COG Table of Contents. 2005. Physics and Advanced Technologies Directorate. 2005. <http://www-phys.llnl.gov/N_Div/COG/Manual/COG_ToC.html>

[2] NAIC. 2005. NASA Institute for Advanced Concepts. 2005. <http://www.niac.usra.edu/>

[3] Dense Plasma Focus. 2005. Lawrenceville Plasma Physics. 2005. <http://www.lawrencevilleplasmaphysics.com/>

[4] Laboratory for Neurotics and Geometry Computation. 2005. Purdue University. 2005. <https://engineering.purdue.edu/ResearchGroups/NEGE>

25 Kloster, Kevin

25.1 Propellant Calculation

Author(s): Kevin Kloster

Contributor(s): Joe Davis, James Kallimani

```
% Kevin Kloster
% AAE450
% finite_burn.m
% This code iterates to calculate tank masses,
% gravity losses, engine shield mass, and
% propellant mass.
% Input is launch window (1-7)
% Outputs are actual delta v, propellant masses for each maneuver,
% tank masses, total initial CTV mass and burn times for each maneuver.

clc
clear all
close all
format long g

F = 1000000;              % N
Isp = 1022.36922;         % s
g0 = 9.80665;             % m/s^2
x0 = 5;                   % number of engines
m_engines = 2195*x0;      % <-- for F = 1000000 N
% calculates mass of engine shield
r_shld = .3869;           % m  <--
h_shld = .6330;           % m  <-- for F = 1000000 N
th_Rhe = .002;            % m
th_Poly = .498;           % m
th_Al = .1;               % m
rho_Rhe = 21040;          % kg/m^3
rho_Poly = 910;           % kg/m^3
rho_Al = 2700;            % kg/m^3
r_outer = r_shld+th_Rhe+th_Poly+th_Al;
m_Rhe = (2*pi*r_shld*h_shld+pi*r_outer^2)*th_Rhe*rho_Rhe;
m_Poly = (2*pi*(r_shld+th_Rhe)*h_shld+pi*r_outer^2)*th_Poly*rho_Poly;
m_Al = (2*pi*(r_shld+th_Rhe+th_Poly)*h_shld+pi*r_outer^2)*th_Al*rho_Al;
m_engshld = (m_Rhe+m_Poly+m_Al)*x0;       % kg
% ---------------------------------
mdot0 = F/(Isp*g0)/x0;     % kg/s
mdot = mdot0*x0;           % kg/s
G = 6.67259e-11;
Me = 5.9736e24;
mue = G*Me;
Re = 6378140;             % km
rp_LEO = Re+200e3;        % km
Mm = 6.4185e23;
mum = G*Mm;
Rm = 3397000;             % km
rp_LMO = Rm+350e3;        % km
```

```
m_ARV = 7600;           % kg
m_MLV = 38033;          % kg
m_humans = 280;         % kg
m_H2scr = 400;          % kg  <-- this is to power the CO2 srubbers
m_truss = 17540;        % kg
m_super = 6456;         % kg
m_crewQ = 15430;        % kg
m_reactshld = 8400;     % kg
m_struct_sub = 56949.3;  % kg
m_mount = 0.1*m_struct_sub; % kg
m_CTV = 133162.2;          % kg
m_CTV0 = m_CTV-m_mount;
tank_str = 0.15;        % 15%
m_tank6 = 0;
m_tank5 = 0;
m_tank4 = 0;
m_tank3 = 0;
m_tank2 = 0;
m_tank1 = 0;
m_tankIFP = 0;
m_tankOFP = 0;
m_tankPERM = 0;
d = 2;     % mission opportunity
% delta v's for each maneuver
% on each mission window
dv =  [3574 1183 500 500 1349  870;
       3807 1015 500 500 1334  852;
       4034 1345 500 500 1636  787;
       3723  857 500 500  925  845;
       4005  715 500 500  929  966;
       3705 3328 500 500 1034 1110;  % <-- Do NOT use
       4590 1264 500 500 1034 1110]; % <-- Worst Case

% v infinities for each maneuver
% on each mission window
vx =  [2797 3176   0   0 3467 3116;
       3630 2860   0   0 3441 3051;
       4300 3460   0   0 3935 2797;
       3350 2540   0   0 2682 3023;
       4220 2220   0   0 2689 3453;
       3290 6270   0   0 2898 3907;
       5650 3320   0   0 2898 3907];

dv1_act = dv(d,1);
dv2_act = dv(d,2);
dv3_act = dv(d,3);
dv4_act = dv(d,4);
dv5_act = dv(d,5);
dv6_act = dv(d,6);

for j=1:10
    m_struct_sub = m_CTV-m_struct_sub*0.1-m_truss-m_super-m_crewQ-m_reactshld-
m_tankPERM-m_engshld;
    m_CTV = m_CTV0+m_struct_sub*0.1;
    m_empty = m_CTV-m_humans;     % kg
    dve_A = dv6_act;                            % m/s
    m_prop6 = exp(dve_A/(Isp*g0))*m_empty-m_empty;     % kg
    m_tank6 = (m_prop6+m_H2scr)*tank_str; % <-- scrub H2 stored in Permtank
    m_tankPERM = m_tank6;
    t6 = m_prop6/mdot;
    mf5 = m_empty+m_prop6+m_humans+m_ARV+m_tankIFP;    % kg
    dvm_D = dv5_act;        % m/s
    m_prop5 = exp(dvm_D/(Isp*g0))*mf5-mf5;    % kg
    m_tank5 = m_prop5*tank_str;
    t5 = m_prop5/mdot;
    m_tankCB = m_MLV-m_MLV/(1+tank_str);
    mf4 = mf5+m_prop5;     % kg
    dv5_act = grav_loss(F,mf4,mdot,2*t5,rp_LMO,mum,vx(d,5));
    dvm_apo2 = dv(d,4);    % m/s
    m_prop4 = exp(dvm_apo2/(Isp*g0))*mf4-mf4;    % kg
    m_tank4 = m_prop4*tank_str;
    t4 = m_prop4/mdot;
    mf3 = mf4+m_prop4+m_MLV+m_tankCB;   % kg
    dvm_apo1 = dv(d,3);        % m/s
    m_prop3 = exp(dvm_apo1/(Isp*g0))*mf3-mf3;    % kg
    m_tank3 = m_prop3*tank_str;
    t3 = m_prop3/mdot;
    mf2 = mf3+m_prop3;
    dvm_A = dv2_act;    % m/s
    m_prop2 = exp(dvm_A/(Isp*g0))*mf2-mf2;    % kg
    m_tank2 = m_prop2*tank_str;
    m_tankIFP = (m_prop2-m_MLV/(1+tank_str)+m_prop3+m_prop4+m_prop5)*tank_str;
    t2 = m_prop2/mdot;
    mf1 = mf2+m_prop2+m_tankOFP;
    dve_D = dv1_act;     % m/s
    m_prop1 = exp((dve_D)/(Isp*g0))*mf1-mf1;    % kg
    m_tank1 = m_prop1*tank_str;
    m_tankOFP = m_tank1;
    t1 = m_prop1/mdot;
    m_tot = mf1+m_prop1;
    dv1_act = grav_loss(F,m_tot,mdot,t1,rp_LEO,mue,vx(d,1));
    m_prop = m_prop1+m_prop2+m_prop3+m_prop4+m_prop5+m_prop6;
end
total_mass = m_tot
propellant_mass = m_prop

dv_act = [dve_D;
          dvm_A;
          dvm_apo1;
          dvm_apo2;
          dvm_D;
          dve_A]

num = [m_tot         m_tot-m_prop1    m_prop1 t1   m_tankOFP;
       mf2+m_prop2       mf2          m_prop2 t2       0;
       mf3+m_prop3       mf3          m_prop3 t3 m_tankCB+m_MLV;
       mf4+m_prop4       mf4          m_prop4 t4       0;
       mf5+m_prop5       mf5          m_prop5 t5       0;
       m_empty+m_prop6 m_empty        m_prop6 t6       0];

tanks = [m_tankOFP;
         m_tankCB;
         m_tankIFP;
         m_tankPERM]

m_prop = [m_prop1;
          m_prop2;
          m_prop3;
          m_prop4;
          m_prop5;
          m_prop6]

save mass num -ASCII -TABS


% grav_loss.m
% calculates actual delta v.
% inputs are thrust, initial mass, mdot, burn time,
% initial radius, gravitational constant mu, and desired v_infinity.
% output is actual delta v.
function [dv_act, v] = grav_loss(F,m0,mdot,tb,rp,mu,v_inf)

thetadot0 = sqrt(mu/rp)/rp;
x0 = [rp 0 0 thetadot0 F/m0];
tspan = linspace(0,2*tb,10000);
[t1, x1] = ode45(@(t,x) eom(t,x,F,m0,mdot,mu),tspan,x0);

v = sqrt(x1(:,2).^2+(x1(:,1).*x1(:,4)).^2);
vx = sqrt(v.^2-2*mu./x1(:,1));
[t_ind] = find(vx>=v_inf,1);
tb = t1(t_ind);
dv_act = x1(t_ind,5);


% eom.m
% equations of motion in polar coordinates
% numerically integrate to get actual velocity.
function xdot = eom(t,x,F,m0,mdot,mu)

m = m0-mdot*t;
v = sqrt(x(2)^2+(x(1)*x(4))^2);
gamma = acos(x(1)*x(4)/(v));

xdot(1) = x(2);
xdot(2) = x(1)*x(4)^2-mu/x(1)^2+F/m*sin(gamma);
xdot(3) = x(4);
xdot(4) = -2*x(2)*x(4)/x(1)+F/m*cos(gamma)/x(1);
xdot(5) = F/m;

xdot = xdot';
return
```

25.2 Trajectory Analysis

Author(s): Kevin Kloster

Contributor(s): Jackie Jaron, Meredith Evans, Brienne Bogenberger, Phil Spindler

```
% Kevin Kloster
% AAE450
% free_return.m
% This code shows why the swingby free-return option was not feasible
% and plots the one and a half year period free-return orbit.
% The input is the true anomaly in Earth’s orbit to launch from.
% The output is the eccentricity of the free return transfer orbit and % a plot of
the orbit.

clc
clear all
close all
format long g

mue = 398600.4418;
Re = 6378.1;
mum = 42828.2869;
Rm = 3397;
mu = 132712200000;
Rs = 695990;
ae = 149597886;
am = 227936636;
ee = 0;
em = .09341233;
pe = ae*(1-ee^2);
pm = am*(1-em^2);
thetastarE = 0*pi/180;
thetastarM = 330*pi/180;
rD = pe/(1+ee*cos(thetastarE));
rA = pm/(1+em*cos(thetastarM));
TA = 180*pi/180;
c = sqrt(rD^2+rA^2-2*rD*rA*cos(TA));
s = .5*(rD+rA+c);
a = s/2;
alpha = 2*asin(sqrt(s/(2*a)));
beta = 2*asin(sqrt((s-c)/(2*a)));
p = 4*a*(s-rD)*(s-rA)/c^2*(sin((alpha+beta)/2))^2;
e = sqrt(1-p/a);
thetastarD = acos((p/rD-1)/e);
thetastarA = acos((p/rA-1)/e);
TOFmin = sqrt(a^3/mu)*((alpha-sin(alpha))-(beta-sin(beta)));

r_LEO = Re+300;
v_LEO = sqrt(mue/r_LEO);
vplus = sqrt(2*mu/rD-mu/a);
vearth = sqrt(2*mu/rD-mu/ae);
gammaE = 0;
gammaD = -acos(sqrt(mu*p)/(rD*vplus));
```

```
dgamma = gammaD-gammaE;
vxe = vplus-vearth;
dvi = sqrt(vxe^2+2*mue/r_LEO)-v_LEO
r_LMO = Rm+300;
v_LMO = sqrt(mum/r_LMO);
vminus = sqrt(2*mu/rA-mu/a);
vmars = sqrt(mu/rA);
vxm = vmars-vminus;
dvf = sqrt(vxm^2+2*mum/r_LMO)-v_LMO
dvtot = dvi+dvf
TOF = pi*sqrt(a^3/mu)/60/60/24

ah = mum/vxm^2;
rp = Rm+100;
eh = rp/ah+1;
delta = 2*asin(1/eh);
vswing = sqrt(vmars^2+vxm^2-2*vmars*vxm*cos(delta))
dveq = sqrt(vxm^2+vxm^2-2*vxm*vxm*cos(delta))
gamma = acos((vxm^2-vmars^2-vswing^2)/(-2*vmars*vswing));
gammachk = asin(vxm*sin(delta)/vswing);
anew = mu/(2*mu/rA-vswing^2);
pnew = (rA*vswing*cos(gamma))^2/mu;
enew = sqrt((rA*vswing^2/mu-1)^2*cos(gamma)^2+sin(gamma)^2);
thetastar = acos((pnew/rA-1)/enew);

theta = linspace(0,2*pi,1000);
thetaE = linspace(pi,3*pi,1000);
thetaT = linspace(pi+thetastarD,pi+thetastarD+2*pi,1000);
thetaN = linspace(thetastar,thetastar+2*pi,1000);
sun = linspace(Rs,Rs,1000);
polar(theta,pm./(1+em*cos(theta)),'r')
hold on
polar(thetaE,pe./(1+ee*cos(theta)))
polar(thetaE,p./(1+e*cos(theta)),'k')
polar(theta,pnew./(1+enew*cos(thetaN)),'g')
polar(theta,sun,'y')
hold off
legend('Mars orbit','Earth orbit','Hohmann transfer','Orbit after swingby')
title('Free Return: Mars swingby')


% One and a half year free-return orbit
re = 149597886.5;
ne = sqrt(mu/re^3);
am = 227936636;
nm = sqrt(mu/am^3);
em = .09341233;
bm = am*sqrt(1-em^2);
pm = am*(1-em^2);
rp = re;
IP = 1.5*365.25*24*60*60;
a = ((IP/(2*pi))^2*mu)^(1/3);
e = 1-rp/a;
p = a*(1-e^2);
ra = a*(1+e);
```

```
rp_LEO = Re+300;
IP_LEO = 7*60*60*24;
a_LEO = ((IP_LEO/(2*pi))^2*mue)^(1/3);
e_LEO = 1-rp_LEO/a_LEO;
ve_LEO = sqrt(2*mue/rp_LEO-mue/a_LEO);
vplus = sqrt(2*mu/re-mu/a);
vearth = sqrt(mu/re);
vxe = vplus-vearth;
dvi = sqrt(vxe^2+2*mue/rp_LEO)-ve_LEO

% Mars passes through this point on
%       October 28, 2004
thetastarM = -acos((pm/ra-1)/em);
% --------------------------------

E = 2*atan(tan(thetastarM/2)/sqrt((1+em)/(1-em)));
t_postap = (pi+E-em*sin(pi+E))/nm;
mars_year = 2*pi/nm;

E2 = E;
earth_day = 60*60*24;
s = linspace(1,14,14);
for k = 1:10
    E2 = E2-(E2-em*sin(E2)-nm*60*60*24*s)/(1-em*cos(E2));
end
E2 = E2+E;
r_window = sqrt((am*(cos(E2)-em)).^2+(bm*sin(E2)).^2);

v_mars = sqrt(2*mu./r_window-mu/am);
v_minus = sqrt(2*mu./r_window-mu/a);
gamma_minus = acos(sqrt(mu*p)./(r_window.*v_minus));
v_xm = sqrt(v_mars.^2+v_minus.^2-2*v_mars.*v_minus.*cos(gamma_minus));
rp_LMO = Rm+300;
IP_LMO = 7*60*60*24;
a_LMO = ((IP_LMO/(2*pi))^2*mum)^(1/3);
e_LMO = 1-rp_LMO/a_LMO;
ve_LMO = sqrt(2*mum/rp_LMO-mum/a_LMO);
d_vf = sqrt(v_xm.^2+2*mum/rp_LMO)-ve_LMO;
dv_freetot = dvi+d_vf';

theta = linspace(0,2*pi,1000);
thetaT = linspace(thetastarM-pi,thetastarM+pi,1000);
sun = linspace(Rs,Rs,1000);
rev = linspace(re,re,1000);
r_int = linspace(0,ra,1000);
theta_int = linspace(thetastarM,thetastarM,1000);
rpT = linspace(0,rp,1000);
theta_rp = linspace(thetastarM-pi,thetastarM-pi,1000);
figure
polar(theta,pm./(1+em*cos(theta)),'r')
hold on
polar(theta,rev,'b')
polar(thetaT,p./(1+e*cos(theta)),'k')
polar(theta,sun,'y')
```

```
hold off
title('Free Return: 1.5 year period orbit')
legend('Mars orbit','Earth orbit','transfer ellipse')


% Kevin Kloster
% AAE450
% deltav_slip.m
% This code calculates the delta v penalty
% that we pay for each day the CTV launch slips.
% The inputs are launch opportunity and days of slip.
% The output is actual delta required.

clc
clear all
close all
format long g

mue = 398600.4418;
Re = 6378.1;
mum = 42828.2869;
Rm = 3397;
mu = 132712200000;
Rs = 695990;
ae = 149597886.5;
ee = .01671022;
pe = ae*(1-ee^2);
ne = sqrt(mu/ae^3);
am = 227936636;
nm = sqrt(mu/am^3);
em = .09341233;
bm = am*sqrt(1-em^2);
pm = am*(1-em^2);

% Ephemeris (from STK 6.0)
Load_Data
d = 1;  % launch date (1-6)
dates = [('December 30, 2015');
         ('March 7, 2018    ');
         ('June 26, 2020    ');
         ('August 3, 2022   ');
         ('October 17, 2024 ');
         ('October 27, 2026 ')];
r_Eo = e_pos(d,1:3);
v_Eo = e_pos(d,4:6);
r_Mo = m_pos(d,1:3);
v_Mo = m_pos(d,4:6);

% Earth parking orbit
rp_LEO = Re+200;
IP_LEO = 10*60*60*24;
a_LEO = ((IP_LEO/(2*pi))^2*mue)^(1/3);
e_LEO = 1-rp_LEO/a_LEO;
v_LEO = sqrt(2*mue/rp_LEO-mue/a_LEO);
```

```
% Mars parking orbit
rp_LMO = Rm+350;
IP_LMO = 7*60*60*24;
a_LMO = ((IP_LMO/(2*pi))^2*mum)^(1/3);
e_LMO = 1-rp_LMO/a_LMO;
v_LMO = sqrt(2*mum/rp_LMO-mum/a_LMO);

% Launch slip
t_slip_day = [-63 -60 -58 -56 -54 -53 -52 -50 -48 -46 -43;
              -69 -66 -64 -62 -60 -59 -58 -56 -54 -52 -49;
              -32 -29 -27 -25 -23 -22 -21 -19 -17 -15 -12;
              -46 -43 -41 -39 -37 -36 -35 -33 -31 -29 -26;
               -9  -6  -4  -2   0   1   2   4   6   8  11;
              -24 -21 -19 -17 -15 -14 -13 -11  -9  -7  -4];
t_slip = 24*60*60*t_slip_day(d,:);
Ee = 0;
for k = 1:10
    Ee = Ee-(Ee-ee*sin(Ee)-ne*t_slip)/(1-ee*cos(Ee));
end
fe = 1-ae/norm(r_Eo)*(1-cos(Ee));
ge = t_slip-1/ne*(Ee-sin(Ee));

Em = 0;
for k = 1:10
    Em = Em-(Em-em*sin(Em)-nm*t_slip)/(1-em*cos(Em));
end
fm = 1-am/norm(r_Mo)*(1-cos(Em));
gm = t_slip-1/nm*(Em-sin(Em));

for j = 1:length(t_slip)
    r_E(j,:) = fe(j)*r_Eo+ge(j)*v_Eo;
    fedot(j) = -sqrt(mu*ae)/(norm(r_E(j,:))*norm(r_Eo))*sin(Ee(j));
    gedot(j) = 1-ae/norm(r_E(j,:))*(1-cos(Ee(j)));
    v_E(j,:) = fedot(j)*r_Eo+gedot(j)*v_Eo;
    r_M(j,:) = fm(j)*r_Mo+gm(j)*v_Mo;
    fmdot(j) = -sqrt(mu*am)/(norm(r_M(j,:))*norm(r_Mo))*sin(Em(j));
    gmdot(j) = 1-am/norm(r_M(j,:))*(1-cos(Em(j)));
    v_M(j,:) = fmdot(j)*r_Mo+gmdot(j)*v_Mo;

    % Transfer ellipse geometry
    rp = norm(r_E(j,:));
    IP = 1.5*365.25*24*60*60;
    a = ((IP/(2*pi))^2*mu)^(1/3);
    e = 1-rp/a;
    p(j) = a*(1-e^2);
    ra = a*(1+e);
    n = sqrt(mu/a^3);

    TA = acos(dot(r_E(j,:),r_M(j,:))/(norm(r_E(j,:))*norm(r_M(j,:))));
    E = 2*atan(tan(TA/2)/sqrt((1+e)/(1-e)));
    t(j) = 1/n*(E-e*sin(E));
    f(j) = 1-a/norm(r_E(j,:))*(1-cos(E));
    g(j) = t(j)-1/n*(E-sin(E));
    fdot(j) = -sqrt(mu*a)/(norm(r_M(j,:))*norm(r_E(j,:)))*sin(E);
    gdot(j) = 1-a/norm(r_M(j,:))*(1-cos(E));
    v_D(j,:) = (r_M(j,:)-f(j)*r_E(j,:))/g(j);
    v_xe(j,:) = v_D(j,:)-v_E(j,:);
    dvi(j) = sqrt(norm(v_xe(j,:))^2+2*mue/rp_LEO)-v_LEO;

    v_A(j,:) = fdot(j)*r_E(j,:)+gdot(j)*v_D(j,:);
    v_xm(j,:) = v_A(j,:)-v_M(j,:);
    dvf(j) = sqrt(norm(v_xm(j,:))^2+2*mum/rp_LMO)-v_LMO;

    h_hat(j,:) = cross(r_E(j,:),v_D(j,:))/norm(cross(r_E(j,:),v_D(j,:)));
    inc(j) = acos(h_hat(j,3));
    Omega(j) = asin(h_hat(j,1)/sin(inc(j)));
    Omegachk(j) = acos(-h_hat(j,2)/sin(inc(j)));
    r_hat(j,:) = r_E(j,:)/norm(r_E(j,:));
    theta_hat(j,:) = cross(h_hat(j,:),r_hat(j,:));
    theta(j) = asin(r_hat(j,3)/sin(inc(j)));
    thetachk(j) = acos(theta_hat(j,3)/sin(inc(j)));
    w(j) = theta(j);   % thetastar is always zero initially for this orbit
end

dvi = dvi'
dvf = dvf'
dvtot = dvi+dvf
[dv day] = min(dvtot);
best_deltav = dv
best_day = t_slip(day)/24/60/60;
TOF = t'/24/60/60

plot(t_slip/24/60/60-best_day,dvi,'b')
hold on
plot(t_slip/24/60/60-best_day,dvf,'r--')
title(['\DeltaV range for ',dates(d,:)])
xlabel('slip time (days)')
ylabel('\DeltaV (km/s)')
legend('Earth escape \Deltav','Mars capture \Deltav',0)
hold off

h = 6;  % slip day (6 is optimal, 1 is 10 days early, 11 is 10 days late)
t_wait = 24*60*60:24*60*60:565*24*60*60;

Em2 = 0;
for k = 1:10
    Em2 = Em2-(Em2-em*sin(Em2)-nm*t_wait)/(1-em*cos(Em2));
end
fm2 = 1-am/norm(r_M(h,:))*(1-cos(Em2));
gm2 = t_wait-1/nm*(Em2-sin(Em2));

t_match = 200*24*60*60;  %initial guess for TOF return
for k = 1:length(t_wait)
    for f = 1:20
        Ee2 = 0;
        for z = 1:10
            Ee2 = Ee2-(Ee2-ee*sin(Ee2)-ne*(t_wait+t(h)+t_match))/(1-ee*cos(Ee2));
        end
        fe2 = 1-ae/norm(r_E(h,:))*(1-cos(Ee2));
        ge2 = t_wait+t(h)-1/ne*(Ee2-sin(Ee2));
        r_E2(k,:) = fe2(k)*r_E(h,:)+ge2(k)*v_E(h,:);
        fedot2(k) = -sqrt(mu*ae)/(norm(r_E2(k,:))*norm(r_E(h,:)))*sin(Ee2(k));
        gedot2(k) = 1-ae/norm(r_E2(k,:))*(1-cos(Ee2(k)));
        v_E2(k,:) = fedot2(k)*r_E(h,:)+gedot2(k)*v_E(h,:);
        r_M2(k,:) = fm2(k)*r_M(h,:)+gm2(k)*v_M(h,:);
        fmdot2(k) = -sqrt(mu*am)/(norm(r_M2(k,:))*norm(r_M(h,:)))*sin(Em2(k));
        gmdot2(k) = 1-am/norm(r_M2(k,:))*(1-cos(Em2(k)));
        v_M2(k,:) = fmdot2(k)*r_M(h,:)+gmdot2(k)*v_M(h,:);

        c(k,:) = r_M2(k,:)-r_E2(k,:);
        s = .5*(norm(r_E2(k,:))+norm(r_M2(k,:))+norm(c(k,:)));
        a2 = s/2;  % use a_min
        alpha = 2*asin(sqrt(s/(2*a2)));
        beta = 2*asin(sqrt((s-norm(c(k,:)))/(2*a2)));
        p2 = 4*(a2)*(s-norm(r_E2(k,:)))*(s-
norm(r_M2(k,:)))/(norm(c(k,:)))^2*sin((alpha+beta)/2)^2;
        e2 = sqrt(1-p2/a2);
        e2v(k) = e2;
        n2 = sqrt(mu/a2^3);

        TA2(k) = acos(dot(r_E2(k,:),r_M2(k,:))/(norm(r_E2(k,:))*norm(r_M2(k,:))));
        E2 = 2*atan(tan(TA2(k)/2)/sqrt((1+e2)/(1-e2)));
        t_match = 1/n2*(E2-e2*sin(E2));
    end
    t2(k)=t_match;
    f2(k) = 1-a2/norm(r_E2(k,:))*(1-cos(E2));
    g2(k) = t2(k)-1/n2*(E2-sin(E2));
    fdot2(k) = -sqrt(mu*a2)/(norm(r_M2(k,:))*norm(r_E2(k,:)))*sin(E2);
    gdot2(k) = 1-a2/norm(r_M2(k,:))*(1-cos(E2));

    v_D2(k,:) = (r_M2(k,:)-f2(k)*r_E2(k,:))/g2(k);
    v_xe2(k,:) = v_D2(k,:)-v_E2(k,:);
    dvi2(k) = sqrt(norm(v_xe2(k,:))^2+2*mue/rp_LEO)-v_LEO;

    v_A2(k,:) = fdot2(k)*r_E2(k,:)+gdot2(k)*v_D2(k,:);
    v_xm2(k,:) = v_A2(k,:)-v_M2(k,:);
    dvf2(k) = sqrt(norm(v_xm2(k,:))^2+2*mum/rp_LMO)-v_LMO;
end

dvtot2 = dvi2'+dvf2';
[dv2 day2] = min(dvtot2);
best_dv2 = dv2
best_day2 = t_wait(day2)/24/60/60
TOF2 = t2(day2)/24/60/60

figure(2)
plot(t_wait/24/60/60,dvtot2)
title('\DeltaV_T_O_T for Return Mission')
xlabel('days on Mars')
ylabel('\Deltav (km/s)')
figure(3)
plot(t_wait/24/60/60,TA2*180/pi)
% Kevin Kloster
% AAE450
% dv_prop_LPPO.m
% CTV from LEO to LPPO
% Orients CTV LPPO to shoot CTV
% into transfer orbit to Mars.
% For a given LPPO geometry (usually defined by the period of the
% orbit) this code outputs the inclination, argument of periapsis,
% delta v and radius at which the CTV needs to get to the transfer
% orbit to Mars.
% ** Analysis in this code ultimately
%    led to cancelation of LPPO
%    CTV is now jumps onto the Mars
%    transfer orbit straight from LEO

clc
clear all
close all
format long g

m0 = 345886;    % kg
mdot = 101.2;   % kg/s
Isp = 1007.6;   % s
g = 9.80665;    % m/s^2
m_prop = 60000; % kg

dv = g*Isp*log((m_prop+m0)/m0)/1000;  % deltav from extra tank (LEO-->LPPO)

mue = 398600.4414;
Re = 6378.1;
rp_LEO = Re+200;
v_LEO = sqrt(mue/rp_LEO);
vp_LPPO = v_LEO+dv;
a = mue/(2*mue/rp_LEO-(vp_LPPO)^2);
e = 1-rp_LEO/a;
p = a*(1-e^2);
IP_LPPO = 2*pi*sqrt(a^3/mue);
ra_LPPO = a*(1+e);
va_LPPO = sqrt(2*mue/ra_LPPO-mue/a);
d = 1;  % 1-6 - launch dates
slip = 6; % 1-11, 6 is optimal day
[vxe inc vdx vdy vdz] = trajectory_vxe(d,slip,IP_LPPO);
a_hyp = mue/vxe^2;
e_hyp = rp_LEO/a_hyp+1;
theta_max = acos(-1/e_hyp);
vdepart = [vdx vdy vdz];
vd_hat = vdepart/norm(vdepart);
rp_hatz = vd_hat(3);
c = cos(theta_max)+vd_hat(3)^2;
rp_hatx = -(vd_hat(2)*sqrt((vd_hat(1)^2+vd_hat(2)^2)^2-c^2)-
vd_hat(1)*c)/(vd_hat(1)^2+vd_hat(2)^2);
rp_haty = sqrt(vd_hat(1)^2+vd_hat(2)^2-rp_hatx^2);
rp_hat = [rp_hatx rp_haty rp_hatz];
rp_LEO_3D = rp_LEO*rp_hat;
vp_LPPO_3D = vp_LPPO*[-rp_haty rp_hatx rp_hatz];
h_hat = cross(rp_LEO_3D,vp_LPPO_3D)/norm(cross(rp_LEO_3D,vp_LPPO_3D));
inc2 = acos(h_hat(3));
% Quad Check
```

```
if h_hat(1)/sin(inc2)>=0 & -h_hat(2)/sin(inc2)>=0 Omega = acos(-h_hat(2)/sin(inc2));
elseif h_hat(1)/sin(inc2)>=0 & -h_hat(2)/sin(inc2)<=0 Omega = acos(-
h_hat(2)/sin(inc2));
elseif h_hat(1)/sin(inc2)<=0 & -h_hat(2)/sin(inc2)<=0 Omega = -acos(-
h_hat(2)/sin(inc2));
elseif h_hat(1)/sin(inc2)<=0 & -h_hat(2)/sin(inc2)>=0 Omega = -acos(-
h_hat(2)/sin(inc2));
end
theta_hat = cross(h_hat,rp_hat);
if rp_hat(3)/sin(inc2)>=0 & theta_hat(3)/sin(inc2)>=0 theta =
acos(rp_hat(3)/sin(inc2));
elseif rp_hat(3)/sin(inc2)>=0 & theta_hat(3)/sin(inc2)<=0 theta =
acos(rp_hat(3)/sin(inc2));
elseif rp_hat(3)/sin(inc2)<=0 & theta_hat(3)/sin(inc2)<=0 theta = -
acos(rp_hat(3)/sin(inc2));
elseif rp_hat(3)/sin(inc2)<=0 & theta_hat(3)/sin(inc2)>=0 theta = -
acos(rp_hat(3)/sin(inc2));
end

w = theta;
rnode_LPPO = p/(1+e*cos(Omega));
vnode_LPPO = sqrt(2*mue/rnode_LPPO-mue/a);
dv_inc = 2*vnode_LPPO*sin(5*pi/180-inc2);
dvtot = dv+dv_inc
dv_escape_LPPO = sqrt(vxe^2+2*mue/rp_LEO)-vp_LPPO
dv_escape_LEO = sqrt(vxe^2+2*mue/rp_LEO)-v_LEO


% trajectory_vxe.m
% this is a function file accessed by dv_prop_LPPO.m.
% the inputs are launch opportunity, days slipped, period of LPPO.
% outputs are v_infinity, inclination, and the x, y, and z components
% of the departure velocity.
function [vxe inc vdx vdy vdz] = trajectory_vxe(d,slip,IP_LPPO)

mue = 398600.4418;
Re = 6378.1;
mum = 42828.2869;
Rm = 3397;
mu = 132712200000;
Rs = 695990;
ae = 149597886.5;
ee = .01671022;
pe = ae*(1-ee^2);
ne = sqrt(mu/ae^3);
am = 227936636;
nm = sqrt(mu/am^3);
em = .09341233;
bm = am*sqrt(1-em^2);
pm = am*(1-em^2);

% Ephemeris (from STK 6.0)
Load_Data
dates = [('December 30, 2015');
         ('March 7, 2018    ');
         ('June 26, 2020    ');
         ('August 3, 2022   ');
         ('October 17, 2024 ');
         ('October 27, 2026 ')];
r_Eo = e_pos(d,1:3);
v_Eo = e_pos(d,4:6);
r_Mo = m_pos(d,1:3);
v_Mo = m_pos(d,4:6);

% Earth parking orbit
rp_LEO = Re+200;
IP_LEO = IP_LPPO;
a_LEO = ((IP_LEO/(2*pi))^2*mue)^(1/3);
e_LEO = 1-rp_LEO/a_LEO;
v_LEO = sqrt(2*mue/rp_LEO-mue/a_LEO);

% Mars parking orbit
rp_LMO = Rm+350;
IP_LMO = 1*60*60*24;
a_LMO = ((IP_LMO/(2*pi))^2*mum)^(1/3);
e_LMO = 1-rp_LMO/a_LMO;
v_LMO = sqrt(2*mum/rp_LMO-mum/a_LMO);

% Launch slip
t_slip = 24*60*60*[-10 -7 -5 -3 -1 0 1 3 5 7 10];
Ee = 0;
for k = 1:length(t_slip)
    Ee = Ee-(Ee-ee*sin(Ee)-ne*t_slip)/(1-ee*cos(Ee));
end
fe = 1-ae/norm(r_Eo)*(1-cos(Ee));
ge = t_slip-1/ne*(Ee-sin(Ee));

Em = 0;
for k = 1:length(t_slip)
    Em = Em-(Em-em*sin(Em)-nm*t_slip)/(1-em*cos(Em));
end
fm = 1-am/norm(r_Mo)*(1-cos(Em));
gm = t_slip-1/nm*(Em-sin(Em));

for j = 1:length(t_slip)
    r_E(j,:) = fe(j)*r_Eo+ge(j)*v_Eo;
    fedot(j) = -sqrt(mu*ae)/(norm(r_E(j,:))*norm(r_Eo))*sin(Ee(j));
    gedot(j) = 1-ae/norm(r_E(j,:))*(1-cos(Ee(j)));
    v_E(j,:) = fedot(j)*r_Eo+gedot(j)*v_Eo;
    r_M(j,:) = fm(j)*r_Mo+gm(j)*v_Mo;
    fmdot(j) = -sqrt(mu*am)/(norm(r_M(j,:))*norm(r_Mo))*sin(Em(j));
    gmdot(j) = 1-am/norm(r_M(j,:))*(1-cos(Em(j)));
    v_M(j,:) = fmdot(j)*r_Mo+gmdot(j)*v_Mo;

    % Transfer ellipse geometry
    rp = norm(r_E(j,:));
    IP = 1.5*365.25*24*60*60;
    a = ((IP/(2*pi))^2*mu)^(1/3);
    e = 1-rp/a;
    p = a*(1-e^2);
    ra = a*(1+e);
    n = sqrt(mu/a^3);

    TA = acos(dot(r_E(j,:),r_M(j,:))/(norm(r_E(j,:))*norm(r_M(j,:))));
    E = 2*atan(tan(TA/2)/sqrt((1+e)/(1-e)));
    t(j) = 1/n*(E-e*sin(E));
    f(j) = 1-a/norm(r_E(j,:))*(1-cos(E));
    g(j) = t(j)-1/n*(E-sin(E));
    fdot(j) = -sqrt(mu*a)/(norm(r_M(j,:))*norm(r_E(j,:)))*sin(E);
    gdot(j) = 1-a/norm(r_M(j,:))*(1-cos(E));

    v_D(j,:) = (r_M(j,:)-f(j)*r_E(j,:))/g(j);
    v_xe(j,:) = v_D(j,:)-v_E(j,:);
    dvi(j) = sqrt(norm(v_xe(j,:))^2+2*mue/rp_LEO)-v_LEO;

    v_A(j,:) = fdot(j)*r_E(j,:)+gdot(j)*v_D(j,:);
    v_xm(j,:) = v_A(j,:)-v_M(j,:);
    dvf(j) = sqrt(norm(v_xm(j,:))^2+2*mum/rp_LMO)-v_LMO;

    h_hat(j,:) = cross(r_E(j,:),v_D(j,:))/norm(cross(r_E(j,:),v_D(j,:)));
    inc(j) = acos(h_hat(j,3));
    Omega(j) = asin(h_hat(j,1)/sin(inc(j)));
    Omegachk(j) = acos(-h_hat(j,2)/sin(inc(j)));
    r_hat(j,:) = r_E(j,:)/norm(r_E(j,:));
    theta_hat(j,:) = cross(h_hat(j,:),r_hat(j,:));
    theta(j) = asin(r_hat(j,3)/sin(inc(j)));
    thetachk(j) = acos(theta_hat(j,3)/sin(inc(j)));
    w(j) = theta(j);   % thetastar is always zero initially for this orbit
end

dvi = dvi';
dvf = dvf';
dvtot = dvi+dvf;
[dv day] = min(dvtot);
best_deltav = dv;
best_day = t_slip(day)/24/60/60;
TOF = t'/24/60/60;
date = dates(d,:);
depart_velocity = v_D;
vxe = norm(v_xe(slip,:));
inc = inc(slip);
vdx = v_D(slip,1);
vdy = v_D(slip,2);
vdz = v_D(slip,3);
return
```

26 Kottlowski, Aaron D.

26.1.1 Kottlowski - Appendix – Nuclear Power Reactor for CTV and MHV

Author(s): Aaron Kottlowski

The following are codes used to determine the reactor parameters. They work in sequence in ultimately yield the thermal power produced, radiation products, and safe distance (in the case of the MHV) for a given electrical power input.

```
%AAE 450 - Spacecraft Design
%Spring 2005
%Power Group
%Aaron Kottlowski

%Reactor Power Code
clc
%INPUT: Electrical Power Requirements, Fraction AC Power

Pe = 140;   %Power Required [kWe]
Pwaste = Cable_Length*.04;    %Power disipated in cables from reactor
ACf = .30;     %Fraction of power produced converted to AC

%Calculate Electrical Losses

%AC Conversion Losses
ACr = Pe*ACf;    %Total AC Power Required [kWe]
InvE = .94;       %DC-AC Inverter Efficiency
ACt = ACr/InvE;   %Power Required of system for AC power [kWe]

%DC Power Conditioning Losses (Comming soon)

%Reactor Power Operating Requirements (pumps, controllers, etc.)
Reacr = 5;  %Power Required to operate reactor [kW]

%Total Electrical Power Required from TE Converters
Ptot = Pe-ACr+ACt+Reacr+Pwaste;

%TE Converter Input
TECe = .17;  %Efficiency of TE Converter

%Heat Transfer Loop 1
HTe = .98;      %Efficiency of Inner Heat transfer loop

%Reactor Thermal Requirements
Pt = Ptot/(TECe*HTe)   ; %Total Thermal Power Required from Reactor [kWt]

%Thermal Waste
Wt = Pt-Ptot;


%At Core Surface
%Fast Neutron Flux Approximation (Linear)
fnf1 = (Pt/1000)*(1.3*10^13)/2.3;

%Gamma Spectrum Radiation Approximation
gr1 = (Pt/1000)*(2.3*10^3)/2.3;

%At Reactor Shield
```

```
%Fast Neutron Flux Approximation (Linear)
fnf2 = (Pt/1000)*(1.3*10^7)/2.3;

%Gamma Spectrum Radiation Approximation
gr2 = (Pt/1000)*(9*10^-1)/2.3;

%At > 23m
%Fast Neutron Flux Approximation (Linear)
fnf3 = (Pt/1000)*(2*10^4)/2.3;

%Gamma Spectrum Radiation Approximation
gr3 = (Pt/1000)*(2.3*10^-3)/2.3;


%Outputs
fprintf('\n************** Space Reactor Code v2.6 **************\n')
fprintf('Electrical Power Required: \t\t%3.2f\t\tkWe\n',Pe)
fprintf('Electrical Power Produced: \t\t%3.2f\t\tkWe\n',Ptot)
fprintf('Electrical Power Cable Losses: \t%3.2f\t\tkWe\n',Pwaste)
fprintf('Reactor Thermal Power: \t\t\t%3.2f\t\tMWt\n',(Pt/1000))
fprintf('Thermal Waste: \t\t\t\t%3.2f\t\tMWt\n',(Wt/1000))
fprintf('Radidation at Reactor Core Surface\n')
fprintf('Fast Neutron Flux: \t\t\t\t%3.2g\tn/cm^2-sec\n',fnf1)
fprintf('Gamma Radiation: \t\t\t\t%3.2g\trad/sec\n',gr1)
fprintf('Radidation at Reactor Shield\n')
fprintf('Fast Neutron Flux: \t\t\t\t%3.2g\tn/cm^2-sec\n',fnf2)
fprintf('Gamma Radiation: \t\t\t\t%3.2g\t\trad/sec\n',gr2)
fprintf('Radidation at greater than 23m\n')
fprintf('Fast Neutron Flux: \t\t\t\t%3.2g\tn/cm^2-sec\n',fnf3)
fprintf('Gamma Radiation: \t\t\t\t%3.2g\t\trad/sec\n',gr3)
fprintf('*****************By: Aaron Kottlowski****************\n')


%AAE 450 - Spacecraft Design
%Spring 2005
%Power Group
%Aaron Kottlowski
%Radiation Distance Attenuation Calculator
clc
close all

Powercode1
%How Many days?
d = 250;

%Core Radiation Flux
gamma = gr2;   %Gamma Radiation (Core Shielded)   [rads/sec]
fnf = fnf2;    %Fast Neutron Flux (Core Shielded) [n/cm^2-sec]

%Gamma Distance Attenuation
r = linspace(95,500,1000);     %Distance from reactor core
Q = 1;    %Radiation Quality Factor
Sg = gamma*3600*24*d*Q;    %Total Gamma Flux for 500 days on Martian surface [rads]
phi_gamma = Sg./(4.*pi.*r.^2);

%Fast Neutron Distance Attenuation
Sn = fnf;     %Fast Neutron Flux for 500 days on Martian surface [n/cm^2-sec]
phi_nn = Sn./(4.*pi.*r.^2);     %FNF at distance r  [n/cm^2-sec]
FNFe = 6;   %Fast Neutron Energy [MeV]
phi_ni = (phi_nn./7)*1;   %FNF conversion to [mREM/hr]
phi_n = (phi_ni./1000).*24.*d;   %FNF Total dose calculation [REM]

%Sum Radiation Products
RT = phi_gamma+phi_n;

%Find Distance for 10 REM
Di = find(RT<=10);
Cable_Length = r(Di(1))

%Plot Gamma  and Fast Neutron flux at a distance
figure(1)
```

```
AXIS([90 400 0 90])
hold on
plot(r,phi_gamma,'b')
plot(r,phi_n,'c')
plot(r,RT,'m')
plot(r,10,'g')
plot(r,25,'y')
plot(r,50,'r')
xlabel('Distance from core [m]')
ylabel('Total Gamma Flux [REM]')
title('Total Gamma and Fast Neutron Flux from Reactor (~250 days)')
legend('Gamma Flux','Fast Neutron Flux','Total Radition Flux')
```

The following code calculates the total mass of the space reactor system as a function of how many TE panels are used in the design.

```
%AAE 450 - Spacecraft Design
%Spring 2005
%Power Group
%Aaron Kottlowski
%Power Reactor Mass Calculation Code
clc

%****Input Number of Panels
pan = 7;
%****Pipe/Coolant Mass Calculation*****
%Density Inputs
rhop = 4480;         %Density of Titanium Ti-3Al-2.5V [kg/m^3]
rhoc = 530;          %Density of Liquid Li [kg/m^3]

L = [pan*3.9 pan*2.3];              %Total Length of Coolant Piping

%Pipe Size
pod = .0508;             %Pipe OD [m]
pid = .04445;             %Pipe ID [m]

acs = (pi*(pod/2)^2)-(pi*(pid/2)^2);   %Area of Pipe Cross section

vp = L.*acs;              %Volume of Pipe [m^3]
mp = vp.*rhop;             %Mass of Pipe [kg]

vc = L.*.001552          %Volume of Coolant [m^3]
mc = vc.*rhoc             %Mass of Li Coolant [kg]

mt = mp+mc;               %Total mass of coolant piping system
%***************************************

%*****Heat Exchanger Mass Calculation*****
vp = .48*.025*1.025;       %Volume of 1 panel

tmr = vp*pan*rhoc;            %Total mass of 9 HE Panels (Hot side only)
%******************************************

%*****Pressure Vessle Mass Calculations*****
mpv = 86.787;              %Mass of single pressure vessle main body [kg] from CATIA
rhoU = 19070;              %Density of U235 [kg/m^3]
vcore = (pi*.12^2)*.62;    %Volume of Fueled Core [m^3]
mcore = vcore*rhoU;          %Mass of Core for single reactor [kg]

mpve = 37.702+20.356;        %Mass of PV Endcaps
mpumps = 2.648*pan;            %Mass of all coolant pumps

%*********************************************

%*****Mass of Radiation Shield*****
mshield = 0;              %Mass of Radiation Shield [kg]
```

```
%**********************************

mte = 4.44*pan;               %Mass of TE converter modules
mpcon = 264;                %Mass of power conditioning system

%*****Total System Mass*************
mt2c = mt(1)+tmr+mpv+mcore*2+mpve*2+mpumps+mshield+mpcon+mte
mt1c = mt(2)+tmr+mpv+mcore+mpve+mpumps+mshield*.75+mpcon+mte

PowerProduced = pan*(200/9)     %[kWe]
```

The following code calculates the necessary design parameters for TE modules given the hot and cold side temperatures.

```
%Aaron Kottlowski
%Thermoelectric Module Parameter Code
%AAE 450 - Spacecraft Design

%Thermal Conditions
Th = 1350;       %Hot side Temp [K]
Tc = 800;        %Cold side Temp [K]

%Calculate Seebeck coefficient
v = 6;           %Volts/cell [volts]
dT = Th-Tc;   %Temerature Differential across module [K]

S = v/dT         %Seebeck Coefficient [v/deg K]

%Calculate Current drawn from module
Rc = .00001;     %Internal Resistance of Module [ohms]
Rl = .5;         %Load Resistance [ohms]

I = (S*dT)/(Rc*Rl)  %Load Current

%Calculate Required Heat Flux
Kc = 1/.59;      %Thermal Conductance of Module
Qh = (S*Th*I)-(.5*I^2*Rc)+(Kc*dT)   %Heat input [W]


%Calculate Module Efficiency
Eg = (v*I)/Qh
```

The following code calculates the number of TE modules necessary to obtain the required power output given the thermal conditions.

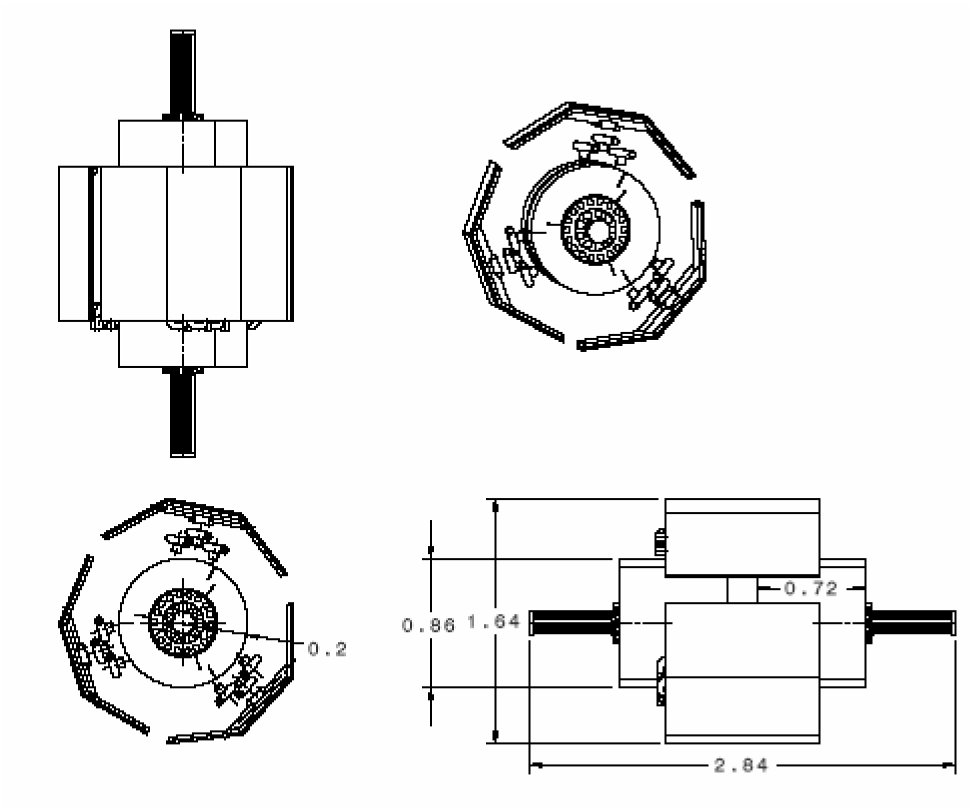
```
%AAE 450 - Spacecraft Design
%Spring 2005
%Power Group
%Aaron Kottlowski
%Thermoelectric Module Code 1

clc
%Power Required
P = 200/9;      %Power generated per power panel

%Thermal Conditions
Th = 1350;       %Hot side Temp [K]
Tc = 800;        %Cold side Temp [K]
```

```
%Calculate Seebeck coefficient
v = 6;          %Volts/cell [volts]
dT = Th-Tc;   %Temerature Differential across module [K]

S = v/dT         %Seebeck Coefficient [v/deg K]

%Calculate Current drawn from couple
Rc = 1.7;     %Internal Resistance of Module [ohms]
Rl = 10;         %Load Resistance [ohms]

I = (S*dT)/(Rc*Rl)  %Load Current

%Calculate Required Heat Flux
Kc = 1/.59;      %Thermal Conductance of couple
Qh = (S*Th*I)-(.5*I^2*Rc)+(Kc*dT)   %Heat input [W]


%Calculate couple Efficiency
Eg = (v*I)/Qh

%Calculate Power output per module [W]
Po = Rl*((S*dT)/(Rc+Rl))^2
```

The following code calculated the necessary power to start the nuclear reactor.

```
%AAE 450 - Spacecraft Design
%Spring 2005
%Power Group
%Aaron Kottlowski
%Reactor Start-up Power Code
clc
pipemass;

c = 3305;   %Heat Capacity of Li [J/kg-C]
TRi = 10;   %Initial Reactor Temp [K]
dT = 443.7-TRi; %Total Change in Temp
mc;     %Mass(s) of coolant
Q = c*mc*dT;    %Energy required to warm up reactor [J]
t = 30;     %Time required to start reactor [Minutes]

P_start = Q/(t*60*1000); %Power required to start up reactor [kW]
n = .98;    %Heating efficiency
Pe = P_start*(1/n)   %Power required to start up reactor [kWe]
```

The following is a dimensioned model of the CTV Power Reactor



27 Long, Angela

27.1 Ascent and Recovery Vehicle (ARV) Entry and Descent Trajectory

Author: Angela Long

27.1.1 Purpose

This appendix describes in detail the procedure by which we implement the theory introduced in the Earth Descent and Entry Trajectory Section. What follows is a description of the MATLAB code needed to numerically integrate the equations of motion governing the descent trajectory.

27.1.2 Significance

The ARV is similar to the Apollo Command Module (CM) in its size, shape, and purpose. While the ARV follows a similar entry trajectory as did the CM, the ARV is traveling considerably faster than the CM when it enters the atmosphere. The increase in entry velocity presents a new problem. This code proves that an entry at these velocities is feasible.

27.1.3 Inputs/Outputs

The entry trajectory simulation code consists of two files: a script file to run the code and a function file containing the equations of motion. The script file initializes the entry velocity based on the entry altitude and energy of the hyperbolic orbit (V_∞). The flight path angle is varied to obtain a feasible solution.

The four phases mentioned in the Earth Descent and Entry Trajectory Section are identified within the script file. We define the lift and drag parameters in the script file. The initial conditions and lift and drag parameters are sent to the function file by calling the function ODE45, which is a numerical integrator within MATLAB. ODE45 returns the values of the state variables (altitude, velocity, and flight path angle) along with a time vector. With this information, the script file calculates the dynamic pressure, aerodynamic forces (lift, drag, and resultant force), and g-loading. Within each phase, the generated variables are checked against the criterion for the termination of that phase. The set of parameters at the end of each phase is defined as the initial conditions for the following phase and the process starts again.

Once the fourth phase is complete (altitude equals zero kilometers), the script file plots the state variables from each phase together on one plot. Specifically, altitude, velocity, flight path angle, g-loading, lift, drag, dynamic pressure, and density are plotted versus time. The code also generates a velocity versus altitude graph. In addition to producing the entry trajectory, the code calculates the amount of ΔV needed to perform the periapsis lower burn, assuming we implement the maneuver one hour prior to atmospheric entry. Below are the plots produced for the entry simulation for a nominal return (launch window 2).

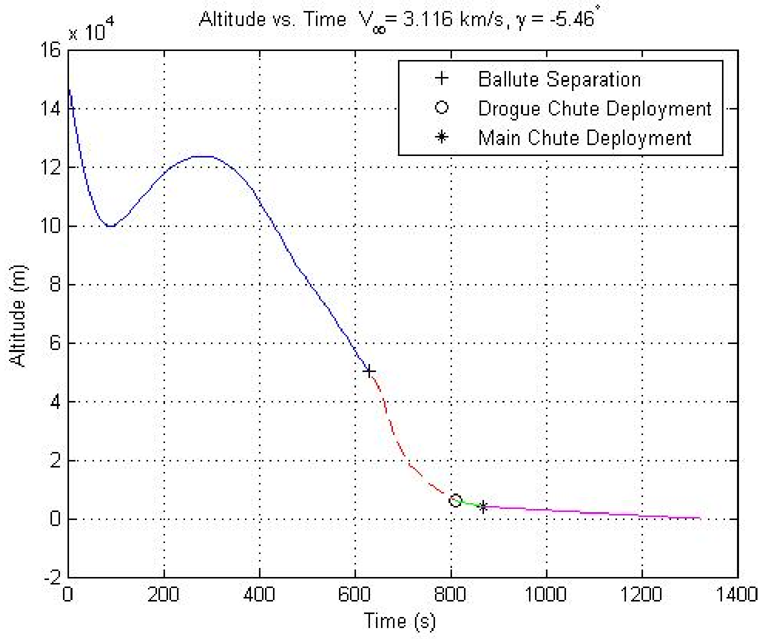


Figure 27-1 - Altitude vs. Time - ARV

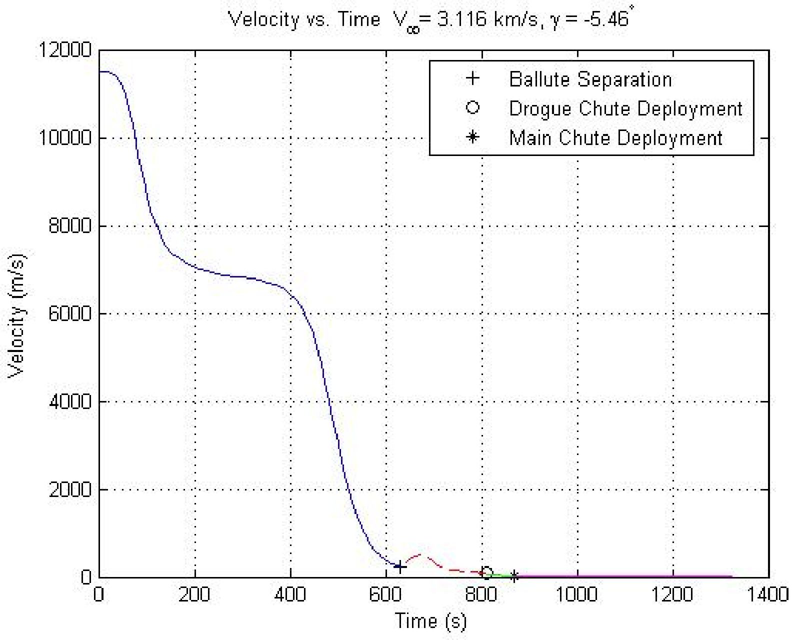


Figure 27-2 - Velocity vs. Time - ARV

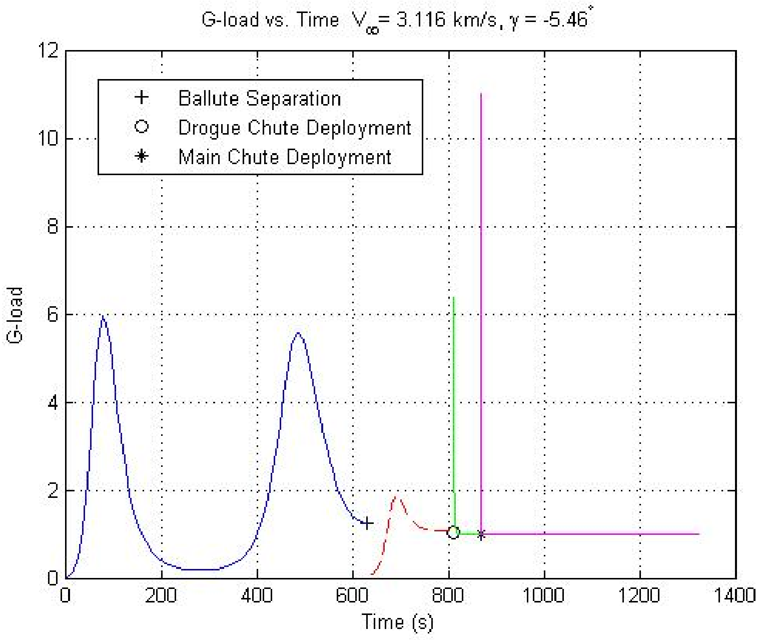


Figure 27-3 - G-loads vs. Time - ARV

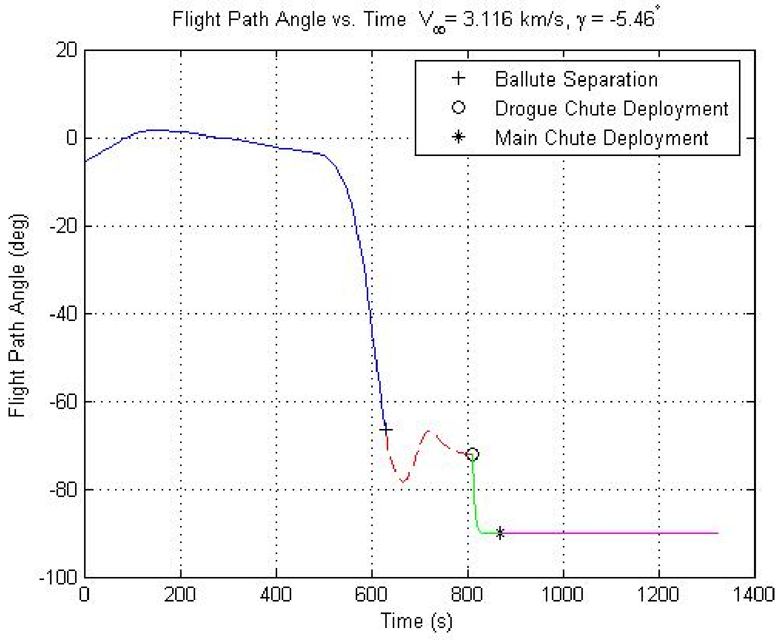


Figure 27-4 - Flight Path Angle vs. Time - ARV

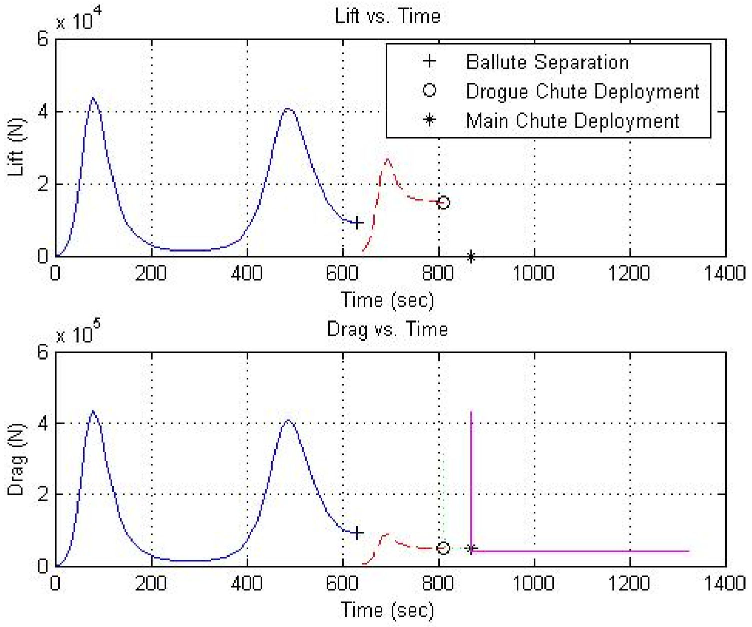


Figure 27-5 - Lift and Drag vs. Time - ARV

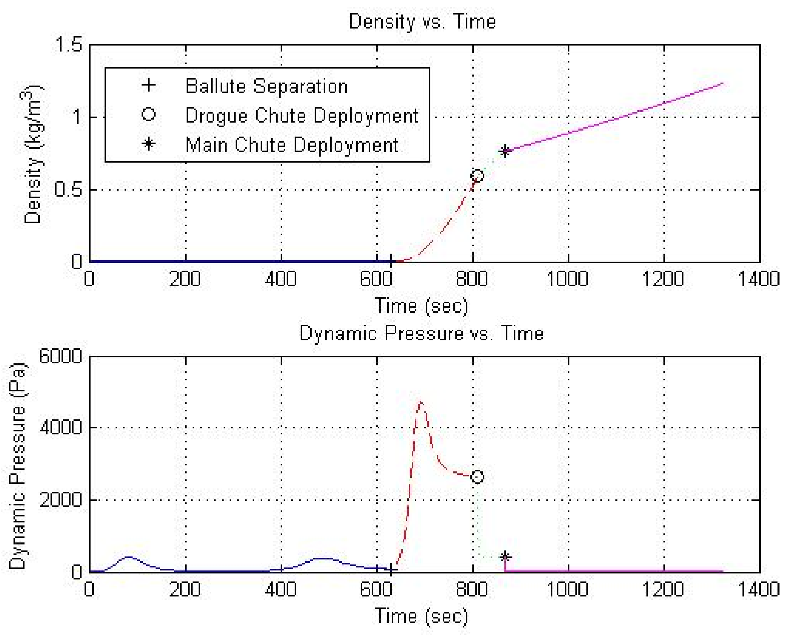


Figure 27-6 - Density and Dynamic Pressure vs. Time - ARV

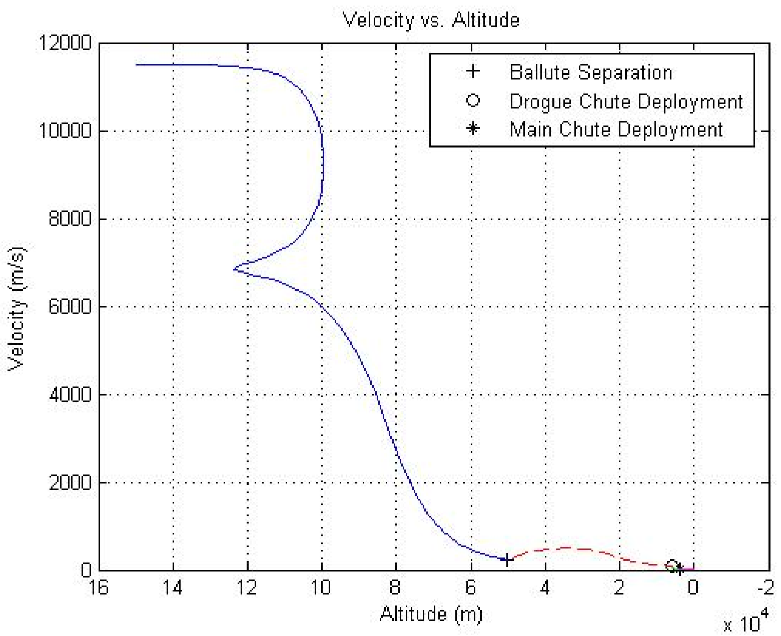


Figure 27-7 - Velocity vs. Altitude - ARV

27.1.4 MATLAB Code

Script file

```
%Angela Long
%AAE 450
%ARV Re-entry trajectory code
%Updated 4/2/05
```

```
close all
clear all
clc

t0= 0.0;
tf = 2000;

% Constants
re = 6378.14e3;                 % m
he = 150e3;                     % m
r = re + he;
mu = 3.986004418e5*1000^3;      % m^3/s^2
rho0 = 1.225;                   % kg/m^3
g = 9.81;                       % m/s^2

% Use v_inf to find entry velocity
vinf = 3.116e3;                 % m/s
a = mu/vinf^2;                  % m
ve = sqrt(2*(mu/r + mu/2/a))    % m/s

%Set flight path angle
gamma = -5.46                   % deg

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Atmospheric Entry - Ballute Deployed
%                           Phase 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Toroidal Ballute drag parameters
m = 7500;
d1 = 60;
d2 = 50;
S1 = pi*d1^2/4;
S2 = pi*d2^2/4;
S = S1 - S2
L_D = 0.1;
cd = 1.25;
cl = L_D*cd;
sigma = 0;

% Set initial conditions for integration
% Altitude, Velocity, Flight Path Angle
u0=[he ve gamma*pi/180];
dt=[t0,tf];
[t,x]=ode45(@EOMtest3,dt,u0,[],m,S,cd,cl,sigma);

%Exponential Denisty Model
rho0 = 1.225;
a = 8.42e3;
rho1 = rho0.*exp(-x(:,1)./a);

%Aerodynamic forces and G-loads
q1 = rho1.*(x(:,2).^2)./2;
L1 = q1.*S.*cl;
D1 = q1.*S.*cd;
Fa1 = sqrt((D1.*cos(x(:,3)) - L1.*sin(x(:,3))).^2 + (D1.*sin(x(:,3)) +...
    L1.*cos(x(:,3))).^2);
a1 = Fa1./m;
gs1 = a1./g;
Gmax1 = max(gs1)

%Phase 1 End Conditions - Deploy ballute when altitude reaches 50 km
anj = 1;
m = size(x,1);
for anj = 1:m-1
if (x(anj,1) < 50000)
    anj = anj - 1;
    break
  end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               Ballute Separation - Heat Shield
%                           Phase 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Heat shield drag parameters
d = 4.5;
S = pi*d^2/4;
m = 6000;
L_D = 0.3;
cd = 1.17;
cl = L_D*cd;
sigma = 0;

%Set initial condition for Phase 2 - end conditions from Phase 1
u1= x(anj,:);
dt=[t0,tf];
[t1,x1]=ode45(@EOMtest3,dt,u1,[],m,S,cd,cl,sigma);

%Exponential density model
rho0 = 1.225;
a = 8.42e3;
rho2 = rho0.*exp(-x1(:,1)./a);

%Aerodynamic forces and G-loads
q2 = rho2.*(x1(:,2).^2)./2;
L2 = q2.*S.*cl;
D2 = q2.*S.*cd;
Fa2 = sqrt((D2.*cos(x1(:,3)) - L2.*sin(x1(:,3))).^2 + (D2.*sin(x1(:,3))+...
    L2.*cos(x1(:,3))).^2);
a2 = Fa2./m;
gs2 = a2./g;
Gmax2 = max(gs2);

%End conditions for Phase 2 - Dynamic Pressure exceeds 700 Pa while
%altitude is below 6 km
slps = 1;
m = size(x1,1);
for slps = 1:m-1
if   (q2(slps,1) > 700) & (x1(slps,1) < 6000)
    slps = slps - 1;
    break
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   Drogue Parachute Deployment
%                           Phase 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Drogue Parachute drag parameters
d = 10;
S = pi*d^2/4;
m = 5000;
L_D = 0;
cd = 1.5;
cl = L_D*cd;
sigma = 0;

%Set initial conditions for Phase 3 - end conditions from Phase 2
u2 = x1(slps,:);
dt=[t0,tf];
[t2,x2]=ode45(@EOMtest3,dt,u2,[],m,S,cd,cl,sigma);

%Exponential density model
rho0 = 1.225;
a = 8.42e3;
rho3 = rho0.*exp(-x2(:,1)./a);

%Aerodynamic forces and G-loads
q3 = rho3.*(x2(:,2).^2)./2;
L3 = q3.*S.*cl;
D3 = q3.*S.*cd;
```

```
Fa3 = sqrt((D3.*cos(x2(:,3)) - L3.*sin(x2(:,3))).^2 + (D3.*sin(x2(:,3))+...
    L3.*cos(x2(:,3))).^2);
a3 = Fa3./m;
gs3 = a3./g;
Gmax3 = max(gs3);

%End conditions for phase 3 - Altitude reaches 4 km
stnky = 1;
m = size(x1,1);
for stnky = 1:m-1
if (x2(stnky,1) < 4000)
    stnky = stnky - 1;
    break
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Parachute Deployment
%                             Phase 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Main parachute drag parameters
d = 25;
S = 3*pi*d^2/4;
m = 4000;
L_D = 0;
cd = 0.7;
cl = L_D*cd;
sigma = 0;

%Set initial conditions for phase 4 - end conditions from phase 3
u3 = x2(stnky,:);
dt=[t0,tf];
[t3,x3]=ode45(@EOMtest3,dt,u3,[],m,S,cd,cl,sigma);

%Exponential density model
rho0 = 1.225;
a = 8.42e3;
rho4 = rho0.*exp(-x3(:,1)./a);

%Aerodynamic forces and G-loads
q4 = rho4.*(x3(:,2).^2)./2;
L4 = q4.*S.*cl;
D4 = q4.*S.*cd;
Fa4 = sqrt((D4.*cos(x3(:,3)) - L4.*sin(x3(:,3))).^2 + (D4.*sin(x3(:,3))+...
    L4.*cos(x3(:,3))).^2);
a4 = Fa4./m;
gs4 = a4./g;
Gmax4 = max(gs4);

%End conditions for phase 4 - altitude reahces 0 km
k = 1;
while x3(k,1) > 0
    if x3(k,1) == x3(end,1)
        break
    else
        k = k + 1;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Orbit Determination
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ah = mu/vinf^2;
rp_lppo = 6378.14e3 + 200e3;
e_lppo = rp_lppo/ah + 1;
b_lppo = ah*sqrt(e_lppo^2 - 1);

r = he + 6378.14e3;
e = sqrt((r*ve*cos(gamma*pi/180))^2/mu/ah + 1);
b = ah*sqrt(e^2 - 1);
rp = ah*(e-1);
```

```
deltab = abs(b_lppo - b);
deltav = 10:10:100;
deltaa = deltav./vinf;
d = deltab./deltaa;
tminus = d./vinf;
t_hours = tminus./3600;

TOF = 3600;
Ph = ah*(e^2 -1);
thetastar1 = -1*acos(1/e*(Ph/r - 1));
H1 = -acosh((e + cos(thetastar1))/(1 + e*cos(thetastar1)));

x0 = 0;
[H2,feval,exitflag] = fsolve(@HyperEccAnom,x0,optimset('fsolve'),...
    TOF,ah,e,mu,H1);

x0 = 1;
[thetastar,feval,exitflag] = fsolve(@HyperTrueAnom,x0,...
    optimset('fsolve'),e,H2);

rh = Ph/(1 + e*cos(thetastar))
rapprox = sqrt(d(10)^2 + b^2);

d = TOF*vinf;
deltaa1 = deltab/d;
deltav1 = deltaa1*vinf

MinV = min(x3(1:k,2))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                            Plot Results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
vinf = vinf/1000;

%Altitude vs Time
figure(1)
plot(t(anj),x(anj,1),'k+')
hold on
grid on
plot(t1(slps)+t(anj),x1(slps,1),'ko')
plot(t2(stnky)+t1(slps)+t(anj),x2(stnky,1),'k*')
plot(t(1:anj),x(1:anj,1),'b')
plot(t1(1:slps)+t(anj),x1(1:slps,1),'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),x2(1:stnky,1),'g')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),x3(1:k,1),'m')
title(['Altitude vs. Time  V_\infty= ' num2str(vinf)...
    ' km/s, \gamma = ' num2str(gamma) '^\circ'])
xlabel('Time (s)')
ylabel('Altitude (m)')
legend('Ballute Separation','Drogue Chute Deployment',...
    'Main Chute Deployment')

%Velocity vs Time
figure(2)
plot(t(1:anj),x(1:anj,2),'b')
hold on
grid on
plot(t1(1:slps)+t(anj),x1(1:slps,2),'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),x2(1:stnky,2),'g')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),x3(1:k,2),'m')
title(['Velocity vs. Time  V_\infty= ' num2str(vinf)...
    ' km/s, \gamma = ' num2str(gamma) '^\circ'])
xlabel('Time (s)')
ylabel('Velocity (m/s)')

%Flight Path Angle vs Time
figure(3)
plot(t(1:anj),x(1:anj,3).*180./pi,'b')
hold on
grid on
plot(t1(1:slps)+t(anj),x1(1:slps,3).*180./pi,'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),x2(1:stnky,3).*180./pi,'g')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),x3(1:k,3).*180./pi,'m')
title(['Flight Path Angle vs. Time  V_\infty= ' num2str(vinf)...
    ' km/s, \gamma = ' num2str(gamma) '^\circ'])
xlabel('Time (s)')
ylabel('Flight Path Angle (deg)')

%G-loads vs Time
figure(4)
plot(t(1:anj),gs1(1:anj),'b')
hold on
grid on
plot(t1(1:slps)+t(anj),gs2(1:slps),'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),gs3(1:stnky),'g')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),gs4(1:k),'m')
title(['G-load vs. Time  V_\infty= ' num2str(vinf) ' km/s, \gamma = '...
    num2str(gamma) '^\circ'])
xlabel('Time (s)')
ylabel('G-load')

%Density and Dynamic Pressure vs Time
figure(5)
subplot(2,1,1)
plot(t(1:anj),rho1(1:anj),'b')
hold on
plot(t1(1:slps)+t(anj),rho2(1:slps),'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),rho3(1:stnky),'g:')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),rho4(1:k),'m')
title('Density vs. Time')
xlabel('Time (sec)')
ylabel('Density (kg/m^3)')

subplot(2,1,2)
plot(t(1:anj),q1(1:anj),'b')
hold on
plot(t1(1:slps)+t(anj),q2(1:slps),'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),q3(1:stnky),'g:')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),q4(1:k),'m')
title('Dynamic Pressure vs. Time')
xlabel('Time (sec)')
ylabel('Dynamic Pressure (Pa)')

%Lift and Drag vs Time
figure(6)
subplot(2,1,1)
plot(t(1:anj),L1(1:anj),'b')
hold on
plot(t1(1:slps)+t(anj),L2(1:slps),'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),L3(1:stnky),'g:')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),L4(1:k),'m')
title('Lift vs. Time')
xlabel('Time (sec)')
ylabel('Lift (N)')

subplot(2,1,2)
plot(t(1:anj),D1(1:anj),'b')
hold on
plot(t1(1:slps)+t(anj),D2(1:slps),'r--')
plot(t2(1:stnky)+t1(slps)+t(anj),D3(1:stnky),'g:')
plot(t3(1:k)+t2(stnky)+t1(slps)+t(anj),D4(1:k),'m')
title('Drag vs. Time')
xlabel('Time (sec)')
ylabel('Drag (N)')

%Delta V vs Time
figure(7)
plot(t_hours,deltav)
hold on
plot(30*ones(size(deltav)),deltav,'r--')
grid on
title(['\DeltaV vs. Time  V_\infty= ' num2str(vinf) ' km/s, \gamma = '...
    num2str(gamma) '^\circ'])
xlabel('Time (h)')
ylabel('\DeltaV (m/s)')

%Velocty vs Altitude
figure(8)
plot(x(anj,1),x(anj,2),'k+')
hold on
grid on
plot(x1(slps,1),x1(slps,2),'ko')
plot(x2(stnky,1),x2(stnky,2),'k*')
plot(x(1:anj,1),x(1:anj,2),'b')
plot(x1(1:slps,1),x1(1:slps,2),'r--')
plot(x2(1:stnky,1),x2(1:stnky,2),'g')
plot(x3(1:k,1),x3(1:k,2),'m')
set(gca,'XDir','reverse')
title('Velocity vs. Altitude')
xlabel('Altitude (m)')
ylabel('Velocity (m/s)')

%Combine all phases into one variable and write to an excel file
%This veloicty profile is used to size the heat shield
x_all = cat(1,x(1:anj,:),x1(1:slps,:),x2(1:stnky,:),x3(1:k,:));
t_all = cat(1,t(1:anj),t1(1:slps)+t(anj),t2(1:stnky)+t(anj)+...
    t1(slps),t3(1:k)+t2(stnky)+t(anj)+t1(slps));
rho_all = cat(1,rho1(1:anj),rho2(1:slps),rho3(1:stnky),rho4(1:k));
profile = cat(2,t_all,x_all,rho_all);
save reentry_data.xls -ASCII -TABS profile
```

ODE45 function file

In the ODE45 function file, the state variables are assigned as such: 1 – altitude, 2 – velocity, 3- flight path angle.

```
function udot = EOMtest(t,u,m,S,cd,cl,sigma)
re = 6378.14e3;                 % m
mu = 3.986004418e5*1000^3;      % m^3/s^2

%Exponential density model
rho0 = 1.225;
a = 8.42e3;
rho = rho0*exp(-u(1)/a);

%Aerodynamic forces
q = rho*(u(2)^2)/2;
L = q*S*cl;
D = q*S*cd;

%Bank Angle
bank = sigma/180*pi;

%EOMs
%u(1) - Altitude
%u(2) - Velocity
%u(3) - Flight Path Angle
udot(1) = u(2) * sin(u(3));
udot(2) =  - D/m  - mu * sin(u(3))/(re + u(1))^2;
udot(3) = L*cos(bank)/m/u(2) + (((u(2)^2/(re + u(1))) -...
     (mu/(re + u(1))^2 ))*cos(u(3))/u(2));
udot = udot';
```

27.2 Mars Habitat Vehicle (MHV) Entry Feasibility Study

Author: Angela Long

27.2.1 Purpose

This section describes the method we use to evaluate the feasibility of landing the Mars Habitat Vehicle (MHV).

27.2.2 Significance

During the design process of the MHV, we proposed splitting the MHV into two separate vehicles. The rationale behind this decision was the mass of the MHV was too large. The contention was that a mass that large would need an unfeasibly large heat shield to land safely. Several difficulties arise when splitting the MHV into two vehicles (more launches, precision landing, etc.). Due to the unfavorable consequences of launching two MHVs, we re-evaluated the entry and descent trajectory problem.

27.2.3 Input/Output

We adopted the Ascent and Recovery Vehicle's (ARV) entry trajectory code, changing the planetary parameters and density model from Earth to Mars. The general inputs and output are exactly the same as for the ARV entry code. Because the purpose of this study is to evaluate the feasibility of landing a large mass, the MHV code only consists of two phases: ballute deployment and main parachute deployment.

For this simulation, we use a toroidal ballute with inner diameter of 90 m and outer diameter of 100 m. The entry velocity is 3.5 km/s and the flight path angle is -15˚. The initial mass of the vehicle is 50,000 kg.

Figure 27-8 shows the vehicle capturing in the Martian atmosphere and descending to the surface. Figure 27-9 is the velocity profile. Without any form of powered descent, the vehicle lands at a velocity of approximately 200 m/s. This velocity is too high for the vehicle to sustain impact. However, with powered descent, the vehicle can be slowed enough from this velocity to land safely. This simulation does not provide an answer to the problem of landing the MHV but rather proves that landing an amount of mass as large as the MHV is feasible. Therefore, the outcome of this study was the decision to land the MHV as one vehicle. Further investigation of this problem was conducted by Chris Statler and Jason Friel.

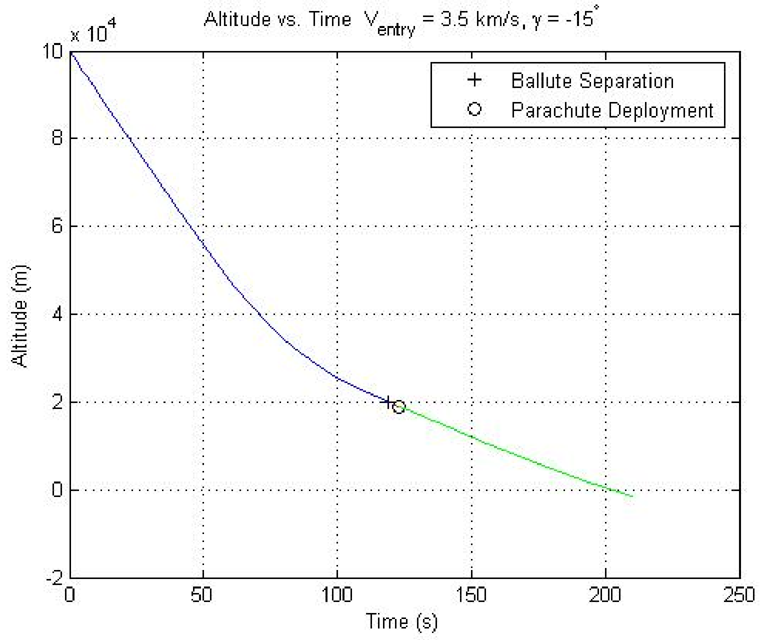


Figure 27-8 - Altitude vs. Time – MHV

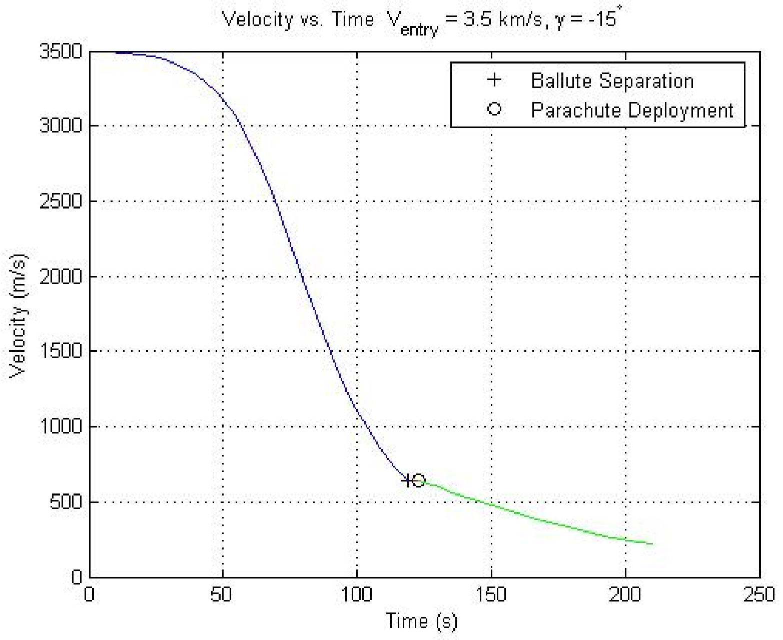


Figure 27-9 - Velocity vs. Time - MHV

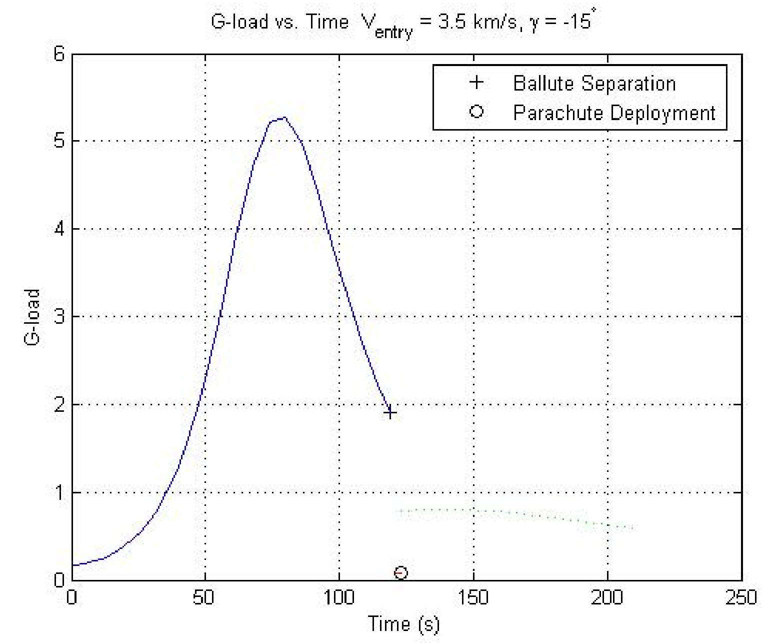


Figure 27-10 - G-loads vs. Time – MHV

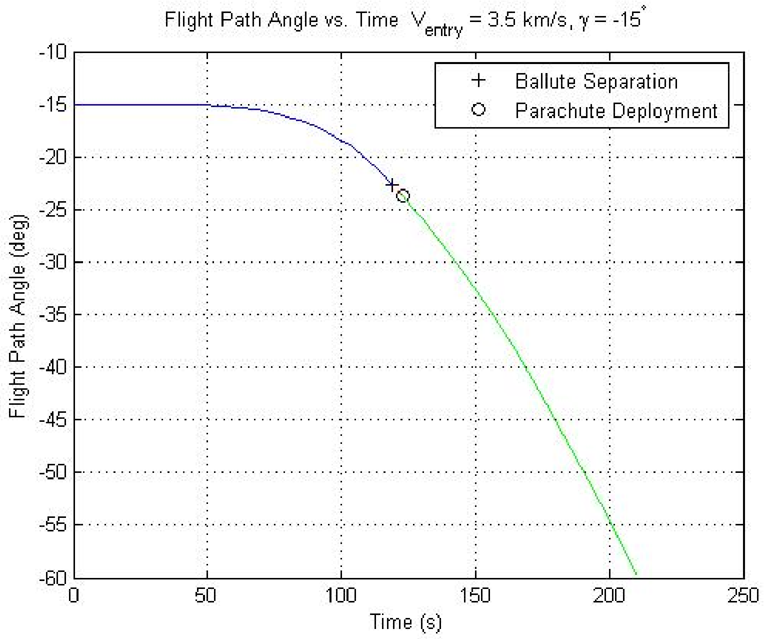


Figure 27-11 - Flight Path Angle vs. Time - MHV

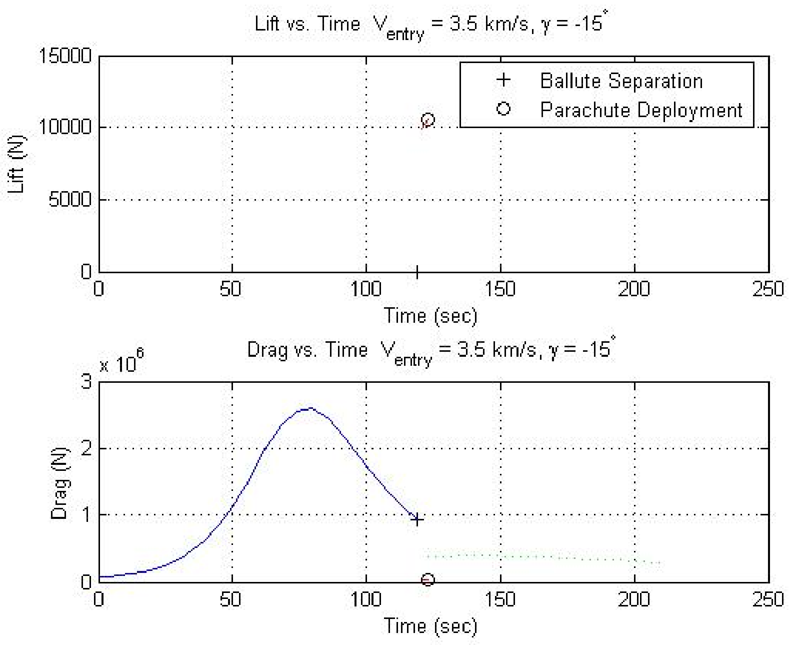


Figure 27-12 - Lfit and Drag vs. Time – MHV

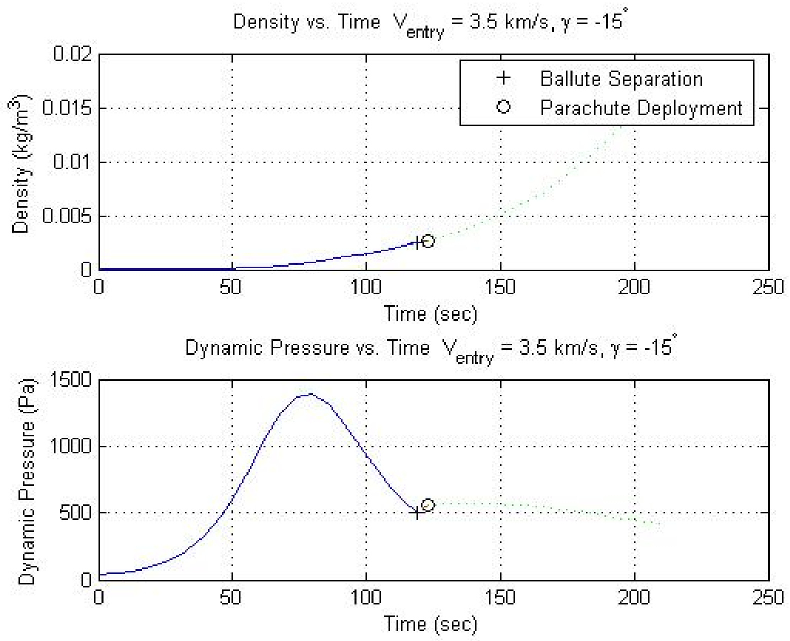


Figure 27-13 - Density and Dynamic Pressure vs. Time - MHV

27.2.4 MATLAB Code

Script file

```
%Angela Long
%AAE 450
%ARV Re-entry trajectory code
%Updated 3/7/05

close all
```

```
clear all
clc

t0= 0.0;
tf = 10000;

% Constants
re = 3397e3;     % m
he = 100e3;      % m
r = re + he;
mu = 42828*1000^3;     % m^3/s^2
%rho0 = 1.225; % kg/m^3
g = 9.81;

%Vr = 10;        % km/s

% Use v_inf to find entry velocity
% vinf = 3.5;
% a = mu/vinf^2;
%ve = sqrt(2*(mu/r + mu/2/a))

%Set entry velocity
ve = 3500   %m/s

%Set entry flight path angle
gamma = -15  %deg

%Torus Ballute
d1 = 100;   %m
d2 = 90;    %m
S1 = pi*d1^2/4; %m^2
S2 = pi*d2^2/4; %m^2
S = S1 - S2     %m^2

%Initial phase (ballute) parameters
m = 50000   %kg
L_D = 0
cd = 1.25
cl = L_D*cd
sigma = 0

% Set initial conditions for integration
% Altitude, Velocity, FPA, Heading Angle, Latitude, Heating
u0=[he ve gamma*pi/180 0 0];
dt=[t0,tf];
[t,x]=ode45(@marsEOMtest,dt,u0,[],m,S,cd,cl,sigma);
%gload = 1000*abs(diff(x(:,2))./diff(t)/9.81);

for i = 1:size(t,1)
[p(i), T(i), rho1(i,1), a(i), mu1(i)] = marsatmosphere2(x(i,1));
end
q1 = rho1.*(x(:,2).^2)./2;
L1 = q1.*S.*cl;
D1 = q1.*S.*cd;

Fa1 = sqrt((D1.*cos(x(:,3)) - L1.*sin(x(:,3))).^2 + (D1.*sin(x(:,3))+...
    L1.*cos(x(:,3))).^2);
a1 = Fa1./m;
gs1 = a1./g;
Gmax1 = max(gs1)

anj = 1;
m = size(x,1);
for anj = 1:m-1
if (x(anj,1) < 20000)
        anj = anj - 1
    break
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%           Ballute Separation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d = 10;
S = pi*d^2/4;
m = 50000;
L_D = .3;
cd = 0.8;
cl = L_D*cd;
sigma = 0;

u1= x(anj,:)
dt=[t0,tf];
[t1,x1]=ode45(@marsEOMtest,dt,u1,[],m,S,cd,cl,sigma);

for i = 1:size(t1,1)
[p(i), T(i), rho2(i,1), a(i), mu1(i)] = marsatmosphere2(x1(i,1));
end
q2 = rho2.*(x1(:,2).^2)./2;
L2 = q2.*S.*cl;
D2 = q2.*S.*cd;

Fa2 = sqrt((D2.*cos(x1(:,3)) - L2.*sin(x1(:,3))).^2 + (D2.*sin(x1(:,3))+...
    L2.*cos(x1(:,3))).^2);
a2 = Fa2./m;
gs2 = a2./g;
Gmax2 = max(gs2);

slps = 1;
m = size(x1,1);
for slps = 1:m-1
if (x1(slps,1) < 6000) | (q2(slps,1) < 700)
    slps = slps + 1
    break
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parachute Deployment
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d = 25;
S = 2*pi*d^2/4;
m = 50000;
L_D = 0;
cd = 0.7;
cl = L_D*cd;
sigma = 0;

u2 = x1(slps,:)
dt=[t0,tf];
[t2,x2]=ode45(@marsEOMtest,dt,u2,[],m,S,cd,cl,sigma);

for i = 1:size(t2,1)
[p(i), T(i), rho3(i,1), a(i), mu1(i)] = marsatmosphere2(x2(i,1));
end
q3 = rho3.*(x2(:,2).^2)./2;
L3 = q3.*S.*cl;
D3 = q3.*S.*cd;

Fa3 = sqrt((D3.*cos(x2(:,3)) - L3.*sin(x2(:,3))).^2 + (D3.*sin(x2(:,3))+...
    L3.*cos(x2(:,3))).^2);
a3 = Fa3./m;
gs3 = a3./g;
Gmax3 = max(gs3);


k = 1;
while x2(k,1) > 0
    if x2(k,1) == x2(end,1)
      break
    else
```

```
            k = k + 1;
        end
    end

    %plot results



    figure(1)
    plot(t(1:anj),x(1:anj,1),'b')
    hold on
    grid on
    plot(t1(1:slps)+t(anj),x1(1:slps,1),'r--')
    plot(t2(1:k)+t1(slps)+t(anj),x2(1:k,1),'g')
    plot(t(anj),x(anj,1),'k+')
    plot(t1(slps)+t(anj),x1(slps,1),'ko')
    title(['Altitude vs. Time  V_entry= ' num2str(ve/1000)...
        ' km/s, \gamma = ' num2str(gamma) '^\circ'])
    xlabel('Time (s)')
    ylabel('Altitude (m)')


    figure(2)
    plot(t(1:anj),x(1:anj,2),'b')
    hold on
    grid on
    plot(t1(1:slps)+t(anj),x1(1:slps,2),'r--')
    plot(t2(1:k)+t1(slps)+t(anj),x2(1:k,2),'g')
    title(['Velocity vs. Time  V_entry= ' num2str(ve/1000)...
        ' km/s, \gamma = ' num2str(gamma) '^\circ'])
    xlabel('Time (s)')
    ylabel('Velocity (m/s)')

    figure(3)
    plot(t(1:anj),x(1:anj,3).*180./pi,'b')
    hold on
    grid on
    plot(t1(1:slps)+t(anj),x1(1:slps,3).*180./pi,'r--')
    plot(t2(1:k)+t1(slps)+t(anj),x2(1:k,3).*180./pi,'g')
    title(['Flight Path Angle vs. Time  V_entry= ' num2str(ve/1000)...
        ' km/s, \gamma = ' num2str(gamma) '^\circ'])
    xlabel('Time (s)')
    ylabel('Flight Path Angle (deg)')

    figure(4)
    plot(t(1:anj),gs1(1:anj),'b')
    hold on
    grid on
    plot(t1(1:slps)+t(anj),gs2(1:slps),'r--')
    plot(t2(1:k)+t1(slps)+t(anj),gs3(1:k),'g:')
    title(['G-load vs. Time  V_entry= ' num2str(ve/1000)...
        ' km/s, \gamma = ' num2str(gamma) '^\circ'])
    xlabel('Time (s)')
    ylabel('G-load')

    figure(5)
    subplot(2,1,1)
    plot(t(1:anj),rho1(1:anj),'b')
    hold on
    plot(t1(1:slps)+t(anj),rho2(1:slps),'r--')
    plot(t2(1:k)+t1(slps)+t(anj),rho3(1:k),'g:')
    title('Density vs. Time')
    xlabel('Time (sec)')
    ylabel('Density (kg/m^3)')

    subplot(2,1,2)
    plot(t(1:anj),q1(1:anj),'b')
    hold on
    plot(t1(1:slps)+t(anj),q2(1:slps),'r--')
    plot(t2(1:k)+t1(slps)+t(anj),q3(1:k),'g:')
    title('Dynamic Pressure vs. Time')
```

```
xlabel('Time (sec)')
ylabel('Dynamic Pressure (Pa)')

figure(6)
subplot(2,1,1)
plot(t(1:anj),L1(1:anj),'b')
hold on
plot(t1(1:slps)+t(anj),L2(1:slps),'r--')
plot(t2(1:k)+t1(slps)+t(anj),L3(1:k),'g:')
title('Lift vs. Time')
xlabel('Time (sec)')
ylabel('Lift (N)')

subplot(2,1,2)
plot(t(1:anj),D1(1:anj),'b')
hold on
plot(t1(1:slps)+t(anj),D2(1:slps),'r--')
plot(t2(1:k)+t1(slps)+t(anj),D3(1:k),'g:')
title('Drag vs. Time')
xlabel('Time (sec)')
ylabel('Drag (N)')
```

<u>ODE45 function file</u>

In the ODE45 function file, the state variables are assigned as such: 1 – altitude, 2 – velocity, 3- flight path angle.

```
function udot = marsEOMtest(t,u,m,S,cd,cl,sigma)
re = 3397e3;             % m
h_a = 100e3;             % m
mu = 42828*1000^3;       % km^3/s^2

%Mars Atmospheric Model
[p, T, rho, a, mu1] = marsatmosphere2(u(1));

%Aerodynamin forces
q = rho*(u(2)^2)/2;
L = q*S*cl;
D = q*S*cd;

%Bank Angle
bank = sigma/180*pi;

%EOMs
%u(1) - Altitude
%u(2) - Velocity
%u(3) - Flight Path Angle
udot(1) = u(2) * sin(u(3));
udot(2) =  - D/m  - mu * sin(u(3))/(re + u(1))^2;
udot(3) = L*cos(bank)/m/u(2) + (((u(2)^2/(re + u(1))) -...
     (mu/(re + u(1))^2 ))*cos(u(3))/u(2));
udot = udot';
```

Mars atmosphere model (provided by John Stalbaum)

```
% This function calculates atmospheric properties based upon a given
% height, of Mars.
% Written by John Stalbaum

function [p, T, r, a, mu] = marsatmosphere2(h)
% Convert altitude to meters.
h = h/0.3048;

% Calculate various atmospheric properties in English units.
```

```
if h < 22960 % Below a certain altitude
    T = -25.68 - .000548 * h;
    p = 14.62 * exp(-.00003 * h);
    r = p / [1149 * (T + 459.7)];
elseif h >= 22960 % Upper atmosphere
    T = -10.34 - .001217 * h;
    p = 14.62 * exp(-.00003 * h);
    r = p / [1149 * (T + 459.7)];
end


R = 192; % Metric R of Martian atmosphere
% Convert back to metric units.
r = 515.378818*r;
T = (460 + T)*0.5555;
p = p * 47.880259;
mu = real((.03170*T^(1.5))*(734.7/(T + 216))*10^-10);  %mu at altitude
gamma = 1.288;
a = sqrt(gamma*R*T);

% If something is fishy, assign these values as zero, representing no
% atmosphere present.
if h > 104e3/0.3048
    r = 0;
    T = 0;
    p = 0;
    mu = 0;
    a = 0;
end


% End of Function
```

28 Long Hafid

28.1 Mars Lander Vehicle Active Thermal Control System

Author: Hafid Long

Contributors: Scott Walker, Christopher Cunha, Matthew Verbeke, Ezdehar Husein

28.1.1 Description

As part of the process of selecting a suitable power source for the Mars Lander Vehicle (MLV), we perform a trade study to determine the power source that yields the largest mass savings for the active thermal control system (ATCS).

28.1.2 Analysis

The assumption for this trade study is that the choice of power source does not significantly affect the volume and power requirement of the ATCS. Thus, focusing on the minimization of the ATCS mass narrows the scope of the trade study. The possible options for the power source are batteries and fuel cells. For the trade study, the following information is used.

Operational life: 2.5 earth days

Table 3-7 MLV power requirements

Systems	Power (kW)
Communication	
SDST Transponder	0.02
Omni Antenna	0.10
Transmitting Dish	0.40
Dynamics And Controls	
Guidance System	1.00
Human Factor	
LiOH System/CO_2 Removal	0.01
Light	0.10
Total	1.63

Using the equations discussed in the ATCS section for the MLV, we are able to obtain the masses for each of the ATCS components. In the case when batteries are used on the MLV, the mass breakdown is given below.

Table 3-8 ATCS component mass when batteries are used

Components	Mass (kg)
Coldplates	19.53
Pumps with accumulator	7.81
Plumbing and valves	4.10
Instruments and controls	1.37
Fluids	1.37
Heat sink	281.30
Multi-Layer Insulation	118.75
Heaters	1.40
Total	435.63

With fuel cells, the following table gives the mass breakdown.

Table 3-9 ATCS component mass when fuel cells are used

Components	Mass (kg)
Coldplates	19.53
Pumps with accumulator	7.81
Plumbing and valves	4.10
Instruments and controls	1.37
Fluids	1.37
Heat sink	142.99
Multi-Layer Insulation	118.75
Heaters	1.40
Total	297.32

As highlighted in the previous two tables, the major difference between the two options is in the mass of the expendable heat sink. By using fuel cells, we are able to lower the mass of the expendable heat sink because we do not have to carry as much water on-board the vehicle. The fuel cells provide the water needed for the expendable heat sink, which the batteries do not offer. The mass of the expendable heat sink can be computed using the equation below:

$$m = \frac{QD}{h_{fg}} - \left(6.4033e-4\right)D$$

Equation 3-7

Where

m	: Mass of expendable heat sink (kg)
Q	: Required heat-rejection rate (kW)
D	: Operational life (sec)
h_{fg}	: Latent heat of vaporization of working fluid (kJ/kg)

Based on this trade study, we select fuel cells as the power source for the MLV.

28.2 Ascent Recovery Vehicle Active Thermal Control System

Author: Hafid Long

Contributor: Scott Walker, Jayleen Guttromson

28.2.1 Description

The sizing process for the active thermal control system (ATCS) on the Ascent Recovery Vehicle (ARV) is very similar to that on the Mars Lander Vehicle (MLV). Both vehicles share quite similar systems due to their short operational life.

28.2.2 Analysis

The ARV has most of the major components of the ATCS on the MLV. The ATCS section of the ARV in Chapter describes the differences in more detail. Sizing of the system is done based on the following:

Operational life: 30 hours

Table 3-10 ARV power requirements

Systems	Power (kW)
Communication	
S-Band Transponder	0.02
Omni Antenna	0.60
Dynamics And Controls	
Guidance System	1.00
Human Factor	
Crew	1.00
Light	0.10
Total	2.72

We employ the same equations as on the MLV to compute the mass, volume and power requirement of each ATCS components. These equations can be seen in the ATCS section for the MLV. With the help of Microsoft Excel, the results shown below are generated for the ARV using the given equations.

Table 3-11 ATCS component mass, volume and power requirement for the ARV

Components	Power (kW)	Volume (m^3)	Mass (kg)
Coldplates	-	0.08	32.59
Pumps with accumulator	0.06	0.05	13.04
Plumbing and valves	-	0.00	6.84
Instruments and controls	0.00	0.00	2.28
Fluids	-	0.00	2.28
Heat sink	-	0.23	234.65
Multi-Layer Insulation	-	0.57	172.37
Total	0.06	0.93	464.05

28.3 Mars Lander Vehicle Thermal Protection System

Author: Hafid Long

Contributor: Dr. Schneider, Robert Manning, Damon Landau, Jackie Jaron, John Stalbaum

28.3.1 Description

The design of the thermal protection system (TPS) for the Mars Lander Vehicle (MLV) evolved from being based on a ballistic entry to a lifting entry. This evolution and process of sizing the system at each phase involve different approaches and tools.

28.3.2 Analysis

We begin the design of the TPS based on a ballistic entry similar to that of the Viking Lander. This type of entry is a straight descent from a parking orbit around Mars to the Martian surface. The mode of decelerating the vehicle is accomplished mostly by the drag force on the vehicle. As a preliminary study to see how feasible this type of entry is for the MLV, we use the same type of aeroshell that the Viking Lander had and size it up to meet the approximated dimensions of the MLV. The reason for using the Viking Lander as the baseline for this study is because of the success it already has. The dimensions of the aeroshell for the MLV are as follows.

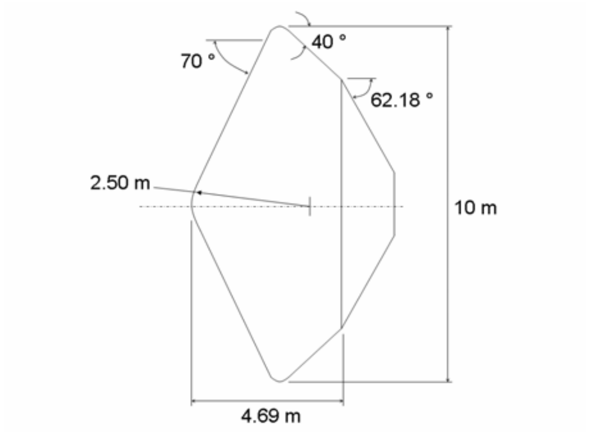


Figure 3-14 Aeroshell geometry for the MLV, adapted from the Viking Lander [1]

Based on the contours of the aeroshell, we assume that the aeroshell should have the same drag coefficient, as the Viking Lander, of 1.6. The ballistic coefficient for the MLV becomes the basis for the feasibility study in determining the mass that can possibly be carried by the vehicle. For this analysis, we use the Viking Lander's ballistic coefficient of 63.7 kg/m^2. Computation of the mass for the MLV is done below.

$$m = \beta S C_D$$

Equation 3-8

$$m = \left(63.7 kg / m^2\right)\left(78.5 m^2\right)\left(1.6\right) \cong 8000 kg$$

Where

m	: Mass of vehicle (kg)
β	: Ballistic coefficient of vehicle (kg/m^2)
S	: Projected area of vehicle (m^2)
C_D	: Drag coefficient of vehicle

From the above results, we clearly see that there is very little mass to accommodate the mission requirement for the MLV when a ballistic entry is used. Due to this reason, we adopt a lifting entry because we are able to benefit from the drag as well as lift to slow down the vehicle during the entry.

To gain an initial idea about how large the mass of the TPS is for the MLV, we take a simple approach of approximating the heating rate on the vehicle and calculate the mass from this information. This approach is described in the following flowchart.

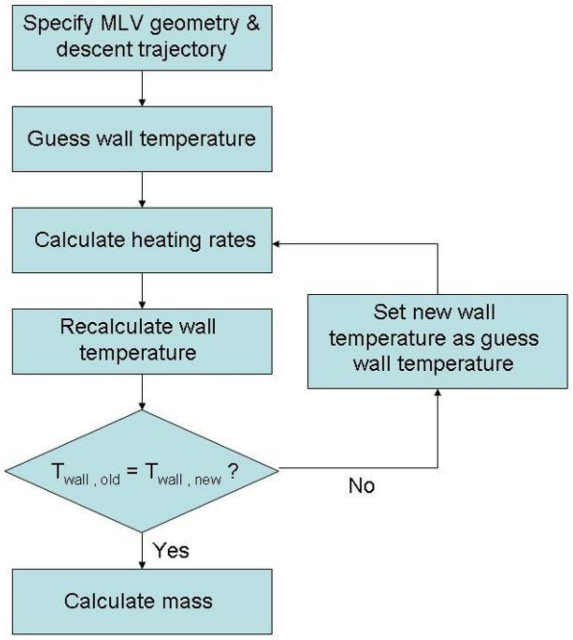

Specify MLV geometry & descent trajectory
Guess wall temperature
Calculate heating rates
Recalculate wall temperature
Set new wall temperature as guess wall temperature
$T_{wall,old} = T_{wall,new}$?
No
Yes
Calculate mass

Figure 3-15 First approach of approximating TPS mass for the MLV

Since the objective is to try and obtain an initial estimate of the TPS mass, using the velocity profile from a non-optimized trajectory of the MLV provided by John Stalbaum is reasonable. The figure below shows this velocity profile.

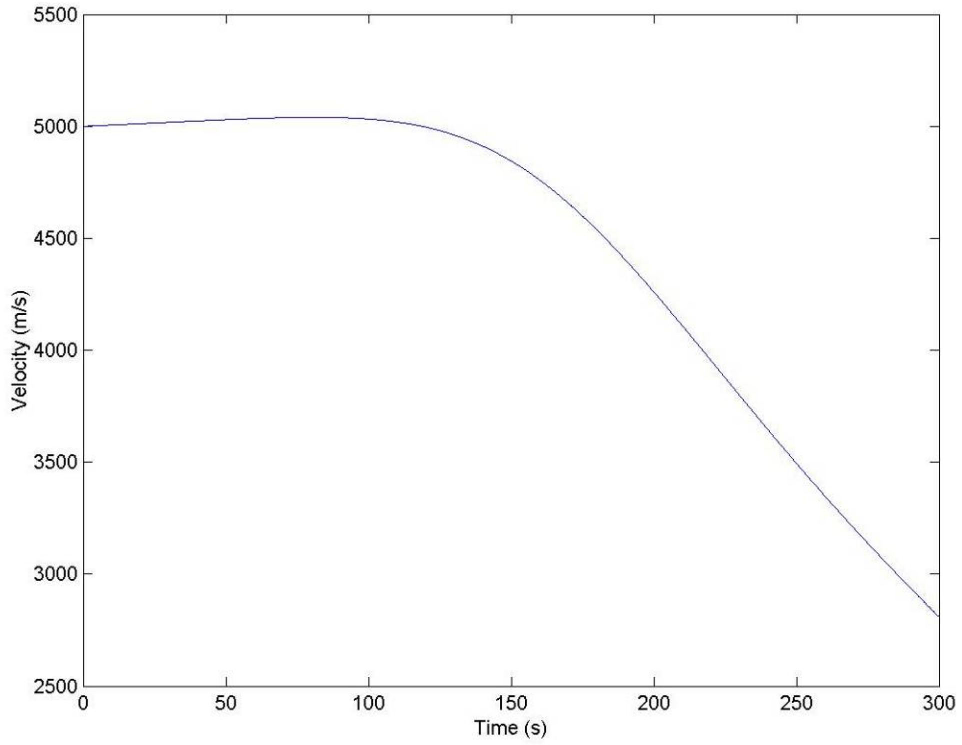


Figure 3-16 Velocity profile of non-optimized trajectory (John Stalbaum)

We also assume a lifting-body geometry and angle of attack for simplicity to allow us to perform the necessary mass calculations. The following is an illustration of the MLV simplified geometry.

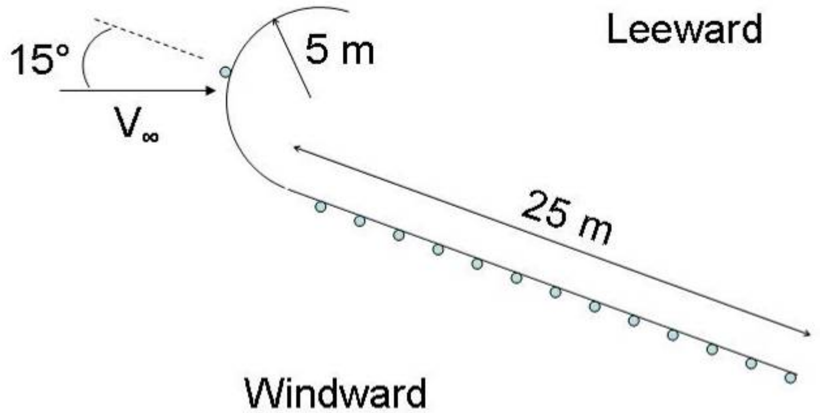


Figure 3-17 Simplified geometry of the MLV used for initial mass calculations

Since there is yet a defined dimension for the lifting body of the MLV at this point of the design process, we derive the above geometry from a very conservative assumption that the vehicle is as large as the payload fairing of the Earth's launch vehicle.

As for the assumptions, they are stated below:

- Fully turbulent flow over the vehicle
- Based on entry velocity, radiative heating is negligible [2]
- Mass of heat shield on leeward side is 25% of that on the windward side [3]
- TPS structure is 60% of total heat shield mass
- SLA-561V is the ablative material considered

Employing the stagnation point and Tauber's turbulent flow equations, which have been mentioned in the TPS section of the MLV, a MATLAB script found at the end of this section calculates the heating rates at each control points seen in the above figure. For the vehicle's windward side, which that has been represented as a flat plate, the heat shield is sized according to the control point with the largest heating rates. The point on the windward side that is subjected to the largest heating can be seen below in the plot.

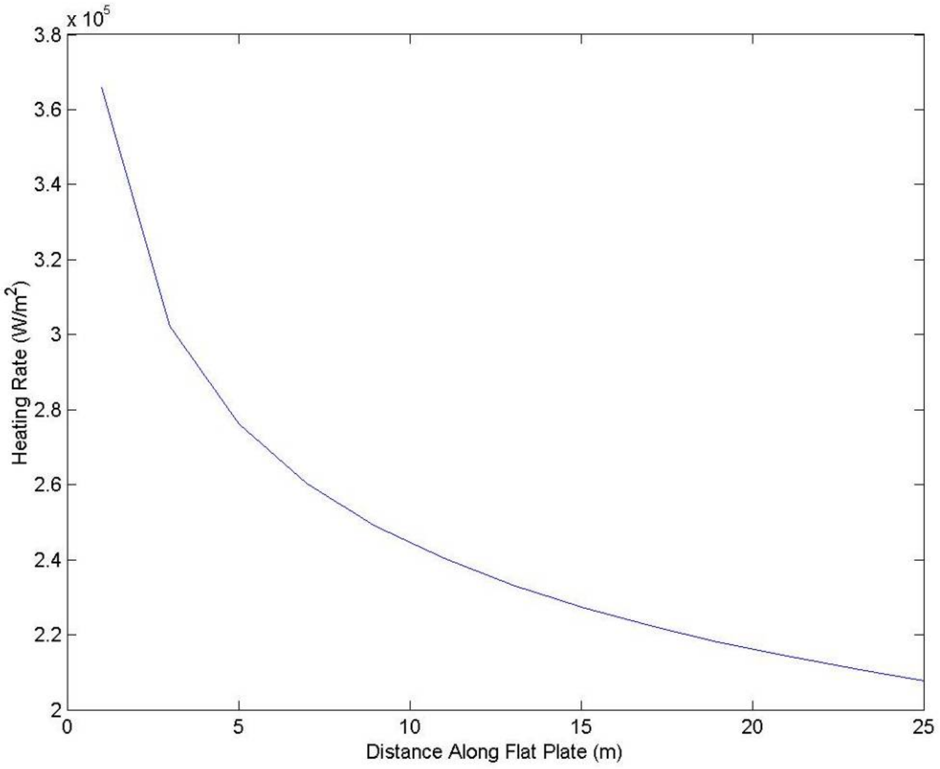


Figure 3-18 Heating rates along the windward side of the MLV

From the above figure, heating rate increases going from the back to the front of the vehicle. Based on this result, we size the heat shield for the windward side based on the heating rate at the front end of the flat plate. As a means of mass reduction, the sizing for the heat shield at the nose is done separately. Using the mentioned MATLAB script, the mass of the TPS is calculated and tabulated as follows.

Table 3-12 Mass breakdown of TPS components obtained through the simple approach

TPS Components	Mass (kg)
Nose Heat Shield	98.68
Windward Side Heat Shield	201.46
Leeward Side Heat Shield	50.37
Structure	210.31
Total	560.82

The initial mass numbers from the table concludes that the simple approach underestimates the size of the TPS for the given MLV's mass of about 40 metric tons. We expect a larger system mass based on the research done on TPS and engineering intuition. To get a more accurate mass that reflected the actual system, devising another approach is necessary.

A more accurate mass for the TPS can only come about with a defined geometry for the lifting-body of the MLV. After researching through some possible geometry for the lifting-body, we decided on the following for the MLV.

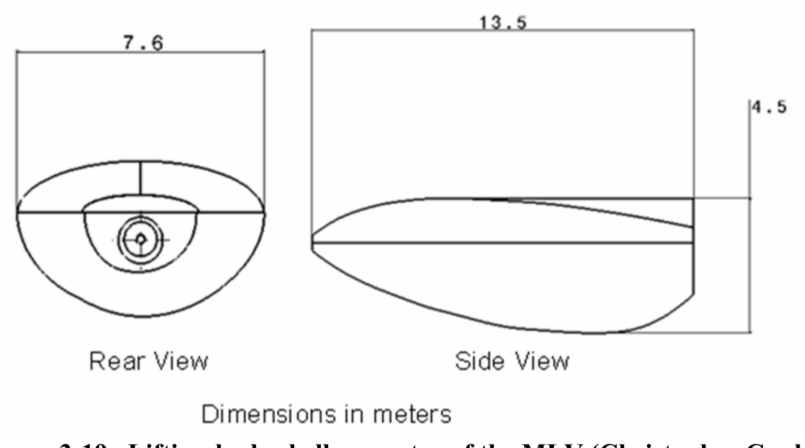


Figure 3-19 Lifting-body shell geometry of the MLV (Christopher Cunha)

The selection of the above geometry is on the merit of having enough volume to contain the MLV, which the lifting-body shell encloses during the Martian entry, and providing the necessary aerodynamic characteristics for a controlled, lifting entry. Mark Vahle discusses in more detail about the aerodynamic characteristics of the lifting-body shell in his appendix section.

A more defined lifting-body geometry as well as a better computation method allows us to yield more accurate results for the TPS. The computation method used at this stage of the design, which we discuss in the TPS section for the MLV, is more robust compared to the earlier approach. Professor Schneider suggests that this method gives the best approximated mass for a preliminary design of the TPS. The more robust design approach is well-illustrated below in the flowchart.

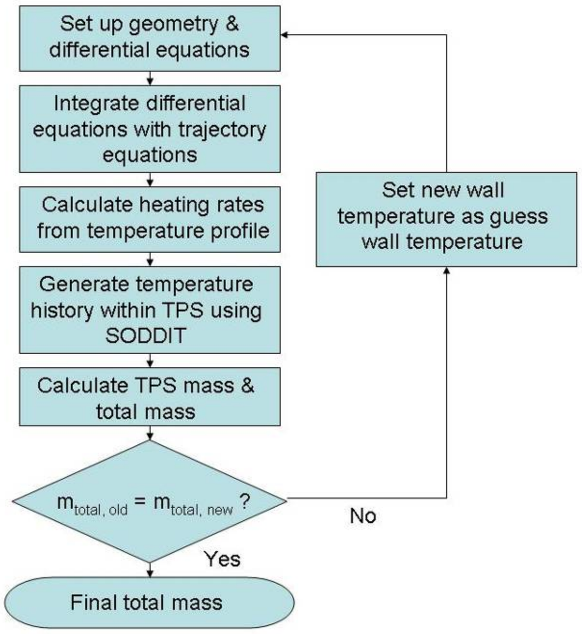

Set up geometry & differential equations
Integrate differential equations with trajectory equations
Calculate heating rates from temperature profile
Generate temperature history within TPS using SODDIT
Calculate TPS mass & total mass
$m_{total, old} = m_{total, new}$?
No
Set new wall temperature as guess wall temperature
Yes
Final total mass

Figure 3-20 Robust approach of approximating TPS mass for the MLV

We set up and integrate the differential equations, which have been mentioned earlier in the TPS section of the MLV for the rate of change of temperature, with the trajectory equations. This is done numerically in MATLAB. John Stalbaum helped out in the process of devising the MATLAB scripts to perform this task. The MATLAB scripts for the set up and integration of the differential equations as well as computation of the heating rates are attached in John Stalbaum's appendix section.

In order to analyze the effect of the computed heating rates in the Sandia one-dimensional direct and inverse thermal code (SODDIT), we need to generate an input file that contains the heating rates and other information pertaining to the ablative material. This process was automated through MATLAB by modifying the script originally written by Damon Landau with the help of Jackie Jaron. This modified script can be found at the end of this section.

As for the trajectory of the MLV, the MATLAB script employs the velocity of the optimized trajectory to calculate the heating rates on the MLV. Below is the plot of the velocity for the optimized trajectory.

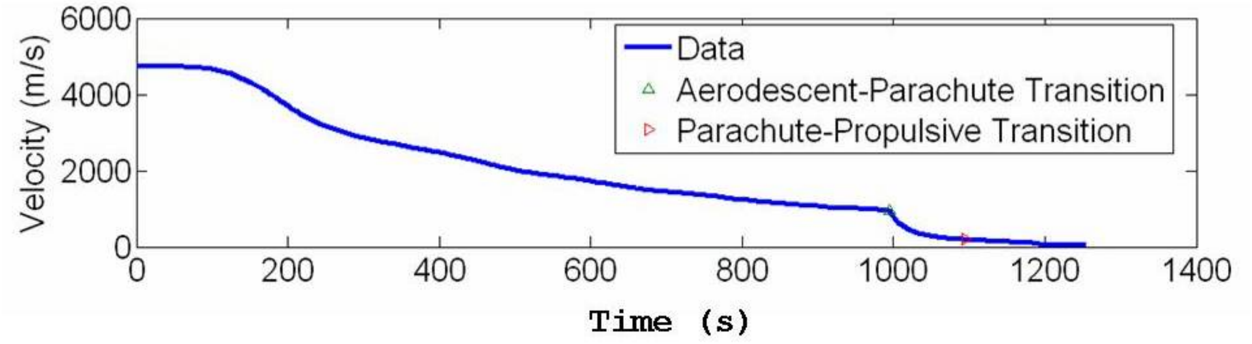


Figure 3-21 Velocity profile of optimized trajectory for the MLV

The TPS section for the MLV in Chapter shows the final breakdown of the TPS mass.

Reference

[1] Kirk, B. B., Intrieri, P. F., and Seiff, A. Aerodynamic Behavior of the Viking Entry Vehicle: Ground Test and Flight Results. AIAA, July 1978.

[2] Tauber, M. E., Sutton, K. Stagnation-Point Radiative Heating Relations For Earth and Mars Entries. AIAA, January 1991.

[3] Blattner, E., Jura, M., Heckler, G. W., Granum, G., Navarro, R. Mars Aero-Gravity Assist. Purdue University, 2003.

MATLAB Script for simple approach of estimating TPS mass

```
% AAE 450
% Hafid A. Long
%
% Code for weight estimation of heat shield for Mars Lander Vehicle.
%
%  Lifting body configuration is used for the Mars Lander Vehicle.
%   For approximation purposes, the part of vehicle that will be
%   exposed to a majority of the atmospheric heating will be considered
%   as a flat plate. A fully turbulent flow and a large angles of attack
%   will be assumed for conservative approximation of the heat transfer.
%   The high-performing and commonly used thermal protection material of
%   SLA-561 (ablators) will be used.
clear all;
close all;
clc;

% Define general constant
```

```
N = 0.8;        % Exponent for heat transfer correlation
M1 = 3.37;      % Exponent for heat transfer correlation [V <= 3962 m/s]
M2 = 3.7;       % Exponent for heat transfer correlation [V > 3962 m/s]
k1 = 3.35e-8;   % Constant for heat transfer correlation [V <= 3962 m/s]
k2 = 2.20e-9;   % Constant for heat transfer correlation [V > 3962 m/s]
SBC = 5.6696e-8;% Stefan-Boltzmann constant (W/(m^2-deg. K^-4))

% Define vehicle geometry
l = 25;         % Length of vehicle (m)
b = 8;          % Width of vehicle (m)
rn = 5;         % Nose radius of vehicle (m)

% Define flight condition
AOA = 15*pi/180;% Angle of attack (rad.)

% Define material constant (SLA-561)
d_m = 14.5/2.205/0.3043^3;      % Density (kg/m^3)
hv = 54.1e6;                    % Effective heat of ablation (J/kg)
E = 0.70;                       % Emissitivity

% Load external data
%  Trajectory
load DT.txt;

t = DT(:,1);    % Trajectory time (s)
d_a = DT(:,2);  % Atmospheric density along trajectory (kg/m^3)
V = DT(:,3);    % Trajectory velocity (m/s)

%   Plot trajectory velocity profile
figure; plot(t,V);
xlabel('Time (s)'); ylabel('Velocity (m/s)');
title('Trajectory Velocity Profile');
axis([0 300 2500 5500]);

%  Material properties (SLA-561)
load AP.txt;

temp = AP(:,1);
Cp = AP(:,2);

% Determine location of maximum heating rate
%  Max heating occurs when t = 183 s at V > 3962 m/s => index = 1831
x = [1:2:l]';

Tw_test = 0;

for n = 1:1:length(x)
    Tw_windward = 1060*9/5;
    while abs(Tw_test-Tw_windward) >= 5
        Tw_test = Tw_windward;
        Cpw = interp1(temp,Cp,Tw_windward,'spline');
        C = k2*cos(AOA)^2.08*sin(AOA)^1.6*x(n)^(-1/5)*(Tw_windward/556)^(-1/4)*...
            (1-1.11*(Cpw*Tw_windward/(0.5*V(1831)^2)));
        qdot = C*d_a(1831)^N*V(1831)^M2;
        x_qdotmax(n,1) = qdot*100^2;
        Tw_windward = (x_qdotmax(n,1)/(E*SBC))^(1/4);
    end
end

% Plot results
figure; plot(x,x_qdotmax);
xlabel('Distance Along Flat Plate (m)'); ylabel('Heating Rate (W/m^2)');
title('Maximum Heating Rate (Windward Side)');

% Define location of maximum heating on windward side of Mars Lander vehicle
x = 1;

% Compute stagnation point heating rate at nose of Mars Lander Vehicle
Tw_nose = 1060*9/5*ones(length(V),1);

Tw_test = 0;
```

```
for n = 1:1:length(V)
    while abs(Tw_test-Tw_nose(n)) >= 5
        Tw_test = Tw_nose(n);
        Cpw = interp1(temp,Cp,Tw_nose(n),'spline');
        qdot = 1.35e-8*sqrt(d_a(n)/rn)*V(n)^3.04*(1-(Cpw*Tw_nose(n)/...
            (0.5*V(n)^2)));
        qdot_nose(n,1) = qdot*100^2;
        Tw_nose(n,1) = (qdot_nose(n,1)/(E*SBC))^(1/4);
    end
end

% Plot results
figure; plot(t,qdot_nose);
xlabel('Time (s)'); ylabel('Heating Rate (W/m^2)');
title('Heating Rate Profile (Stagnation Point)');

figure; plot(t,Tw_nose);
xlabel('Time (s)'); ylabel('Wall Temperature (^oK)');
title('Wall Temperature Profile (Stagnation Point)');

% Compute maximum total heat load (J)
A = 4*pi*rn^2/3;        % Approximate nose area to be 1/3 of a sphere

Q_nose = trapz(t,qdot_nose)*A

% Compute mass of heat shield at nose of Mars Lander Vehicle
m_nose = Q_nose/hv

% Compute thickness of heat shield at nose of Mars Lander Vehicle
b_nose = Q_nose/(d_m*hv*A)

% Compute maximum heating rate on windward side of Mars Lander Vehicle
Tw_maxqdot = 1060*9/5*ones(length(V),1);

Tw_test = 0;

for n = 1:1:length(V)
    while abs(Tw_test-Tw_maxqdot(n)) >= 5
        Tw_test = Tw_maxqdot(n);
        Cpw = interp1(temp,Cp,Tw_maxqdot(n),'spline');
        if V(n) <= 3962
            C21 = k1*cos(AOA)^1.78*sin(AOA)^1.6*x^(-1/5)*(Tw_maxqdot(n)/556)^(-1/4)*...
                (1-1.11*(Cpw*Tw_maxqdot(n)/(0.5*V(n)^2)));
            qdot = C21*d_a(n)^N*V(n)^M1;
        else
            C22 = k2*cos(AOA)^2.08*sin(AOA)^1.6*x^(-1/5)*(Tw_maxqdot(n)/556)^(-1/4)*...
                (1-1.11*(Cpw*Tw_maxqdot(n)/(0.5*V(n)^2)));
            qdot = C22*d_a(n)^N*V(n)^M2;
        end
        qdot_max(n,1) = qdot*100^2;
        Tw_maxqdot(n,1) = (qdot_max(n,1)/(E*SBC))^(1/4);
    end
end

% Plot results
figure; plot(t,qdot_max);
xlabel('Time (s)'); ylabel('Heating Rate (W/m^2)');
title('Heating Rate Profile (At Point Of Largest Heating On Windward Side)');

figure; plot(t,Tw_maxqdot);
xlabel('Time (s)'); ylabel('Wall Temperature (^oK)');
title('Wall Temperature Profile (At Point Of Largest Heating On Windward Side)');

% Compute maximum total heat load (J)
Q_max = trapz(t,qdot_max)*l*b

% Compute mass of heat shield on windward side of Mars Lander Vehicle
m_windward = Q_max/hv

% Compute thickness of heat shield on windward side of Mars Lander Vehicle
```

```
b_windward = Q_max/(d_m*hv*l*b)

% Compute mass of heat shield on leeward side of Mars Lander Vehicle
m_leeward = 0.25*m_windward     % Approximate mass to be 25% of windward side

% Compute mass of heat shield structure
mtotal_HS = m_nose+m_windward+m_leeward;

m_structure = 0.6*mtotal_HS     % Approximate mass to be 60% of total heat shield mass

% Compute total mass of thermal protection system
mtotal = mtotal_HS+m_structure
```

MATLAB Script for generating SODDIT input file

```
function [density, masses, Tmaxwall,heatout] = soddit_unix(mat,thick,filename,hindex)
% Created by Damon Landau (originally soddit.m)
% Modified by Hafid A. Long & Jackie Jaron
%
% Function is self-automated. It creates an input file for SODDIT, runs SODDIT and
%  plots the output data from SODDIT. Modification had to be made to the original
%  code in order for it to run properly on the newer version of MATLAB. A sample of
%  how to run the function is shown below
%
%  soddit_unix({'Teflon','Carbon Insulator','Composite'},[1e-3 4e-2 2e-3],'test',1)

% Define location of SODDIT program and input file
soddit_path = '/home/bfrc/a/aae450s/soddit/';
current_directory = '/home/bfrc/a/jaron/SC/';

% Problem constants
n = 15;                         % Number of elements per material
timemax = 3000;                 % Maximum time of simulation
n_output = 500;                 % Number of output time values

% Geometry
nmat = length(thick);           % Number of layers modelled
node = nmat*n+1;                % Total number of "nodes"
chunk = sum(thick);             % Total thickness
dx = thick/n;                   % Thickness of element per material

% Form vector of temperature distribution across thickness
tinit = [200 290];                      % 0 & thickness initial temp, linearized across length
delT = thick/chunk*(tinit(1)-tinit(2)); % Change in temperature across each individual layer
Tn = tinit(1)-cumsum(delT);             % Temperature at the bottom of each layer
Tn = [tinit(1) Tn(1:nmat-1)];           % Temperature at the top of each layer
Tdist = tinit(1);                       % Initialize Temperature distribution vector
for mc = 1:nmat
    Tdist_mc = Tn(mc)-delT(mc)/n*[1:n]; % Find temperature at each node for a given layer
    Tdist = [Tdist Tdist_mc];           % Append to the vector for use in Block six
end

disp('Creating input file');

% Create input file for SODDIT
inputfname = [filename '.inp'];
infile = fopen(inputfname,'w');

%  Set flags for SODDIT
prt=1;                          % 1 to write to print file, 0 not to

%  If top layer is an ablator, specify the proper flag, and create FRACT for use in block 4
if findstr('Ablator',mat{1});
    abl = 3;
else;
    abl = 0;
end
if abl == 3;
    fract = [.001   0.05];
```

```
else;
    fract = [];
end

flags=zeros(1,80);              % Set all other flags are zero
flags(4)=abl;
flags(5)=prt;
flags(16)=1;
flags(2)=3;

%  Get heating data from text file generated by qwg.m
heat = load([filename '.qw']);                                                      % Load heating
data from file
heat = [heat(:,1) zeros(size(heat,1),1) heat(:,1+hindex) zeros(size(heat,1),1)];    % Format for use
in SODDIT
heatout(1) = max(heat(:,3));                                                        % Find maximum
heating rate
heatout(2) = trapz(heat(:,1),heat(:,3));                                            % Find total heat
load

%  Condense heating data by removing long series of zeros (SODDIT will interpolate)
i = 2;

while i< length(heat)-1
    % If the heating value is zero for three successive points then remove the middle one.
    % This leaves the beginning and end of long stretches of zeros.
    if (heat(i-1,3) == 0 & heat(i,3) == 0 & heat(i+1,3) == 0)
        heat(i,:)=[];
        i=i-1;
    end
    i = i+1;
end

if heat(end,1)>timemax
    timemax = heat(end,1);                              % Run longer than user-specified time values
else
    % Add q = 0 at end to allow for equilibrium
    heat = [heat;heat(end,1)+1 0 0 0;timemax 0 0 0];    % Append
end

mp = fopen('matprop');      % Open material property file

%  Write to input file

%   Block 1
fprintf(infile,'%s \n',...
    '**block 1 control flags',...
    'c23456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789',...
    sprintf('%1i',flags));

%   Block 2

%    Write material names and thicknesses
fprintf(infile,'  %s, ',mat{:});fprintf(infile,'\n');
fprintf(infile,'  %f, ',thick);
fprintf(infile,'\n  Heating with known time dependant flux\n');

%   Block 3
fprintf(infile,'%s \n',...
    '**block 3 print times and print intervals, maximum of 20 and 19 resp',...
    ['  0.0       '  num2str(timemax) ],...
    ['  ' num2str(ceil(timemax/n_output))] );           % Round off to infinity & keep number of time
steps to under 300

%   Block 4
fprintf(infile,'%s \n',...
    '**block 4 general problem constants',...
    '  dtmin      dtmax    theta    dtempm    sigma    radius     expn   fract1 fract2',...
    ['  0.000001   0.5      1.0        10.  5.66961e-8  0.0       0.0    ' num2str(fract)]);

%   Block 5
```

```
fprintf(infile,'%s\n','**block 5**********material property data ');
for matc = 1:length(mat)
    fprintf(infile,'%s \n','**rho    dhf   treff   qstar   tabl     matnam (start past col 41)');

    % Find material
    findline(mat{matc},mp);                             % See function below
    l1 = str2num(fgetl(mp));                            % First line of property for given material
    rho(matc) = l1(1);                                  % Density
    Tm(matc) = l1(2);                                   % Melting temperature
    numcheck = str2num(fgetl(mp));                      % Get number of columns
    data = fscanf(mp,'%f',[length(numcheck) inf]);
    data = data';
    data = [numcheck;data];                             % Total data set
    % If ablation, then write ablation data
    % Also, if no emmisivity data is given for the top layer, assume its 0.8 to run SODDIT (get real
data later)
    if matc == 1
        if abl == 3;
            qabl = [l1(3) l1(2)];
        else
            qabl = [];
        end
        if size(data,2) < 4
            data(:,4)=0.8*ones(size(data,1),1)
        end
        % Set emmisivity to zero of lower layers
    else
        data(:,4) = zeros(size(data,1),1);
        qabl = [];
    end
    % Write material property to input file
    fprintf(infile,'%s \n',...
        ['  ' num2str(l1(1)) ' 0.0    298. ' sprintf('%.4E ',qabl) '                         '
mat{matc}],...
        '**temp            cp          cond    emit  absrp');
    fprintf(infile,'  %8.2f    %8.2f     %8.3f%8.2f     1\n',data(1:end-1,:)');
    fprintf(infile,'-1%8.2f    %8.2f     %8.3f%8.2f     1\n',data(end,:)');
    frewind(mp);                                        % Go back to beginning to find next material
end

fprintf(infile,'%s \n','/end of block 5 material property tables');

%   Block 6
fprintf(infile,'%s \n',...
    '**block 6 user defined node generation data',...
    '**initial temperature',['**  ' num2str(tinit)],...
    '**(node no) (mat) (dxi) (Temperature) (Area) (Vol) )(NGAP)');
mat_dist = ceil([1:node]/n);
mat_dist(end) = mat_dist(end)-1;    % Find which material corresponds to each node
thick_dist = thick(mat_dist)/n;     % Thickness of each element
A_dist = ones(n*nmat+1,1)';         % Relative area of each element (area into the page)
V_dist = thick_dist.*A_dist;        % Relative volume of each element
nodedat = [ [1:n*nmat+1]' mat_dist' thick_dist' Tdist' A_dist' V_dist'  zeros(n*nmat+1,1)];     % Make
matrix
fprintf(infile,'%i %4i %10.8f %10.2f %10.8f %10.8f %5i \n',nodedat');                          %
Format & write
fprintf(infile,'/end of Block 6 Node generation\n');

%   Block 7

%    Get heating data
fprintf(infile,'%s \n',...
    '**block 7 front face boundary condition data',...
    '**ibcty1 ifmt1 trad1',...
    ['  '  num2str([1 0 20])],...
    '**time   rec enthapy rad input   C sub h           P',...
    '** sec    J/Kg         W/m^2     Kg/m^2-s         atm');

%    Write qdot data to input file
fprintf(infile,'  %i                    %i              %.4E          %i            \n',heat');
fprintf(infile,'%s \n',...
    '/end of block 7 aero heating boundary conditions',...
    '**block 8 back face boundary condition data',...
    '**ibctyn ifmtn tradn (perfectly insulated back face boundary condition)',num2str(0));

%   Block 9 - pseudo atmosphere data (not used but necessary to fool SODDIT, hahaha)
fprintf(infile,'%s\n',...
    ' 1.0E-03          1.0E-9   1000.               -1720.44         MARS NO ABLATE ',...
    ' 1.0E-03          1.0E-9    300.               -1929.71         MARS NO ABLATE 1 ',...
    ' 1.0E+01          1.0E-9   1000.               -1720.44         MARS NO ABLATE ',...
    ' 1.0E+01          1.0E-9    300.               -1922.91         MARS NO ABLATE 3 ');

disp(['Created ' inputfname]);

% Find index of .inp extension
ext = findstr('.inp',inputfname);

% Put input filename into a file for use in SODDIT
fprintf(fopen('inf','w'),[current_directory inputfname '\n']);

% Delete old output files, and run SODDIT
fprintf(fopen('s','w'),['rm ' inputfname(1:ext-1) '_out.txt ' inputfname(1:ext-1) '_plt.txt\n'
soddit_path 'soddit.exe < inf\n']);
fclose('all');

% Tell user to run soddit and wait until finished
disp('Type "s" in a terminal to run SODDIT.  Hit any button once completed.')
pause

% Run UNIX commands stored in file
!chmod u+x s
!s
!rm inf

% Open plot file based on input filename
datfile = fopen([inputfname(1:ext-1) '_plt.txt']);
disp(['Opened ' filename '_plt.txt']);

% Find temperature data
%  Find heading of temperature histories
findline('**t/c temperatures**',datfile);           % See function below
fgetl(datfile);                                     % Skip 1 line to get to start of data
data = fscanf(datfile,'%f',[length(mat)+2 inf]);    % Read in matrix
data = data';

% Place data into appropriate vectors and find times of maximum temps
t = data(:,1);                                      % Time
Twall = data(:,2:length(mat)+2);                    % Temperature
T = Twall;                                          % Don't want to screw things up
Tmaxwall = [max(Twall) data(end,length(mat)+2)];    % Max and final temperatures

% Find times at maximum surface and inner wall temps
for Tc = 1:length(mat)+1
    iTmax(Tc) = min(find(Twall(:,Tc) == Tmaxwall(Tc)));
end

% Time vector, contains zero for use in following for loop
tmaxwall = [0 t(iTmax)' t(end)];

figure(1),clf

for figc = 5                            % Length(Tmaxwall)
    tindex = [1:length(Tmaxwall)+1];    % Indices of timevalues in tmaxwall that you want to plot.

    % tmaxwall is [inital time, time of max temp at top of each layer, t maxtemp of inner wall, final
time]
    pc = tindex(figc);
    a = sprintf('time = %12.5E',tmaxwall(pc));

    disp(a);
    findline(a,datfile);                            % Find heading of each desired time value
    fgetl(datfile); fgetl(datfile);                 % Skip 2 lines
```

```
        data = fscanf(datfile,'%f',[3 inf]);            % Read in temperature profile
        data = data';
        xp(:,1) = data(:,2); Tp(:,1) = data(:,3);       % Assign to proper variables

        % Plot data
        figure(1); subplot(1,1,1); plot(xp,Tp); grid on; hold on;
        xlines = cumsum(thick);
        ylines = get(gca,'Ylim');
        plot([xlines;xlines],ylines','k')
        set(gca,'Xlim',[0 xlines(end)]);

        if pc == 1
            dmat = '(Initial Distribution)';
            dtemp = num2str(Tmaxwall(1));
        elseif pc == nmat+2
            dmat = ['(Final Distribution)'];
            dtemp = num2str(Tmaxwall(pc-1));
        else
            dmat = [' (T_m_a_x for ' mat{pc-1} ')'];
            dtemp=num2str(Tmaxwall(pc-1));
        end
        title(['t = ' sprintf('%i',tmaxwall(pc)) 'sec,' dmat])
        xlabel('Depth, m');ylabel('Temperature, K')
    end

    disp('Read temperature data')

    % Display max temperature
    figure(2);
    clf;

    for f2c = 1:nmat+1
        mindex = [1:nmat+1];        % Columns of temperature history output to plot

        %Columns are:  top of each layer, innerwall
        matc = mindex(f2c);
        figure(2); subplot(1,1,1); plot(t,T(:,matc));

        if f2c == nmat+1
            dmat = 'Inner wall';
            dtemp = '~310';
        else
            dmat = [mat{matc}];
            dtemp = num2str(Tm(matc));
        end

        title([dmat ', T_m_a_x = ' sprintf('%5.1f',Tmaxwall(matc)) 'K,T_a_l_l_o_w =  ' dtemp 'K'])
        xlabel('Time, sec')
        ylabel('Temperature, K')
        grid on
        hold on
        pause
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function findline(str,fid)
    % Used to find a given string in a file
    %  str is desired string, fid is file identifier
    b1 = 1;

    while b1 == 1
        b1 = isempty(findstr(fgetl(fid),str));
    end

    return
```

29 McCurdy, Justin

29.1 Communications Satellites System Overview

Author: Justin McCurdy

Contributors: Jeri Metzger, Brendan Eash, Zade Shaw

Developing the CSS was a long and iterative process in part due to the ever-changing requirements, but also because the more we learned about the necessary communications network needed for this mission to succeed, the more we realized we had to change our then-current configuration.

We began with analyzing the Deep Space Network (DSN), which is a series of high gain, exceedingly large parabolic dishes that keep communications with satellites and probes. However, upon learning that there are already approximately 28 spacecraft and satellites already using the DSN, and that the price for employing their dishes for our intricate communications requirements would be enormous, we abandoned the DSN idea. In Equation 29-1 seen below, the basic equation for DSN pricing is listed.

Equation 29-1: Aperture Fee for DSN Use

$$A_F = R_B\left[A_W\left(.9+\frac{F_C}{10}\right)\right]$$

where A_F is the aperture fee for DSN's services, R_B is the hourly rate, A_W is the aperture weighting (depending on which DSN dish is used), and F_C is the number of station contacts per calendar week [1]. The approximate total cost is \$308 million per mission, assuming only one contact is made with the crew per hour. This number will rise rapidly as the number of communication contacts increases.

Briefly, we toyed with the thought of building our own communications network, and selling the time we would not be using it to existing space programs, but we ended up deciding to use a large-aperture university parabolic dish to meet our needs. See Figure 29-1 below for locations of the three massive parabolic dishes that comprise the Deep Space Network.

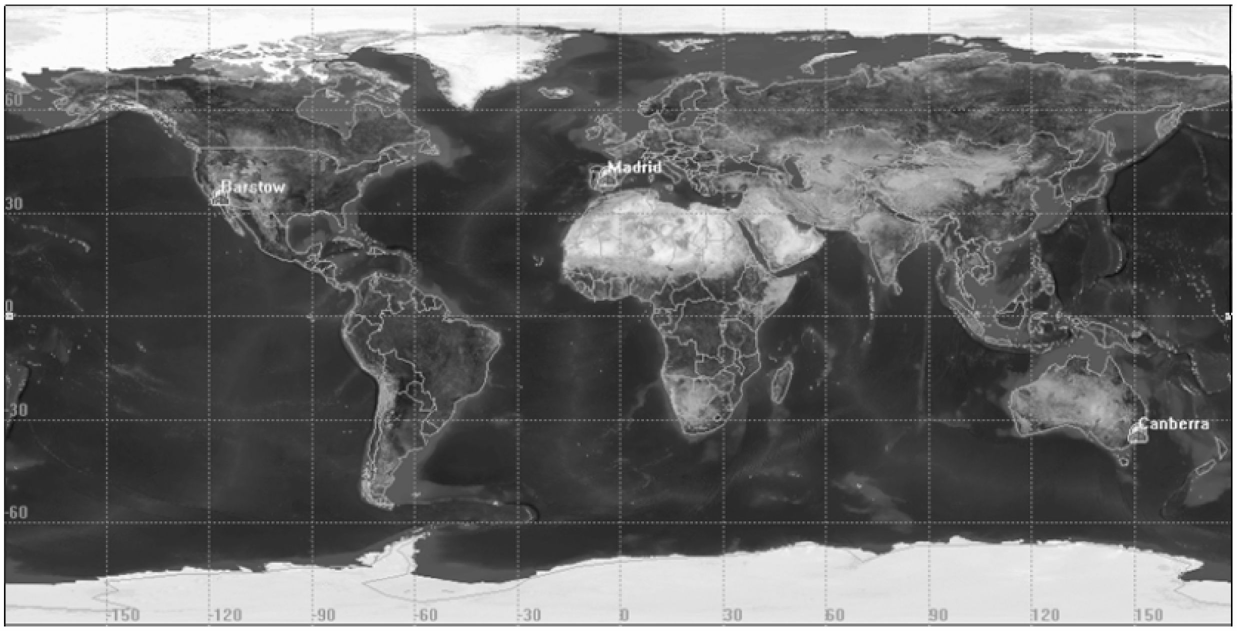


Figure 29-1: Location of DSN Dishes

29.2 The MORS System

Author: Justin McCurdy

Contributors: Jeri Metzger, Brendan Eash, Zade Shaw

Upon abandoning DSN, we no longer had the advantage of using two 34m and one 70m parabolic dishes here on Earth—this meant we had to have a stronger communications system on the vehicles. Thus, MORS was born, and the basic ideology and framework that it was based upon never strayed far from that of the MORS satellite depicted in Figure 29-2.

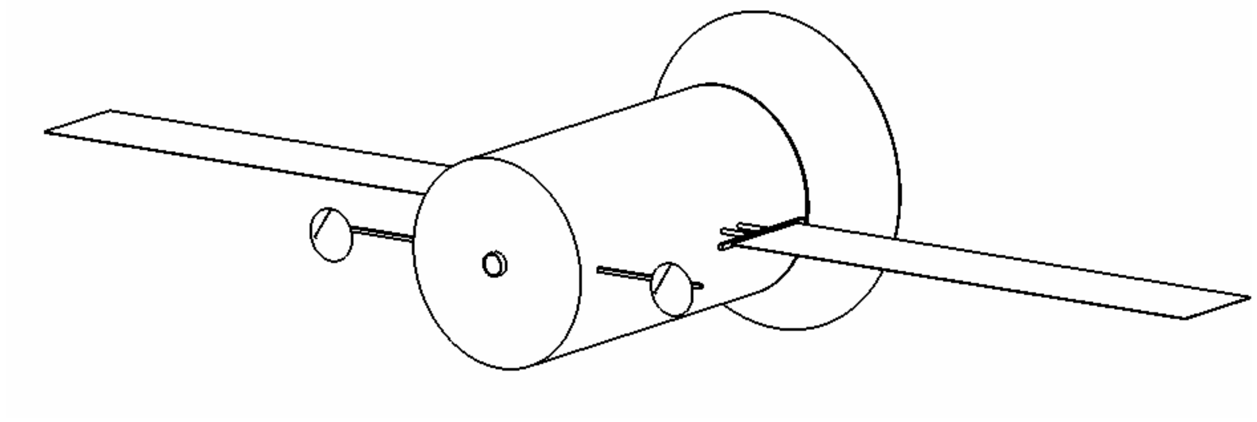


Figure 29-2: Deployed MORS

We knew that the crew had to have constant communications with the Earth, and the only way for this to happen is to provide a relay satellite system for when the astronauts are on the far side of the planet and out of communications line-of-sight with Earth. Upon analyzing the orbital coverage, we decided upon two MORS satellites to cover this daily blackout period, and to provide a degree of redundancy. With two MORS and a fully-equipped CTV (which has two 6m dishes of its own) able to relay transmissions, there are three functioning communications satellites in Martian orbit. Although the MORS orbits began as elliptical, we decided upon using circular "Mars-synchronous" orbits for the two satellites, and an elliptical CTV orbit. The two MORS and the CTV can be seen orbiting Mars in Figure 29-3 below.

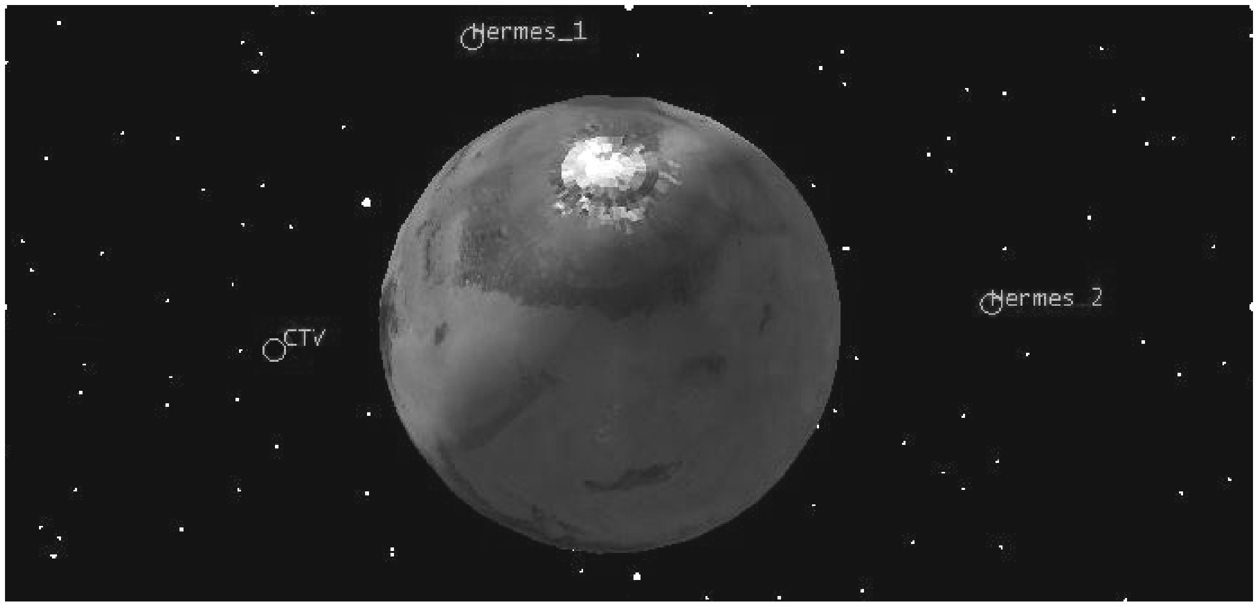


Figure 29-3: Two MORS and the CTV orbiting Mars

29.3 Parabolic Dish Geometrical Parameters

Author: Justin McCurdy

Contributors: Jeri Metzger, Brendan Eash, Zade Shaw

The dish itself then had to be analyzed in order to maximize the performance, gain, and efficiency, and minimize mass. Parabolic reflectors made of mesh present impressive mass savings over solid dishes, however, that is their only advantage. There is a "0.8 factor," which essentially states that the performance of a mesh dish is generally 0.8 times that of a solid dish of the same size (in terms of gain and efficiency). We cannot afford this performance drop if we are to meet our communications requirements. With the high frequency radio bands we are using to transmit data between Mars and Earth, the X-band and Ka-band, the mesh would have to be woven impossibly fine in order to avoid suffering massive performance drops, and this in turn would cause the price to skyrocket. Not only would the mesh dish be more vulnerable to micrometeorite damage than a solid dish while in space, but parabolic reflectors made of mesh material are much more susceptible to construction and fabrication error. Thus, we decided to use solid parabolic reflectors for the mission, especially since the mass of the dish isn't too consequential anyways.

All of the parabolic dishes employed an f/D ratio of 0.55, which translates to a shallow reflector, for the following reasons:

- Deep dishes are hard to illuminate efficiently
- Deeper dishes also are much more sensitive to focal length errors
- Shallow dishes achieve more gain

In below Table 29-1, all of the basic dimensions are given for each dish size used during this mission.

Table 29-1: Dish Dimensions

Dish Diameter	Equation	Focal Distance	Depth	Collector Arm Length
6m	$y(x) = 0.758x^2$	3.30m	0.682m	3.983m
1m	$y(x) = 0.455x^2$	0.55m	0.114m	0.664m
0.5m	$y(x) = 0.909x^2$	0.28m	0.057m	0.332m

The final basic antenna information can all be seen below, in Table 29-2. Everything seen here was optimized for this particular mission.

Table 29-2: Basic Dish Antenna Description

Element	Description
Reflecting Surface	Solid for increased shape accuracy and performance
Backside Stiffening	Radial ribs
Deployment Mechanism	Reverse-umbrella retracting surface
Shape Providing Element	Solid reflector, small radial ribs
Feed Type	Center-fed (prime focus) for optimal gain

Code for 6m dish Matlab representation:

```
% Justin McCurdy
% 6m Parabolic Dish

clc
x = linspace(-3,3,1000);
x2 = linspace(0,3.3,1000);
y = .0758.*(x.^2);
t = .6818;
f = 0;
plot(x,y)
AXIS([-3.0 3.0 0 3.5])
hold on
plot(y,t,'b')
hold on
plot(x,t,'b')
plot(f,x2,'b')
plot(0,3.3,'o')
hold on
t3 = linspace(-3,0,1000);
d1 = 3.3 + .8727.*t3;
plot(t3,d1,'r--')
t4 = linspace(0,3,1000);
d2 = 3.3 - .8727.*t4;
plot(t4,d2,'r--')
title('2-D Dish Configuration, f/D = 0.55')
```

Using this code, a graphical representation of the 6m dish is produced, and can be seen below in Figure 29-4.

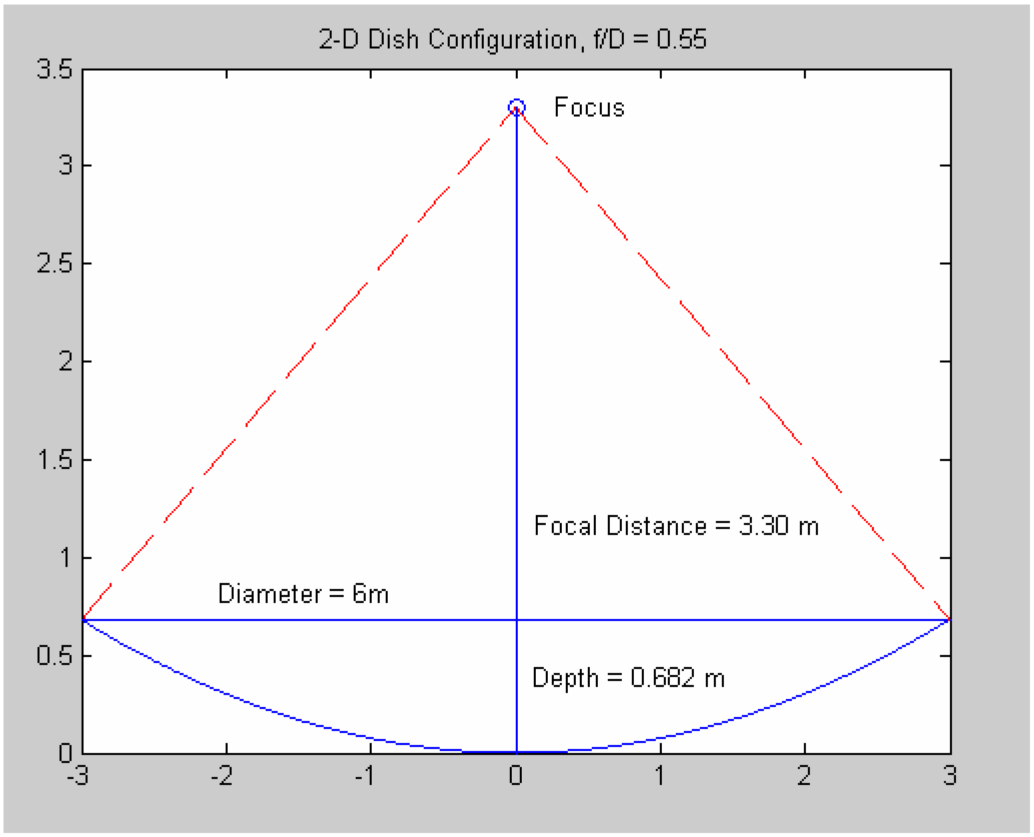


Figure 29-4: Matlab 6m Dish Representation

29.4 CSS Models and Projections

Author: Justin McCurdy

With the addition of the HRS to the Communications Satellite System, everything was complete. The isometric views of both the MORS and the HRS can be seen below.

29.4.1 Isometric CSS Drawings

Figure 29-5 shows the isometric views for the stowed version of the MORS.

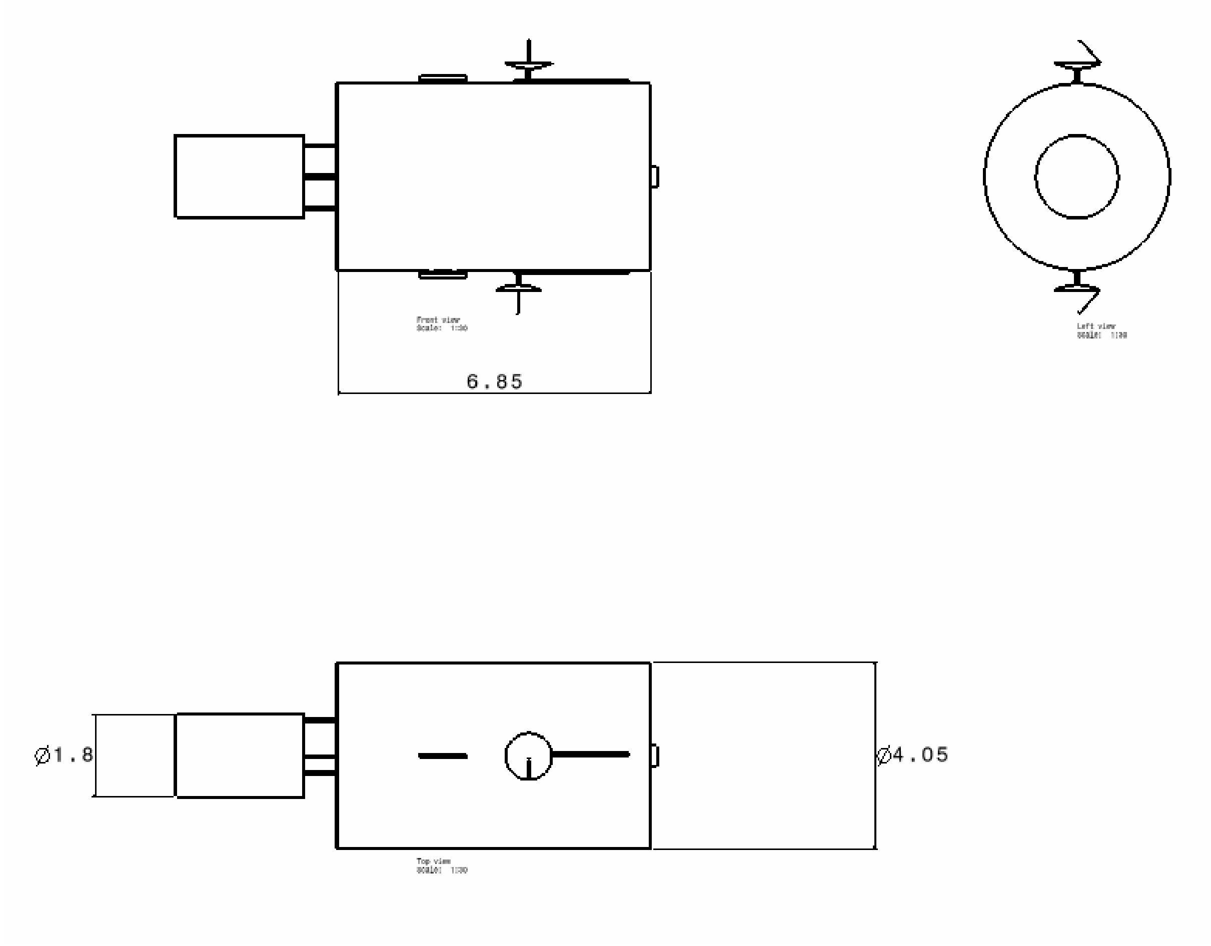


Figure 29-5: Stowed MORS

Figure 29-6 displays the isometric views for the fully deployed version of the MORS.

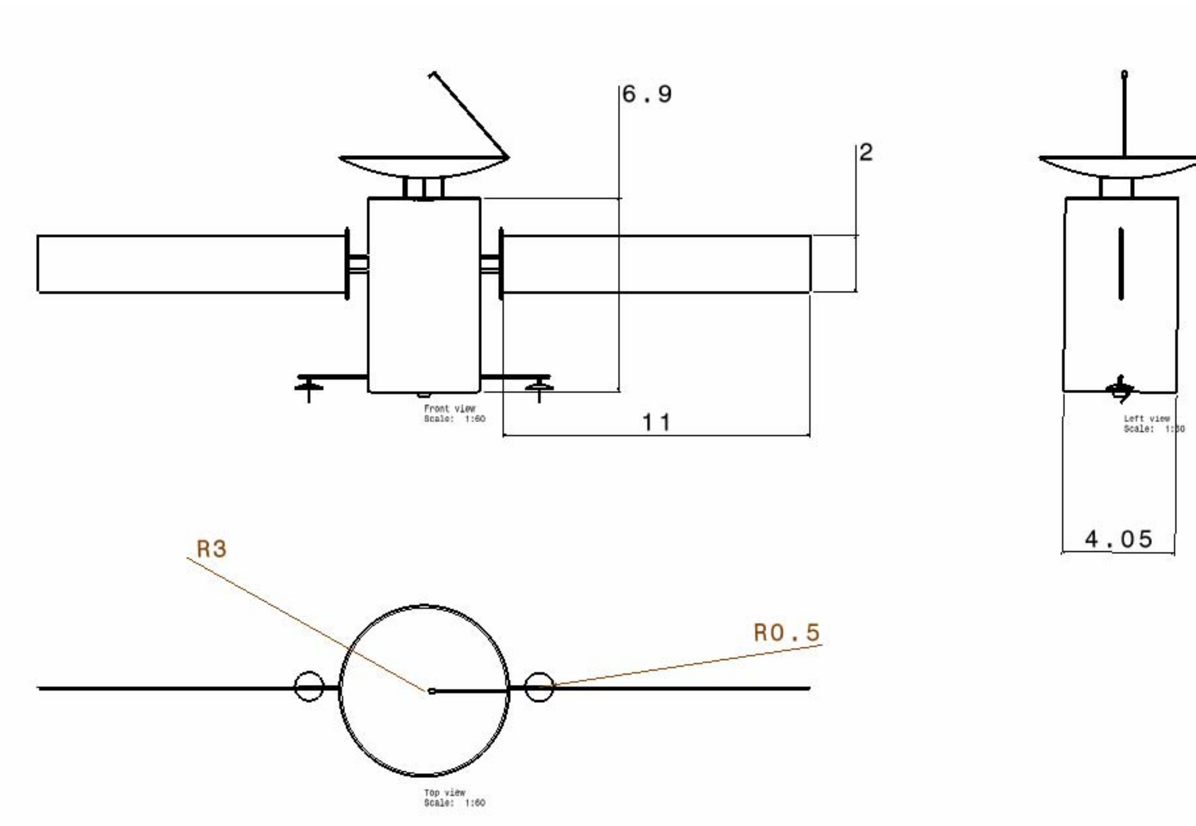


Figure 29-6: Deployed MORS

Figure 29-7 shows the isometric views for the stowed version of the HRS.

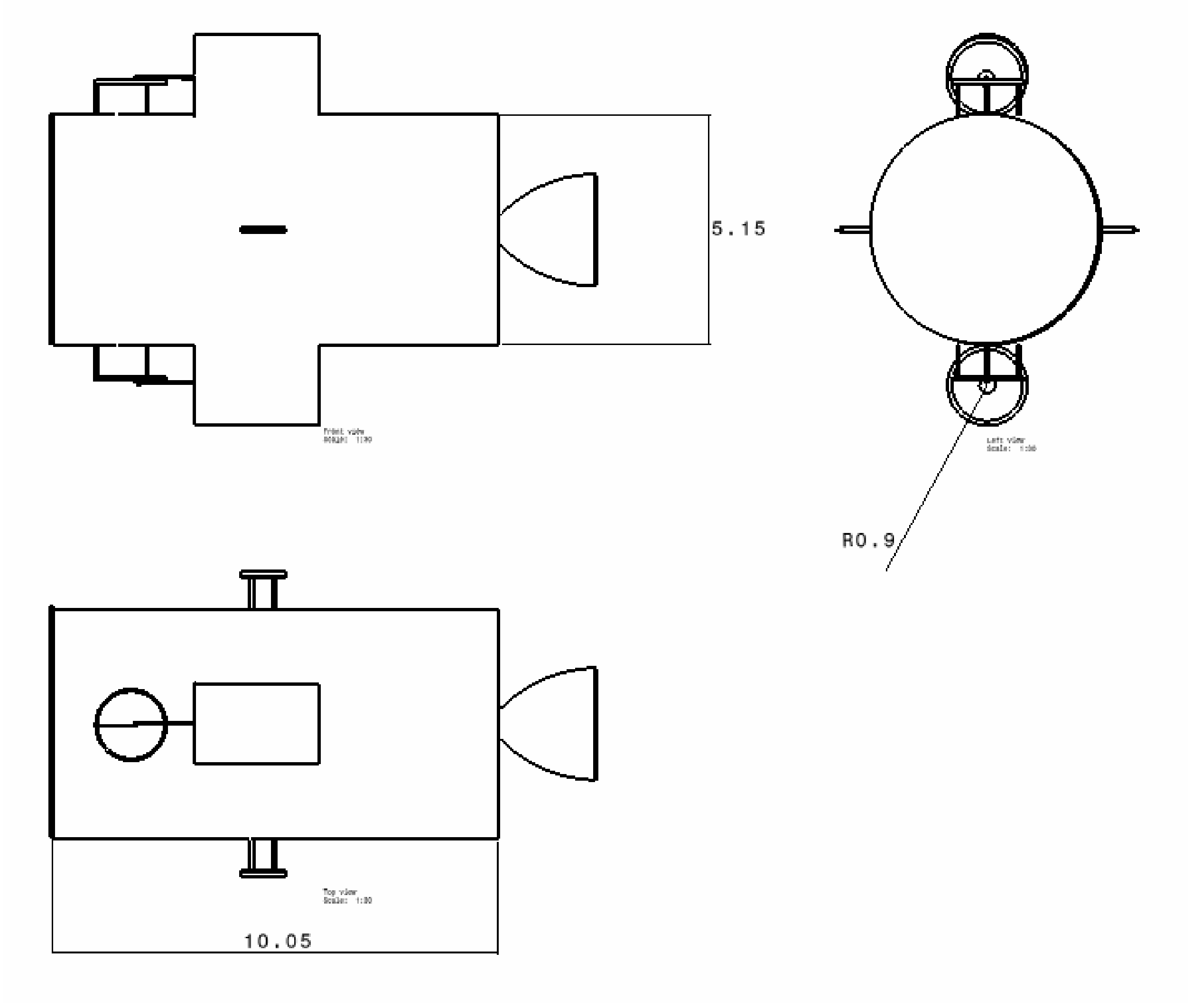


Figure 29-7: Stowed HRS

Figure 29-8 displays the isometric views for the fully deployed version of the HRS.

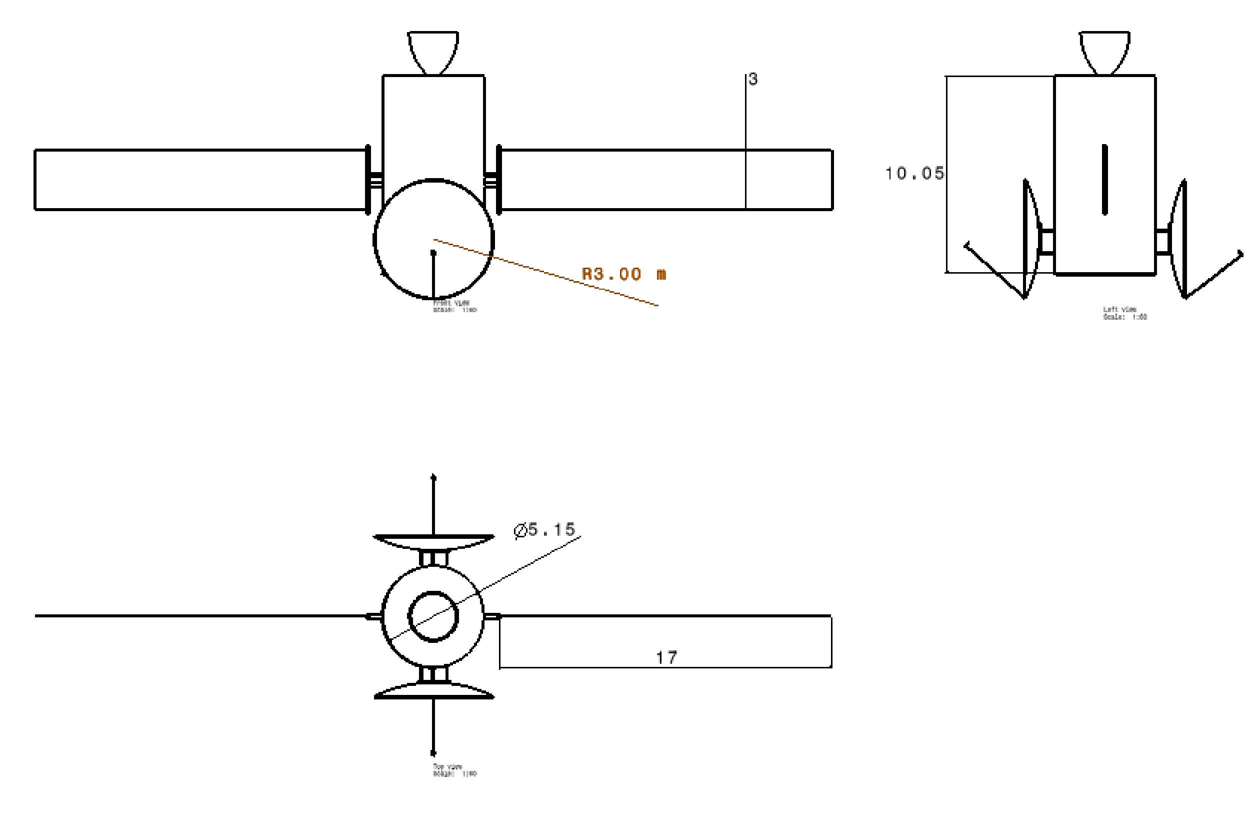


Figure 29-8: Deployed HRS

29.4.2 Selected 3-D Projections

The following three-dimensional projections help illustrate the overall appearance of the MORS and the HRS, as well as exemplify their communications functions.

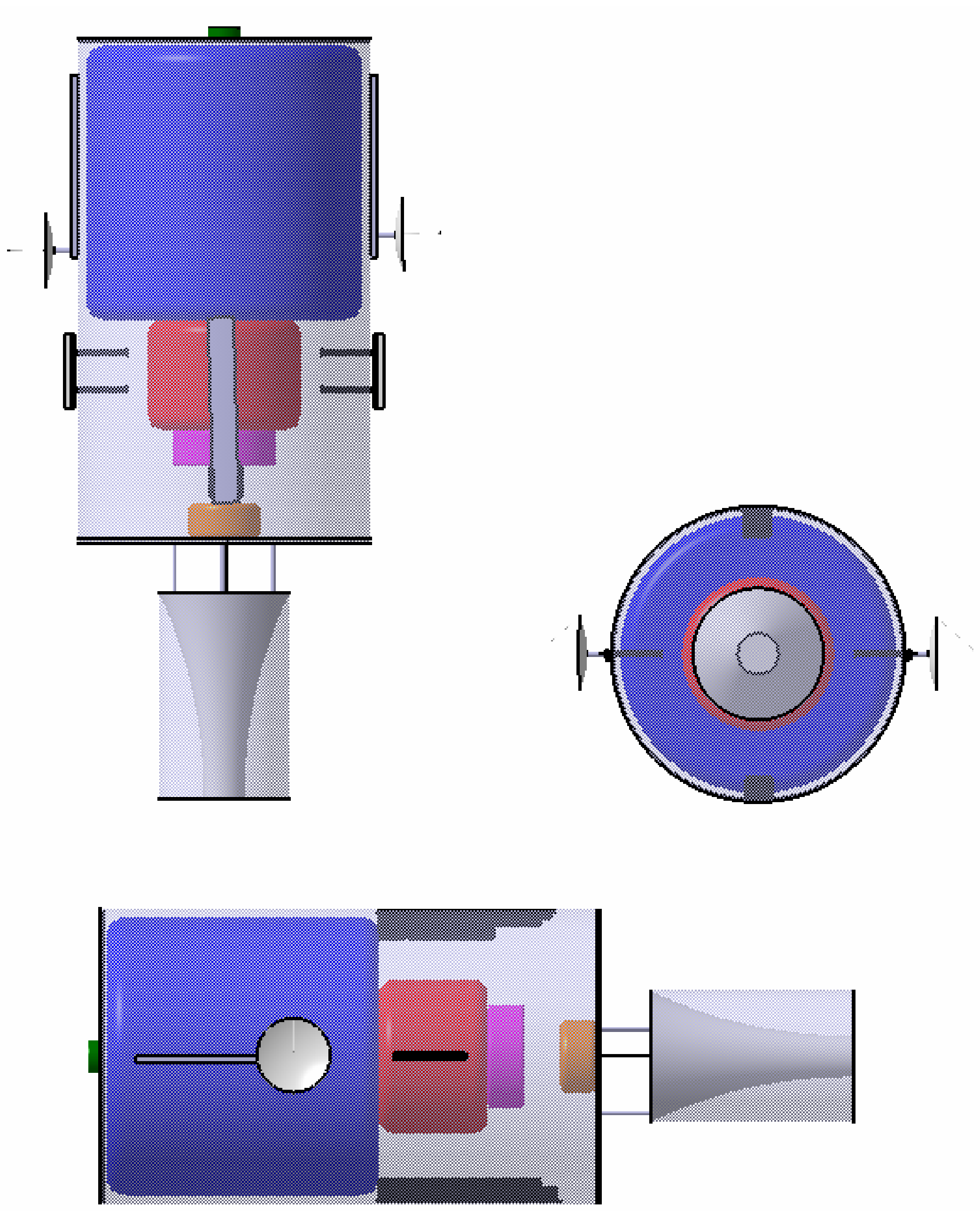


Figure 29-9: Stowed MORS

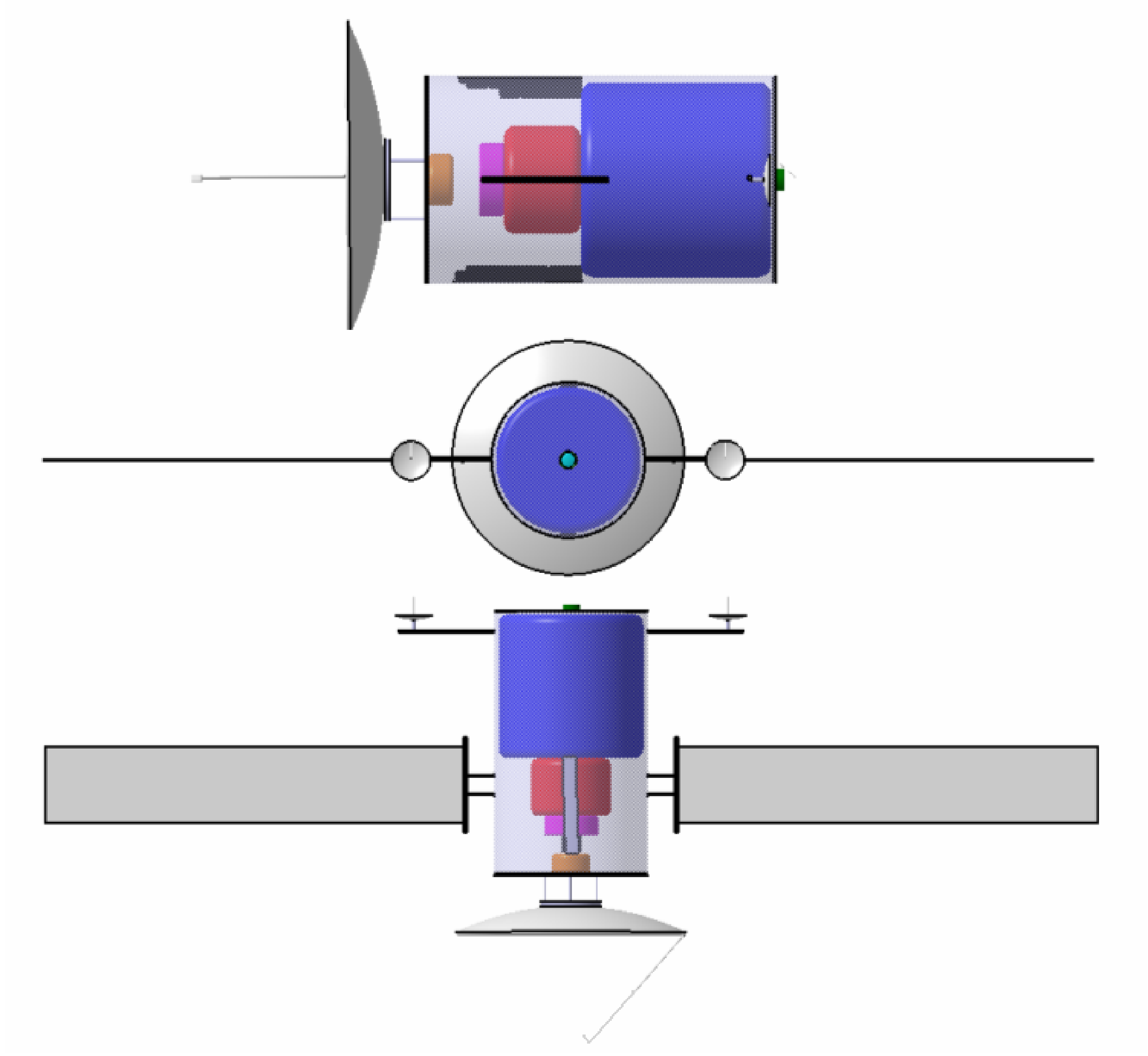


Figure 29-10: Deployed MORS

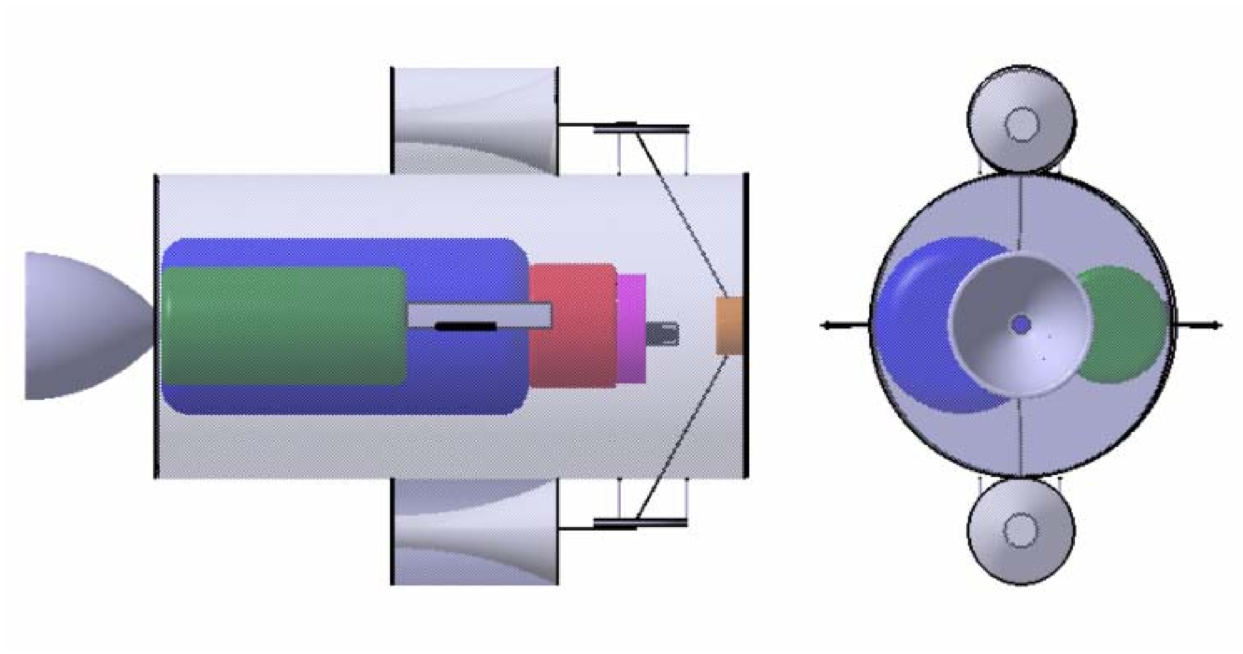


Figure 29-11: Stowed HRS

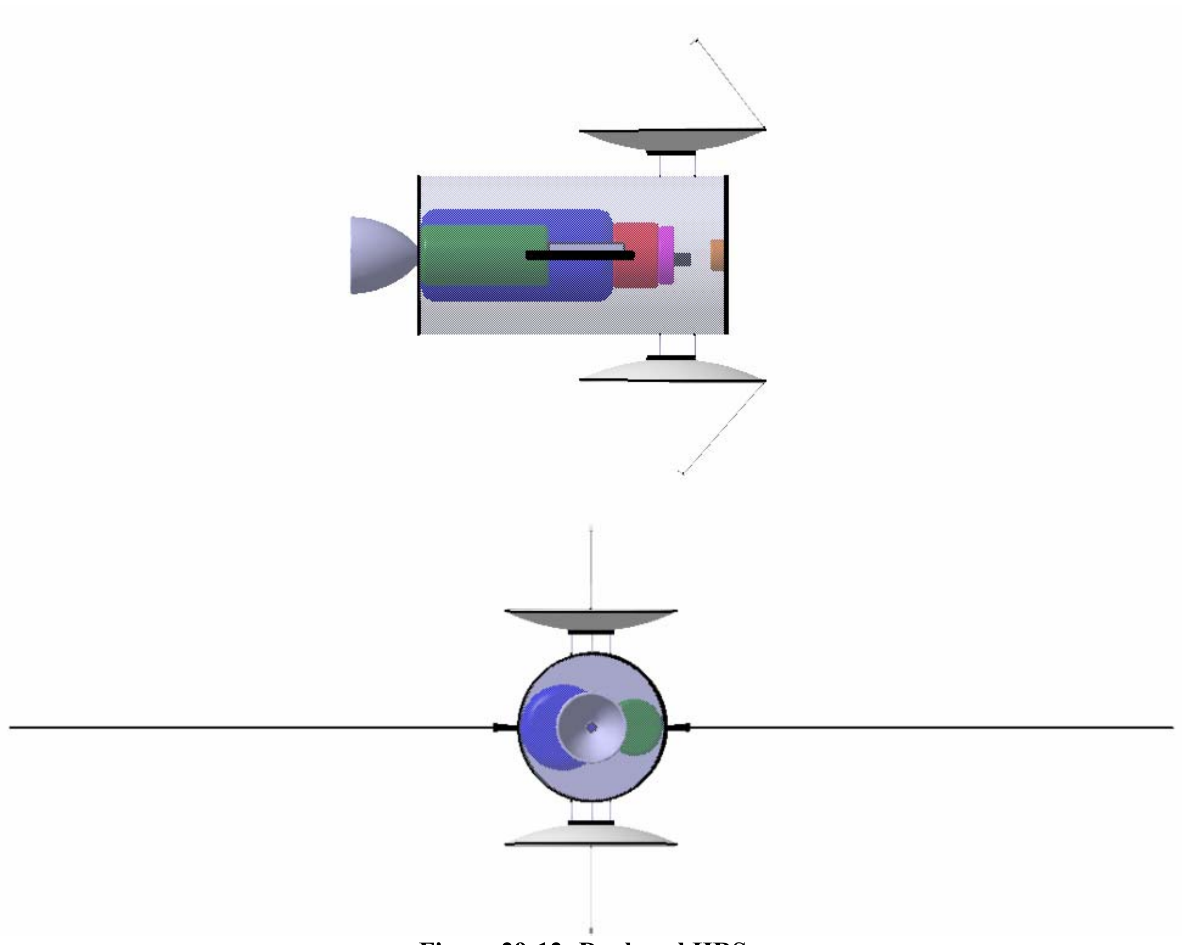


Figure 29-12: Deployed HRS

Figure 29-9 and Figure 29-10 display the MORS three-dimensionally, while Figure 29-11 and Figure 29-12 show the HRS in three dimensions.

References:

[1] "NASA's Mission Operations and Communications Services"
http://deepspace.jpl.nasa.gov/advmiss/docs/NASA_MO&CS.pdf

30 Metzger, Jeri Lynn

30.1 Communications Efforts

Author: Jeri Lynn Metzger

Contributors: Brendan Eash, Justin McCurdy, Zade Shaw

30.1.1 Parabolic Dish Sizing

We present our theory regarding communications design in the Crew Transport Vehicle (CTV) Antenna Hardware and Design, Section 2.3.1.1. The following scripts implement the link budget theory for the specific parameters of each vehicle, as modified from Brendan Eash's generic link budget code. In each modification of the code data rate, receiving dish diameter, and link distance are input. A plot of power required for varying dish diameters and signal-to-noise ratios is output for design decisions. From the plots, mass and power are traded for each vehicle.

30.1.1.1 Heliocentric Relay Satellite (HRS) Link Budget

We detail the Heliocentric Relay Satellite (HRS) analysis in the Communications Satellite System (CSS) section. The HRS is the last design consideration because the receiving dishes at Earth and Mars are predetermined from other link budget codes. The Earth dishes are twenty meters in diameter, while the CTV and MORS contain six meter dishes at Mars. We evaluate power requirements for each link of the HRS. We calculate both distances based on meeting a three degree minimum link angle, while minimizing the distance between the HRS and Mars. We calculate this distance for each blackout within our mission windows, as seen in Table 30-1. Optimizing the lag angle of the HRS from Mars and X-band radius determine the worst case link distances input to the link budget code.

Table 30-1: Lag Angle Optimizing

Mission	Mars r	Mars Θ*	HRS r	X-band r	Ka-band r	Ka-band Θ*
	(km)	(deg)	(km)	(km)	(km)	(deg)
2016	2.46e+8	-150.3	2.47e+8	2.58e+7	3.98e+8	3.7
2018	2.34e+8	-110.4	2.36e+8	2.47e+7	3.87e+8	3.6
2020	2.15e+8	-57.5	2.17e+8	2.27e+7	3.95e+8	3.3
2022	2.07e+8	-15.2	2.08e+8	2.17e+7	3.71e+8	3.5
2024	2.22e+8	-78.5	2.24e+8	2.34e+7	3.71e+8	3.6
2026	2.38e+8	-124.0	2.40e+8	2.51e+7	3.88e+8	3.7
2028	2.48e+8	-161.0	2.49e+8	2.60e+7	3.98e+8	3.7

```
clear all; close all; clc; format long; format compact;
%X band transmission between MORS and HRS
Rd = 10*log10(50e6); %bps - data rate
%Same full data rate as nominal
f = 8.4; %X Band
%worst case helio distance from Mars surface for 6 degree Mars lagging
S = 25990527.3391;
p_min = 1; %power value in kW
p_max = 100; %power value in kW
POWER = [p_min*1e3:1e3:p_max*1e3]; %Transmission Power, W
etaR = .65; %receiving antenna efficency
etaT = .7; %transmitting antenna efficency
Dr = 6; %diameter of receiving antenna on MORS/CTV, m
Dt = [0:1:10]; %diameter of transmitting antenna, m
% varying on the CTV, for talking to Earth and Mars surface
T = 150; %noise equivalent temperature of receiving system (K)
for i = 1:length(Dt)
    Dt(1) = .5;
    for j = 1:length(POWER)
           GT = 20.4 + 20*log10(Dt(i)) + 20 * log10(f) + 10*log10(etaT);
%gain of transmission antenna
           EIRP = 10*log10(POWER(j)) + GT; %effective isotropic radiated power
           L_space = 92.45 + 20*log10(f) + 20*log10(S); %space loss
           L_other = 3; %losses due to ineffeciences in system
           L_atm = 6; %atmospheric loss
           k = -228.6; %Boltzmann's constant, dB
           GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);
%gain of recieving antenna
           T_dB = 10 * log10(T);
           G_T = GR - T_dB;
           SNR_avial(i,j) = EIRP + G_T - L_space - L_other - k - Rd - L_atm;
    end
end
figure; hold on;
for i = 1:length(Dt)
    POWER_plot(i,:) = POWER/1e3;
end
plot(POWER_plot',SNR_avial');
plot([0 p_max],[3 3],'r-.');
plot([0 p_max],[1 1],'r--');
axis([0 60 0 10]);
%xlabel('Power (kW)');
xlabel('Power (kW)');
ylabel('S/N Ratio (dB)');
legend('.5m Dia.','1m Dia.','2m Dia.','3m Dia.','4m Dia.','5m Dia.',...
    '6m Dia.','7m Dia.','8m Dia.','9m Dia.','10m Dia.','Ideal S/N','Min. S/N',-1);
ttl = sprintf('Transmitting Antenna Power Required on MORS for
%im Recieving Antenna on HRS',Dr);
title(ttl);
%X band transmission between MORS and HRS
Rd = 10*log10(50e6); %bps - data rate
%Same full data rate as nominal
f = 8.4; %X Band

%worst case helio distance from Mars surface for 6 degree Mars lagging
S = 25990527.3391;
p_min = 1; %power value in kW
p_max = 100; %power value in kW
POWER = [p_min*1e3:1e3:p_max*1e3]; %Transmission Power, W
etaR = .65; %receiving antenna efficency
etaT = .7; %transmitting antenna efficency
Dr = 6; %diameter of receiving antenna on MORS/CTV, m
Dt = [0:1:10]; %diameter of transmitting antenna, m
% varying on the CTV, for talking to Earth and Mars surface
T = 400; %noise equivalent temperature of receiving system (K)
for i = 1:length(Dt)
    Dt(1) = .5;
    for j = 1:length(POWER)
           GT = 20.4 + 20*log10(Dt(i)) + 20 * log10(f) + 10*log10(etaT);
%gain of transmission antenna
           EIRP = 10*log10(POWER(j)) + GT; %effective isotropic radiated power
           L_space = 92.45 + 20*log10(f) + 20*log10(S); %space loss
           L_other = 3; %losses due to ineffeciences in system
           L_atm = 6; %atmospheric loss
           k = -228.6; %Boltzmann's constant, dB
           GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);
%gain of recieving antenna
           T_dB = 10 * log10(T);
           G_T = GR - T_dB;
           SNR_avial(i,j) = EIRP + G_T - L_space - L_other - k - Rd - L_atm;
    end
end


figure; hold on;
for i = 1:length(Dt)
    POWER_plot(i,:) = POWER/1e3;
end
plot(POWER_plot',SNR_avial');
plot([0 p_max],[3 3],'r-.');
plot([0 p_max],[1 1],'r--');
axis([0 60 0 10]);
%xlabel('Power (kW)');
xlabel('Power (kW)');
ylabel('S/N Ratio (dB)');
legend('.5m Dia.','1m Dia.','2m Dia.','3m Dia.','4m Dia.','5m Dia.',...
    '6m Dia.','7m Dia.','8m Dia.','9m Dia.','10m Dia.','Ideal S/N','Min. S/N',-1);
ttl = sprintf('Transmitting Antenna Power Required on MORS for %im Recieving Antenna on HRS',Dr);
title(ttl);
%Ka band transmission to Earth
Rd = 10*log10(50e6); %bps - data rate
f = 33; %Ka band
%worst case helio distance to Earth for 6 degree Mars lagging
S = 398220584.2605;
p_min = 1; %power value in kW
p_max = 100; %power value in kW
POWER = [p_min*1e3:1e3:p_max*1e3]; %Transmission Power, W
etaR = .55; %receiving antenna efficency
etaT = .55; %transmitting antenna efficency
Dr = 20; %diameter of receiving antenna on Earth, m
Dt = [0:1:10]; %diameter of transmitting antenna, m
% varying on the CTV, for talking to Earth and Mars surface
T = 150; %noise equivalent temperature of receiving system (K)

for i = 1:length(Dt)
    Dt(1) = .5;
    for j = 1:length(POWER)
        GT = 20.4 + 20*log10(Dt(i)) + 20 * log10(f) + 10*log10(etaT);
%gain of transmission antenna
        EIRP = 10*log10(POWER(j)) + GT; %effective isotropic radiated power
        L_space = 92.45 + 20*log10(f) + 20*log10(S); %space loss
        L_other = 3; %losses due to ineffeciences in system
        L_atm = 6; %atmospheric loss
        k = -228.6; %Boltzmann's constant, dB
        GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);
%gain of recieving antenna
        T_dB = 10 * log10(T);
        G_T = GR - T_dB;
        SNR_avial(i,j) = EIRP + G_T - L_space - L_other - k - Rd - L_atm;
    end
end
figure; hold on;
for i = 1:length(Dt)
    POWER_plot(i,:) = POWER/1e3;
end
plot(POWER_plot',SNR_avial');
plot([0 p_max],[3 3],'r-.');
plot([0 p_max],[1 1],'r--');
axis([0 60 0 10]);
%xlabel('Power (kW)');
xlabel('Power (kW)');
ylabel('S/N Ratio (dB)');
legend('.5m Dia.','1m Dia.','2m Dia.','3m Dia.','4m Dia.','5m Dia.',...
    '6m Dia.','7m Dia.','8m Dia.','9m Dia.','10m Dia.','Ideal S/N','Min. S/N',-1);
ttl = sprintf('Transmitting Antenna Power Required on HRS for %im Recieving Antenna on Earth',Dr);
title(ttl);
%Ka band transmission to Earth
Rd = 10*log10(50e6); %bps - data rate
f = 33; %Ka band
%worst case helio distance to Earth for 6 degree Mars lagging
S = 398220584.2605;
p_min = 1; %power value in kW
p_max = 100; %power value in kW
POWER = [p_min*1e3:1e3:p_max*1e3]; %Transmission Power, W
etaR = .55; %receiving antenna efficency
etaT = .55; %transmitting antenna efficency
Dr = 20; %diameter of receiving antenna on Earth, m
Dt = [0:1:10]; %diameter of transmitting antenna, m
% varying on the CTV, for talking to Earth and Mars surface
T = 400; %noise equivalent temperature of receiving system (K)
for i = 1:length(Dt)
    Dt(1) = .5;
    for j = 1:length(POWER)
        GT = 20.4 + 20*log10(Dt(i)) + 20 * log10(f) + 10*log10(etaT);
%gain of transmission antenna
        EIRP = 10*log10(POWER(j)) + GT; %effective isotropic radiated power
        L_space = 92.45 + 20*log10(f) + 20*log10(S); %space loss
        L_other = 3; %losses due to ineffeciences in system
        L_atm = 6; %atmospheric loss
        k = -228.6; %Boltzmann's constant, dB
        GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);
%gain of recieving antenna
        T_dB = 10 * log10(T);
        G_T = GR - T_dB;
        SNR_avial(i,j) = EIRP + G_T - L_space - L_other - k - Rd - L_atm;
    end
end
figure; hold on;
for i = 1:length(Dt)
    POWER_plot(i,:) = POWER/1e3;
end
plot(POWER_plot',SNR_avial');
plot([0 p_max],[3 3],'r-.');
plot([0 p_max],[1 1],'r--');
axis([0 60 0 10]);
%xlabel('Power (kW)');
xlabel('Power (kW)');
```

```
ylabel('S/N Ratio (dB)');
legend('.5m Dia.','1m Dia.','2m Dia.','3m Dia.','4m Dia.','5m Dia.',...
    '6m Dia.','7m Dia.','8m Dia.','9m Dia.','10m Dia.','Ideal S/N','Min. S/N',-1);
ttl = sprintf('Transmitting Antenna Power Required on HRS for %im Recieving Antenna on Earth',Dr);
title(ttl);
%Created by:  Brendan Eash,  Jan. 23, 2005
%Modified by:  Brendan Eash, Jan. 24, 2005
%Modified by:  Brendan Eash, Jan. 26, 2005
%Modified by: Jeri Metzger, Jan 29, 2005 for Mars Surface Ops
%Corrected by: Brendan Eash, Feb 5, 2005 GT eq'n
%Modified by: Jeri Metzger, Feb 12, 2005 Array trade for JIMO data
%Modified by: Jeri Metzger, Feb 23, 2005 Creating both links of HRS
```

30.1.1.2 Mars Habitat Vehicle (MHV) Link Budget

The communications section, 2.5.1.1, of the Mars Habitat Vehicle (MHV) details the MHV link budget. We modify the generic link budget code for the approximately 20,000 km MORS orbit, and the receiving dishes on the CTV and MORS.

```
%link_budget_MHV
clear all; close all; clc; format long; format compact;
Rd = 10*log10(100e6); %bps - data rate
% 100e6 for MHV
f = 8.4; %X band, GHz
%Worst Case Distance from Mars surface to MORS
S = 20418.9; %circular Martian Day Orbit
p_min = .1; %power value in kW
p_max = 100; %power value in kW
POWER = [p_min*1e3:1e3:p_max*1e3]; %Transmission Power, W
etaR = .55; %receiving antenna efficency
etaT = .55; %transmitting antenna efficency
Dr = 1; %diameter of receiving antenna on MORS/CTV, m
% may need to be varied for MHV
Dt = [0:1:4]; %diameter of transmitting antenna, m
% varying on the CTV, for talking to Earth and Mars surface
T = 400; %noise equivalent temperature of receiving system (K)
for i = 1:length(Dt)
    Dt(1) = .5;
    for j = 1:length(POWER)
           GT = 20.4 + 20*log10(Dt(i)) + 20 * log10(f) + 10*log10(etaT);
%gain of transmission antenna
           EIRP = 10*log10(POWER(j)) + GT; %effective isotropic radiated power
           L_space = 92.45 + 20*log10(f) + 20*log10(S); %space loss
           L_other = 3; %losses due to ineffeciences in system
           L_atm = 6; %atmospheric loss
           k = -228.6; %Boltzmann's constant, dB
           GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);
%gain of recieving antenna
           T_dB = 10 * log10(T);
           G_T = GR - T_dB;
           SNR_avial(i,j) = EIRP + G_T - L_space - L_other - k - Rd - L_atm;
    end
end
figure; hold on;
for i = 1:length(Dt)
    POWER_plot(i,:) = POWER/1e3;
end
plot(POWER_plot',SNR_avial');
plot([0 40],[3 3],'r-.');
plot([0 40],[1 1],'r--');
axis([0 30 0 40]);
%xlabel('Power (kW)');
xlabel('Power (kW)');
ylabel('S/N Ratio (dB)');
legend('.5m Dia.','1m Dia.','2m Dia.','3m Dia.','4m Dia.',...
    'Ideal S/N','Min. S/N',-1);
```

```
ttl = sprintf('Transmitting MHV Antenna Power Required for %im Receiving Antenna on CTV',Dr);
title(ttl);
%Created by:  Brendan Eash,  Jan. 23, 2005
%Modified by:  Brendan Eash, Jan. 24, 2005
%Modified by:  Brendan Eash, Jan. 26, 2005
%Modified by: Jeri Metzger, Jan 29, 2005 for Mars Surface Ops
%Modified by: Jeri Metzger, Jan 30, 2005 specific to MHV
%Corrected by: Brendan Eash, Feb 5, 2005 GT eq'n
%Modified by: Jeri Metzger, Feb 14, 2005 Circular Martian Orbit
%Modified by: Jeri Metzger, Feb 27,2005 1 Martian day circular MORS orbit
```

30.1.1.3 Mars Lander Vehicle (MLV) Preliminary Link Budget

During the design process, we begin the Mars Lander Vehicle (MLV) communications system with the same methodology as the Mars Habitat Vehicle (MHV) and Crew Transport Vehicle (CTV). This technique modifies the same generic link budget code, link_budget.m. The link distance decreased to the 20,000 km MORS' orbit, and the point at which a parabolic dish was no longer necessary. We include the new design process in the MLV communications section.

```
%link_budget_MLV
clear all; close all; clc; format long; format compact;
Rd = 10*log10(40e6); %bps - data rate
% 40e6 for MLV
f = 8.4; %X band, GHz
%Worst Case Distance from Mars surface
S = 20418.9; %circular Martian Day Orbit
p_min = .1; %power value in kW
p_max = 100; %power value in kW
POWER = [p_min*1e3:1e3:p_max*1e3]; %Transmission Power, W
etaR = .55; %receiving antenna efficency
etaT = .55; %transmitting antenna efficency
Dr = 1; %diameter of receiving antenna on CTV/MORS, m
% may need to be varied for MHV
Dt = [0:1:4]; %diameter of transmitting antenna, m
% varying on the CTV, for talking to Earth and Mars surface
T = 150; %noise equivalent temperature of receiving system (K)
for i = 1:length(Dt)
    Dt(1) = .5;
    for j = 1:length(POWER)
           GT = 20.4 + 20*log10(Dt(i)) + 20 * log10(f) + 10*log10(etaT);
%gain of transmission antenna
           EIRP = 10*log10(POWER(j)) + GT; %effective isotropic radiated power
           L_space = 92.45 + 20*log10(f) + 20*log10(S); %space loss
           L_other = 3; %losses due to ineffeciences in system
           L_atm = 6; %atmospheric loss
           k = -228.6; %Boltzmann's constant, dB
           GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);
%gain of recieving antenna
           T_dB = 10 * log10(T);
           G_T = GR - T_dB;
           SNR_avial(i,j) = EIRP + G_T - L_space - L_other - k - Rd - L_atm;
    end
end
figure; hold on;
for i = 1:length(Dt)
    POWER_plot(i,:) = POWER/1e3;
end
plot(POWER_plot',SNR_avial');
plot([0 p_max],[3 3],'r-.');
plot([0 p_max],[1 1],'r--');
axis([0 40 0 40]);
```

```
%xlabel('Power (kW)');
xlabel('Power (kW)');
ylabel('S/N Ratio (dB)');
legend('.5m Dia.','1m Dia.','2m Dia.','3m Dia.','4m Dia.',...
    'Ideal S/N','Min. S/N',-1);
ttl = sprintf('Transmitting MLV Antenna Power Required for %im Receiving Antenna on CTV',Dr);
title(ttl);
%Created by:  Brendan Eash,  Jan. 23, 2005
%Modified by:  Brendan Eash, Jan. 24, 2005
%Modified by:  Brendan Eash, Jan. 26, 2005
%Modified by: Jeri Metzger, Jan 29, 2005 for Mars Surface Ops
%Modified by: Jeri Metzger, Jan 30, 2005 specific to MLV
%Corrected by: Brendan Eash, Feb 5, 2005 GT eq'n
%Modified by: Jeri Metzger, Feb 14, 2005 Circular Martian Orbit
```

30.1.2 Communications Satellite System (CSS) Orbits

We develop the Communications Satellite System (CSS) orbits based on engineering judgment. MATLAB allows us to visualize the CSS orbits. We implement codes previously developed in Purdue's A&AE 532 – Orbital Mechanics.

30.1.2.1 Mars Orbital Relay Satellites (MORS) Orbit

We eventually choose a circular Martian orbit for the two Mars Orbiting Relay Satellites (MORS). This orbit keeps one of the satellites over the Mars Habitat Vehicle (MHV) and Mars Lander Vehicle (MLV) at all times, while the second satellite relays communications around the planet. We determine the radius of the MORS' circular orbit to create a7 Mars synchronous motion.

```
%MORS_orbit
%used to size Martian orbits for worst case comm distances
clear all; close all; clc;
%Martian Surface
r_circ = 3397.00;
theta_star1 = [0:.01:360]*pi/180;
x1 = r_circ.*cos(theta_star1);
y1 = r_circ.*sin(theta_star1);
%MORS Orbit
r_MORS = 20418.9;
x2 = r_MORS.*cos(theta_star1);
y2 = r_MORS.*sin(theta_star1);
%CTV Orbit
aT3 = 20418.9;
e3 = .816494;
p3 = aT3*(1-e3^2);
r3 = p3./(1+e3.*cos(theta_star1));
x3 = r3.*cos(theta_star1);
y3 = r3.*sin(theta_star1);
%MORS 1 Location
rm1 = 20418.9;
theta_star = deg2rad(0);
x4 = rm1*cos(theta_star);
y4 = rm1*sin(theta_star);
%MORS 2 Location
rm2 = 20418.9;
theta_star = deg2rad(120);
x5 = rm2*cos(theta_star);
y5 = rm2*sin(theta_star);
%CTV Location
```

```
rctv = 37090.8;
theta_star = deg2rad(180);
x6 = rctv*cos(theta_star);
y6 = rctv*sin(theta_star);
%MHV Location
rmhv = 3397;
theta_star = deg2rad(0);
x7 = rmhv*cos(theta_star);
y7 = rmhv*sin(theta_star);
%output
figure
plot(x1,y1,'r-','LineWidth',2)
hold on
plot(x2,y2,'g-','LineWidth',2)
plot(x3,y3,'k-','LineWidth',2)
plot(x4,y4,'gd','MarkerSize',8,'LineWidth',2)
plot(x5,y5,'cd','MarkerSize',8,'LineWidth',2)
plot(x6,y6,'ko','MarkerSize',8,'LineWidth',2)
plot(x7,y7,'rs','MarkerSize',8,'LineWidth',2)
axis equal
title('MORS Orbits')
xlabel('km')
ylabel('km')
legend('Mars Surface', 'MORS Orbit', 'CTV Orbit','MORS 1 Location','MORS 2 Location','CTV
Location','MHV Location')
% Created by: Jeri Metzger 29 January 2005
% Modified by: Jeri Metzger 26 February 2005 HRS Orbits
% Modified by: Jeri Metzger 15 March 2005 MORS Orbits
```

30.1.2.2 Heliocentric Relay Satellite (HRS) Orbit

We choose a Mars lagging orbit in developing the distances for the Heliocentric Relay Satellite (HRS) link budget. The orbit of the HRS is defined by the same radius and eccentricity as Mars.

```
%HRS_orbit
%used to size HRS orbits for worst case comm distances
clear all; close all; clc;
%Sun Surface
r_circ = 696000.00;
theta_star1 = [0:.01:360]*pi/180;
x1 = r_circ.*cos(theta_star1);
y1 = r_circ.*sin(theta_star1);
%Earth Orbit
theta_star2 = [0:.01:360]*pi/180;
r_earth = 1.49597807e8;
x3 = r_earth.*cos(theta_star2);
y3 = r_earth.*sin(theta_star2);

%Martian Orbit
aT3 = 2.27936636e8;
e3 = .09341233;
p3 = aT3*(1-e3^2);
r3 = p3./(1+e3.*cos(theta_star2));
x5 = r3.*cos(theta_star2);
y5 = r3.*sin(theta_star2);
%Earth location
re = 1.49597807e8;
theta_star = deg2rad(-161.007871+180);
x4 = re*cos(theta_star);
y4 = re*sin(theta_star);
%Mars location
rm = 2.4784e8;
theta_star = deg2rad(-161.007871);
x6 = rm*cos(theta_star);
y6 = rm*sin(theta_star);
%HRS location
```

```
rhrs = 2.4857e8;
theta_star = deg2rad(-167.007871);
x7 = rhrs*cos(theta_star);
y7 = rhrs*sin(theta_star);
%output
figure
plot(0,0,'k*','MarkerSize',8)
hold on
plot(x3,y3,'g-','LineWidth',2)
plot(x5,y5,'r-','LineWidth',2)
plot(x4,y4,'co','MarkerSize',8,'LineWidth',2)
plot(x6,y6,'ro','MarkerSize',8,'LineWidth',2)
plot(x7,y7,'md','MarkerSize',8,'LineWidth',2)
axis equal
title('2026 Blackout Orbits')
xlabel('km')
ylabel('km')
legend('Sun', 'Earth Orbit', 'Mars Orbit','Earth Location','Mars Location','HRS Location')
% Created by: Jeri Metzger 29 January 2005
%Modifed by: Jeri Metzger 26 February 2005 HRS Orbits
```

30.2 Vehicle Parameters

Author: Jeri Lynn Metzger

Contributors: Project Legend Design Team

Many calculations throughout the project require inputs from other systems or vehicles. We ensure accurate calculations by keeping a parameter spreadsheet on the A&AE 450 course website. The spreadsheet includes each vehicle: Earth Launch Vehicle (ELV), Ascent and Recovery Vehicle (ARV), Crew Transport Vehicle (CTV, Mars Lander Vehicle (MLV), Mars Habitat Vehicle (MHV), and Communications Satellite System (CSS). Each worksheet outlines the subsystems and details each vehicle to the component level. We track mass, volume, and power of the components, and iterate items depending on overall vehicle mass values as appropriate. The following figures are derived from the parameter spreadsheet file titled numbers.xls.

As of: 6:30PM 4/5/2005	Item	Mass	Volume	Power
ELV		kg	m^3	kW
Communications				
	Earth Telemetry	21.9	0.005	0.4
		21.9	0.005	0.4
Dynamics and Controls				
	Avionics	33.0	0.045	0.1
		33.0	0.045	0.1
Power				
	Battery	10.6	0.001	-0.9
		10.6	0.001	-0.9
Propulsion				
	SRB (total, quantity: 2)	1180000.0	1014.920	Pad
	RD-180 engines (quantity: 4)	21572.0	100.660	Self
	SSME (quantity: 3)	9531.0	26.540	-
	Stage 1: RP-1	300000.0	370.370	-
	Stage 1: LOX	814000.0	714.035	-
	Stage 2: LH2	39500.0	556.338	-
	Stage 2: LOX	237300.0	208.158	-
		2601903.0	2991.022	0.0
Structures				
	fairing	15679.0	undefined	-
	first stage	74724.0	undefined	-
	second stage	31339.0	undefined	-
		121742.0	0.000	0.0
Thermal				
	Cryogenic Tank Insulation	1550.0	40.330	undefined
		1550.0	40.330	0.0
Total		**2725260.5**	**3031.402**	**-0.3**
ELV		Mass	Volume	Power

Figure 30-1: ELV Parameters

As of: 1:30AM 4/6/2005	Item	Mass	Volume	Power
ARV		kg	m^3	kW
Science				
	Payload	100.0	0.500	-
		100.0	0.500	0.0
Aerodynamics				
	ballute	1819.0	undefined	-
	pressurized gases	40.0	8.480	undefined
	parachute	143.4	undefined	-
	heat shield	893.0	142.185	-
		2895.4	150.665	0.0
Communications				
	S-Band Transponder	2.9	0.003	0.0
	Omni Antenna	26.0	0.003	0.6
		28.9	0.006	0.6
Dynamics and Controls				
	Guidance System	132.3	0.109	1.0
		132.3	0.109	1.0
Human Resources				
	Crew	280.0	-	1.0
	Food	1.0	-	-
	Water	15.0	undefined	undefined
	Air (starting from Earth)	10.0	-	-
	O2 Consumption	4.2	-	-
	O2 resupply for leaks	1.5		
	O2 tanks & piping	2.1	-	-
	CO2 removal	8.8	-	-
	Diapers	1.0	-	-
	Avionics	20.0	-	-
	Seats	52.0	-	-
	Suits (4ACES)	180.0	undefined	undefined
	Fire Suppresion System	5.0	undefined	undefined
	Lights	5.0	undefined	0.1
	Docking Port	290.0	0.400	-
		875.5	0.400	1.1
Power				
	Batteries	91.0	0.024	-3.0
		91.0	0.024	-3.0

Propulsion				
	RCS inert	725.3	undefined	-
	RCS fuel	147.9	0.168	-
	RCS ox	242.6	0.168	-
		1115.8	0.337	0.0
Structures				
	Pressurized Volume	1315.0	8.300	-
	Helium Tank	4.4	undefined	-
	Support Components	500.0	undefined	-
		1819.4	8.300	0.0
Thermal				
	Thermal Control System	464.1	0.930	0.1
		464.1	0.930	0.1
Subtotal		4627.0		
Total		**7522.5**	**161.270**	**-0.2**
ARV		Mass	Volume	Power
LES				
Propulsion	Launch Escape Motor	2950.0	undefined	undefined
	Tower Jettison Motor	330.0	undefined	undefined
	Pitch Control Motor	11.5	undefined	undefined
		3291.5	0.000	0.0
Structures	Empty Mass	1600.0	undefined	undefined
		1600.0	0.000	0.0
Total		**4891.5**	**0.000**	**0.0**

Figure 30-2: ARV and LES Parameters

As of: 6:30PM 4/5/2005	Item	Mass	Volume	Power
CTV		kg	m^3	kW
External for Lander				
	Zero Boil-off Power	-	-	6.0
		0.0	0.000	6.0
Communications				
	SDST Transponder	5.8	0.005	0.0
	S Band Transponder	2.9	0.002	0.0
	Transmitting Dish to MHV	150.0	1m dia.	0.2
	Transmitting Dish to MLV	150.0	1m dia.	0.2
	Transmitting Dish to Earth	770.0	6m dia.	18.5
	Deployment for 6m	108.0	13.620	9.0
	HD Capable TVs	100.0	0.150	0.1
	Cameras	18.0	0.200	0.2
	cables, batteries, etc	100.0	undefined	undefined
	Inertia Track	200.0	undefined	1.0
		1604.7	13.977	29.2
Dynamics and Controls				
Navigation Hardware	Tier 1 Computers	24.0	0.025	0.2
	Tier 2 Computers	24.0	0.025	0.2
	Tier 3 Computers	49.0	0.051	0.5
	RF Hubs	2.0	0.007	0.0
	Personal Workstations	12.0	0.013	0.2
	File Server	3.1	0.006	0.1
	Replacement Batteries	1.0	0.001	-
	Ethernet Cable	30.0	0.020	-
	Coaxial Cable	53.0	0.035	-
	Safety Factor	4.0	0.037	0.0
	CT-632 Star Tracker (4)	18.4	6.400	0.0
		220.5	6.620	1.4

Human Resources				
Atmosphere Management	Atmosphere & Leakage: Nitrogen	974.2	-	-
	Nitrogen Tank	541.6	0.677	-
	Atmosphere and Leakage: Ox	292.3	undefined	undefined
	Oxygen Consumption	500.0	undefined	-
	Oxygen Tank	288.4	0.550	-
	Sabatier/Electrolysis	190.0	0.200	1.2
	Sabatier/Electrolysis Hydrogen	400.0	-	-
	Trace Contaminant Control System	79.8	0.272	0.2
	Storage Tanks	-	-	-
	Redundancy	27.0	0.100	-
Humans	Medical/Surgical/Dental	733.3	3.700	0.5
	Exercise Equipment	96.7	0.200	0.0
	Personal Preference Kits	100.0	2.000	-
	Personal Laptops	-	-	-
	Bunks	40.0	2.000	-
	Clothes (package all)	83.4	3.000	-
	Crew (+120 m^3 required volume)	280.0	120 m^3	-
	Redundancy	83.0	0.500	-
Water Management	Water Processing Assembly	658.0	2.200	0.9
	Consumption	7135.1	8.000	10.0
	Redundancy	780.0	1.100	-
Waste Management	Equipment & Consumables	1983.0	4.000	10.0
	Redundancy	916.0	10.400	-
Hygiene Facilities	Equipment	247.2	2.000	1.0
Food Preparations	Packaged Food (all frozen)	10120.0	35.200	-
	Long Term Freezer	7590.0	39.800	3.3
	Refridgerator/Freezer	412.0	2.000	0.0
	Food Warmer (2x)	10.0	0.010	0.0
	Kitchen Cleaning Supplies (wipes)	36.7	0.200	-
	Eating Utensils	7.3	0.000	-
	Repair Kits for Above Appliances	150.0	0.900	-
Fire Suppressant System		25.0	0.100	0.0
EVA	Suit (4MKII)	220.0	-	-
Maintanence	Vacuum (3)	13.0	0.100	0.0
	Hand Tools and accessories	150.0	1.000	0.0
	Redundancy	10.0	0.100	-
Docking	Port	580.0	0.800	-

Bunker Stuff	Waste Bags	2.8	0.010	-
	Fans	1.0		0.0
	Data Link	0.2	0.001	undefined
		35757.0	121.119	27.1
Power				
	SP100 Space Reactor	1763.0	4.948	-200.0
	Reactor Shielding	8400.0	undefined	-
		10163.0	4.948	-200.0
Propulsion				
	Main Engines (5)	10975.0	5.000	undefined
	Engine Shielding	20298.4	undefined	-
	spin-up/down propellant	1600.0	undefined	-
	xips	2000.0	undefined	40.0
	permanent tank	2327.0	undefined	-
	stationkeeping fuel	1800.0	undefined	-
	stationkeeping engines	700.0	undefined	-
		37200.4	5.000	40.0
Structures				
	Crew Quarters	15430.0	230.000	-
	super room	6456.0	undefined	-
	main truss	17540.0		-
	robotic arm	1800.0	undefined	undefined
	Mounting Structures	5787.6	undefined	-
		47013.6	0.000	0.0
Thermal				
Heat Acquisition	heat exchangers	52.1	0.119	-
	cold plates (1 kW cap)	240.0	0.560	-
Heat Transport	pumps/accumulator	347.9	1.232	0.8
	plumbing & valves	193.5	undefined	-
	instruments and controls	64.5	undefined	-
	fluids	64.5	undefined	-
	heat pumps	80.0	undefined	1.0
Hardware for Passive Thermal	heat pipes	319.7	0.221	-
	MLI	250.0	2.500	-
Heat Rejection	Radiators	223.5	1.578	1.0
	Cryogenic Storage	-	-	29.1
	Reactor Radiating	320.0	1.344	-
		2155.7	6.210	31.9
Structures Subtotal		57875.8		
Total		**134114.8**	**157.874**	**-64.4**
CTV		Mass	Volume	Power

Figure 30-3: CTV Parameters

As of: 6:30PM 4/5/2005	Item	Mass	Volume	Power
MLV		kg	m^3	kW
Science				
	Payload	100.0	0.500	-
		100.0	0.500	0.0
Aerodynamics				
	Parachutes	986.3	1.000	-
		986.3	1.000	0.0
Communications				
	S-Band Transponder	2.9	0.003	0.0
	Omni Antenna	26.0	0.003	0.6
		28.9	0.006	0.6
Dynamics and Controls				
	Guidance System	89.0	0.106	1.0
	Tracking and Radar Hardware	10.0	0.020	0.1
		99.0	0.126	1.0
Human Resources				
Atmosphere Management	LiOH system/ CO2 Removal	21.6	0.247	0.0
	Oxygen - Pressurization	4.8	-	-
	Oxygen - Breathing	10.4	-	-
	Nitrogen	16.9	-	-
	Atmosphere Tanks	-	-	-
Water Management	Drinking Water	19.7	undefined	-
	Water Tanks	2.0	0.020	-
Food System	Food	2.0	-	-
Waste Management	Diapers	2.0	-	-
	Wipes (hygiene/body/tp)	0.9	0.010	-
	Trash Bags (used food, used diapers)	0.1	0.012	-
Human Factors	Crew	280.0	-	-
	Seats/Sleeping Cots	52.0	1.060	-
EVA payload	Mark III	220.0	-	1
Other Systems	Smoke Detectors/Fire Extinguishers	5.0	0.100	undefined
	Lights	5.0	0.010	0.1
		642.5	1.349	0.1
Power				
	converters	10.0	undefined	0.9
	fuel cell	95.0	0.140	-6.0
		105.0	0.140	-5.1

Propulsion				
	Inert Mass: Descent	1336.6	undefined	-
	Inert Mass: Ascent	2885.3	undefined	-
	Inert Mass: RCS	506.3	undefined	-
	Inert Mass: Outer RCS	628.1	undefined	-
	Fuel Mass: Descent	777.1	10.945	-
	Fuel Mass: Ascent	2418.5	34.063	-
	Fuel Mass: RCS	127.9	0.326	-
	Fuel Mass: Outer RCS	158.6	0.180	-
	Ox Mass: Descent	4569.2	4.001	-
	Ox Mass: Ascent	14140.7	12.382	-
	Ox Mass: RCS	209.7	0.326	-
	Ox Mass: Outer RCS	260.1	0.180	-
	Engine: Pratt and Whitney RL-60 (2x)	1000.0	undefined	-
		29018.3	62.403	0.0
Structures				
	ascent stage structure	720.0	9.000	-
	descent stage structure	94.8	12.600	-
	legs	106.0	undefined	-
	docking hatch	290.0	undefined	-
	ladder	27.2	undefined	-
		1238.0	21.600	0.0
Thermal				
	Thermal Control System	584.5	1.010	2.1
	Heat Shield	5749.2	undefined	-
		6333.8	1.010	2.1
subtotal(for outer RCS)		37504.8		
subtotal (less heat shield)		31816.2		
Total		**38551.7**	**87.633**	**-1.3**
MLV		Mass	Volume	Power

Figure 30-4: MLV Parameters

As of: 6:30PM 4/5/2005	Item	Mass	Volume	Power
MHV		kg	m^3	kW
External for Lander				
	Zero Boil-Off Power	-	-	11.0
		0.0	0.000	11.0
Aerodynamics				
	ballute	3797.9	0.300	-
	helium and tank	8344.4	undefined	-
	heat shield	13170.3	100.000	-
	parachutes	1469.8	undefined	-
		26782.4	100.300	0.0
Communications				
	SDST Transponder	2.9	0.003	0.0
	Transmitting Dish	200.0	.5m dia.	0.2
	HD Capable TVs	100.0	0.100	0.1
	Cameras	18.0	0.100	0.2
	cables, batteries, etc	100.0	undefined	undefined
		420.9	0.203	0.5
Dynamics and Controls				
Navigation Hardware	Tier 1 Computers	24.0	0.025	0.2
	Tier 2 Computers	24.0	0.025	0.2
	Tier 3 Computers	49.0	0.051	0.5
	RF Hubs	2.0	0.007	0.0
	Personal Workstations	12.0	0.013	0.2
	File Server	3.1	0.006	0.1
	Replacement Batteries	1.0	0.001	-
	Ethernet Cable	30.0	0.020	-
	Coaxial Cable	53.0	0.035	-
	Safety Factor	4.0	0.037	0.0
		202.1	0.220	1.3
Human Resources				
Atmosphere Management	Atmosphere Gases	143.0	undefined	0.1
	Trace Contaminant Control	79.8	0.272	0.2
	Redundancy	10.0	-	1.0

Humans	Medical/Surgical/Dental	166.7	undefined	0.5
	Exercise Equipment	96.7	undefined	0.0
	Personal Preference Kits	100.0	undefined	-
	Bunks	40.0	undefined	-
	Clothes (package all)	28.4	undefined	-
	Crew (96 m^3 required Volume)	280.0	93.000	-
	Redundancy	160.0	1.500	-
Water Management	Water Processing Assembly	658.0	2.200	0.9
	Consumption	6924.3	8.000	10.0
	Redundancy	758.0	1.200	-
Waste Management	Equipment & Consumables	975.0	4.000	10.0
	Redundancy	25.0	2.000	-
Sabatier/Electrolysis	Sabatier	0.2	0.060	3.0
	Hydrogen	541.0	undefined	-
	Hydrogen Tank	347.0	13.030	-
	Condenser	3.0	0.030	undefined
	Heat Exchanger	5.7	0.050	undefined
	Electrolysis Machine	15.0	0.200	undefined
	Piping and Valves	5.0	0.070	-
	Redundancy	110.0	1.300	-
Hygiene Facilities	Equipment & Consumables	232.3	2.000	1.0
Food Preparations	Packaged Food (Assume all frozen)	1750.0	undefined	-
	plant closed system	2800.0	undefined	-
	hydroponic plant system	1170.0	40.000	18.0
	Nutrient Tank	180.0	1.200	-
	Long Term Freezer	1415.0	9.000	1.1
	Refridgerator/Freezer (2x)	824.0	undefined	0.0
	Food Warmer (2x)	10.0	undefined	0.0
	Dishwasher	40.0	undefined	0.0
	Oven	50.0	undefined	0.0
	Kitchen Cleaning Supplies (wipes)	8.7	undefined	-
	Eating Utensils	3.3	undefined	-
	Redundancy	700.0	undefined	-
Fire Suppressant System		25.0	0.100	0.0
EVA	Tools/Workaids	150.0	-	-
Maintanence	Vacuum (3)	13.0	undefined	0.0
	Hand Tools and accessories	150.0	undefined	0.0
	Redundancy	125.0	undefined	-
		21118.0	179.212	45.8

Power				
	SP100 Space Reactor	1087.0	2.863	-155.6
	Power Cables	320.0	undefined	20.0
	Reactor Shielding	4200.0	undefined	-
		5607.0	2.863	-135.6
Propulsion				
	engine	382.6	0.053	-
	engine shielding	7200.0	undefined	-
	additional hardware	700.0	undefined	-
Transfer	Fuel Mass	86240.0	undefined	-
RCS	Inert Mass	794.1	undefined	-
	Oxygen Mass	328.9	undefined	-
	Fuel Mass	200.5	undefined	-
Propulsive Descent	Fuel	4600.0	undefined	-
	4 RD-0146	968.0	undefined	undefined
		101414.1	0.053	0.0
Structures				
	External Structure	3827.0	211.000	-
	Super Room	6456.0	undefined	-
	Airlock	800.0	5.000	-
		11083.0	5.000	0.0
Thermal				
Heat Acquisition	heat exchangers	54.4	0.130	-
	cold plates (1kw cap.)	240.0	0.560	-
Heat Transport	pumps/accumulators	394.2	1.390	0.9
	plumbing and valves	199.8	undefined	-
	instruments and controls	66.6	undefined	undefined
	fluids	66.6	undefined	-
	heat pumps (5 kW cap)	80.0	undefined	1.0
Hardware for Passive Thermal Con	heat pipes	360.4	0.250	-
	MLI	205.0	undefined	-
Heat Rejection	Radiators (minus convection)	384.2	2.710	1.0
	Cryogenic Storage	-	-	48.0
	Reactor Radiating	320.0	1.344	
		2371.2	6.384	50.9
subtotal to aerobrake (heat shield)		65851.7		
subtotal to land (parachute)		47413.6		
Total		**168998.7**	**294.234**	**-26.0**
MHV		Mass	Volume	Power

Figure 30-5: MHV Parameters

As of: 6:30PM 4/5/2005	Item	Mass	Volume	Power
MORS		kg	m^3	kW
Communications				
	SDST Transponder	8.7	0.008	0.1
	Transmitting Dish to MHV	150.0	1m dia.	0.2
	Transmitting Dish to MLV	150.0	1m dia.	0.2
	Transmitting Dish to Earth/HRS	385.0	6m dia.	18.5
	Deployment for 6m	51.3	6.180	-
		745.0	6.188	19.0
Dynamics and Controls				
	Avionics	132.3	0.109	1.0
		132.3	0.109	1.0
Power				
	SP100 Space Reactor	985.0	2.319	-44.4
	Reactor Shielding	4000.0	undefined	-
		4985.0	2.319	-25.0
Propulsion				
	Transfer Propellant and System	822.0	42.500	16.5
		822.0	42.500	0.0
Structures				
	Structural	1314.3	undefined	-
	Propellant Tank	27.6	undefined	-
		1341.9	0.000	0.0
Thermal				
Heat Acquisition	heat exchangers	0.0	0.000	0.0
	cold plates (1kw cap.)	72.0	0.168	0.0
Heat Transport	pumps/accumulators	107.8	0.382	0.5
	plumbing and valves	39.9	undefined	-
	instruments and controls	13.3	undefined	undefined
	fluids	13.3	undefined	-
	heat pumps (5 kW cap)	0.0	0.000	1.0
Hardware for Passive Therm	heat pipes	66.0	0.046	-
	MLI	20.0	0.200	-
Heat Rejection	Radiators (minus convection)	186.2	1.310	1.0
	Reactor Radiating	163.0	0.240	-
		681.5	2.346	2.5
subtotal		6571.4		
Total		**8707.7**	**53.462**	**-2.5**
MORS		Mass	Volume	Power

Figure 30-6: MORS Parameters

As of: 6:30PM 4/5/2005	Item	Mass	Volume	Power
HRS		kg	m^3	kW
Communications				
	SDST Transponder	5.8	0.005	0.0
	Transmitting Dish to MORS	385.0	6m dia.	18.5
	Transmitting Dish to Earth	385.0	6m dia.	29.5
	Deployment for 6m	102.6	13.620	-
		878.4	13.625	48.0
Dynamics and Controls				
	Avionics	132.3	0.109	1.0
		132.3	0.109	1.0
Power				
	SP100 Space Reactor	1005.0	2.319	-66.7
	Reactor Shielding	4000.0	undefined	-
		5005.0	2.319	-55.0
Propulsion				
	Oxygen	15000.0	13.160	0.5
	Hydrogen	3000.0	44.280	0.8
		18000.0	57.440	1.3
Structures				
	Structural	2137.8	undefined	-
	Oxygen Tank	945.0	undefined	-
	Hydrogen Tank	330.0	undefined	-
		3412.8	0.000	0.0
Thermal				
Heat Acquisition	heat exchangers	0.0	0.000	0.0
	cold plates (1kw cap.)	72.0	0.168	
Heat Transport	pumps/accumulators	249.3	0.883	1.2
	plumbing and valves	74.1	undefined	-
	instruments and controls	24.7	undefined	undefined
	fluids	24.7	undefined	-
	heat pumps (5 kW cap)	0.0	0.000	1.0
Hardware for Passive Therm	heat pipes	152.7	0.105	-
	MLI	20.0	0.200	-
Heat Rejection	Radiators (minus convection)	431.0	3.040	-
	Reactor Radiating	212.0	0.500	-
		1260.4	4.896	2.2
subtotal		10689.0		
Total		**28689.0**	**78.389**	**-2.5**
HRS		Mass	Volume	Power

Figure 30-7: HRS Parameter

31 Mrozek, Kimberly

31.1 Initial MHV Trajectory Analysis

Author: Kimberly Mrozek

The initial Mars Habitat Vehicle (MHV) trajectory we examined was a Hohmann transfer. This assumed that the MHV would have identical Earth and Mars parking orbits that are circular with an altitude of 300 km. We also assumed that the orbits of Mars and Earth around the Sun are circular and co-planar. Distance from Sun to Earth was taken to be 149,597,807 km, and distance from Sun to Mars was taken to be 227,936,636 km. gravitational parameter of the Sun was taken to be 1.32712 x 10^{11} km^3/s^2. These numbers were averages given to AAE 532 students to use in orbital mechanics homework. A picture of a Hohmann transfer is shown below in Figure 31-1Hohmann transfer.

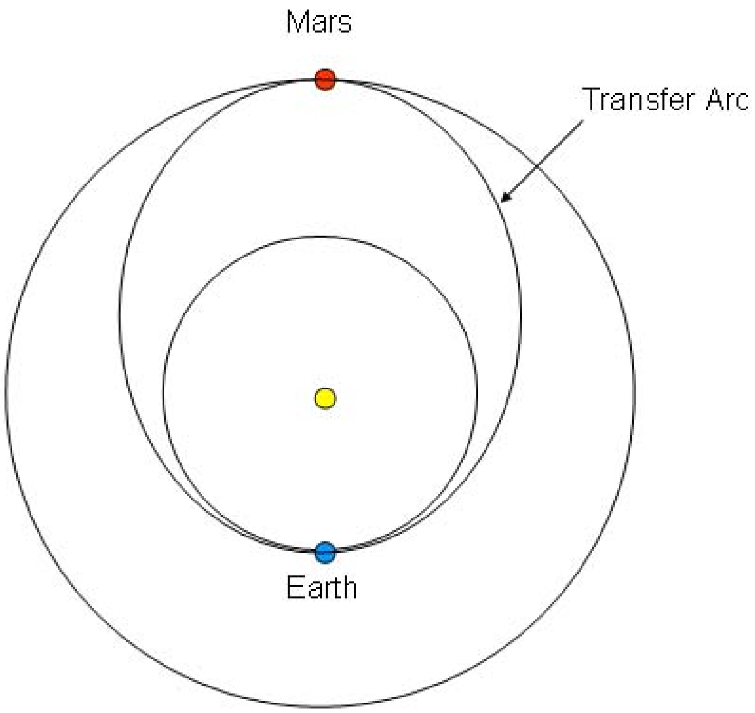


Figure 31-1Hohmann transfer

The velocity of the vehicle in the Earth parking orbit is found by using the circular velocity equation (Equation 31-1) where v_1 is the velocity, μ_s is the gravitational parameter, and r_1 is the distance from the Sun to the Earth. This velocity is solved to be 29.78 km/s.

$$v_1 = \sqrt{\frac{\mu_s}{r_1}}$$

Equation 31-1

The semimajor axis of the transfer ellipse was found with Equation 31-2, where a_T is the semi major axis, and r_2 is the distance from the Sun to Mars. Semi major axis was calculated to be 188,767,521.5 km.

$$a_T = \frac{1}{2}(r_1 + r_2)$$

Equation 31-2

The velocity of the vehicle on the transfer arc at the point of departure (leaving Earth parking orbit) is determined by using the elliptical velocity equation (Equation 31-3) where v is the velocity. This velocity is 32.73 km/s.

$$v = \sqrt{\frac{2\mu_s}{r_1} - \frac{\mu_s}{a_T}}$$

Equation 31-3

The first maneuver is a burn for the vehicle to increase velocity from the circular velocity to the elliptical velocity. This equation is the delta v equation (Equation 31-4), where Δv_1 is the change in velocity for this first maneuver. Δv_1 was calculated to be 2.95 km/s.

$$\Delta v_1 = v - v_1$$

Equation 31-4

The velocity of the MHV at Mars arrival is found by using Equation 31-3, with the exception of using r_2 in place of r_1. This velocity is v_a and was found to be 21.48 km/s. The velocity of the MHV in Mars parking orbit is determined by using Equation 31-1, again using r_2 in place of r_1. This velocity is v_2 and was found to be 24.12 km/s. The second maneuver is a burn to change the velocity from that of the elliptical arrival velocity to the circular parking orbit velocity. We found this delta v by using Equation 31-4, using v_2 in place of v and v_a in place of v_1. This second delta v is Δv_2 and was found to be 2.65 km/s. The total delta v required for the trajectory is the sum of the delta v's and is found with Equation 31-5. The total delta v was found to be 5.60 km/s.

$$\textit{total}\ \Delta v = \Delta v_1 + \Delta v_2$$

Equation 31-5

The time of flight for this trajectory is found with Equation 31-6, where TOF is the time of flight. We found that the time of flight is approximately 259 days.

$$TOF = \pi \sqrt{\frac{a_T^3}{\mu_s}}$$

Equation 31-6

31.2 MHV Maneuver Burn Analysis

Author: Kimberly Mrozek

Contributors: Meredith Evans, Eric Gustafson

After delta v information was determined for each Hohmann transfer launch window and for multiple departure dates for each window, we then used the delta v information to complete a burn analysis to determine burn times and the corresponding required fuel mass. Our assumptions for the vehicle propulsion were a mass flow rate of 1.6 kg/sec, an Isp of 1007 seconds, and one engine. At this time the final mass of the MHV was 82543.31 kg. We also assumed a total of three burns: one to leave Earth parking orbit, one to change the inclination halfway through the transfer arc, and one to enter Mars parking orbit. We had three main equations to calculate burn times and propellant mass. The first was Equation 31-7, where m_o is the total mass, m_f is the final mass, and m_p is the mass of the propellant.

$$m_o = m_f + m_p$$

Equation 31-7

Equation 31-8 is the equation for burn time, where t_p is the burn time, and mdot is the mass flow rate.

$$t_p = m_p / \text{mdot}$$

Equation 31-8

Equation 31-9 is the rocket equation, where Δv is the delta v, I_{sp} is the specific impulse, and g is the earth's gravitational constant of 9.81 m/s^2.

$$m_p = m_f e^{\left(\frac{\Delta v}{I_{sp} g}\right)} - m_f$$

Equation 31-9

Below is the main code used to output trajectory information, and burn analysis information:

```
Main MATLAB output code
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This code calls on "orbit3" and runs for a specified range of julian
%dates. It then spits out the total delta_v and tof.

clear all; close all; clc;

counter=0;
for jdate=2459917:1:2459917
    counter=counter+1;
    output=orbit3(jdate);
%     [jdateofcounter, TA, tof_day, delta_v1, delta_v2, deltav_tot]=output
    jdateofcounter(counter)=output(1);
    TA(counter)=output(2);
    tof_day(counter)=output(3);
```

```
        delta_v1(counter)=output(4);
        delta_v2(counter)=output(5);
        deltav_tot(counter)=output(6);
        a_a(counter)=output(7);
        deltav_inc_change(counter)=output(8);
        mass_prop_A(counter)=output(9);
        time_burn_A(counter)=output(10);
        mass_prop_inc(counter)=output(11);
        time_burn_inc(counter)=output(12);
        mass_prop_D(counter)=output(13);
        time_burn_D(counter)=output(14);
        mass_prop_total(counter)=output(15);
    end

      TA=TA'
      tof_day=tof_day'
      delta_v1=delta_v1'
      delta_v2=delta_v2'
      deltav_tot=deltav_tot'
      a_a=a_a'
      deltav_inc_change=deltav_inc_change'
      mass_prop_A = mass_prop_A'
      time_burn_A = time_burn_A'
      mass_prop_inc = mass_prop_inc'
      time_burn_inc = time_burn_inc'
      mass_prop_D = mass_prop_D'
      time_burn_D = time_burn_D'
      mass_prop_total=mass_prop_total'
```

This MATLAB code is orbit3.m and uses the same process stated in Non-Free Return Appendix for the code orbit3_minimize.m.

```
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This code calls on "planet data" and finds the best delta_V from a range
%of time of flights and and a specified julian date.


function out=orbit2(jdate)

counter=0;
for TOF=380:1:380
    counter=counter+1;
    [r1_vector, v1_vector, theta_starE, r2_vector_junk, v2_vector_junk, theta_star_junk]
=planet_data(jdate);
    [RE_vector_junk, VE_vector_junk, theta_star_junk, r2_vector, v2_vector, theta_starM]
=planet_data(jdate+TOF);
    r2_vector_actual=r2_vector
    r1_vector(3)=0;
    r2_vector(3)=0;
    v1_vector(3)=0;
    v2_vector(3)=0;
    r1=norm(r1_vector); r2=norm(r2_vector);
    v1=norm(v1_vector); v2=norm(v2_vector);
    c_vector=r2_vector - r1_vector;
    c=norm(c_vector);
    s=0.5*(c+r1+r2);
    r1_hat=r1_vector/r1;
    r2_hat=r2_vector/r2;
    h=cross(r1_vector,r2_vector);
    del=acos((dot(r1_vector,r2_vector))/(r1*r2));
    if h(3)<0
        h=-h;
        TA(counter)=(2*pi)-del;
    else
        h=h;
        TA(counter)=del;
    end
    TA(counter)
    muE=398600.485043; mu=1.3271244E+11; muM=4.28282868534e+04;
    h_mag=norm(h);
    h_hat=h/h_mag;

    %Earth transformation matrix
    theta_hat=cross(h_hat,r1_hat);
    trans_matrix=[r1_hat(1) theta_hat(1) h_hat(1);r1_hat(2) theta_hat(2) h_hat(2);r1_hat(3)
theta_hat(3) h_hat(3)];
    inc=acos(h_hat(3));
    omega1=acos(h_hat(2)/-sin(inc));
    omega2=asin(h_hat(1)/sin(inc));
    theta1=acos(theta_hat(3)/sin(inc));
    theta2=asin(r1_hat(3)/sin(inc));

    %Mars transformation matrix
    theta_hat_M=cross(h_hat,r2_hat);
    trans_matrix_M=[r2_hat(1) theta_hat_M(1) h_hat(1);r2_hat(2) theta_hat_M(2) h_hat(2);r2_hat(3)
theta_hat_M(3) h_hat(3)];

    %inclination change
    r2_hat_actual=r2_vector_actual/norm(r2_vector_actual);
    r1_hat_new=cross(r2_hat_actual,h_hat);
    h_new=cross(r1_hat_new,r2_hat_actual);
    h_new_hat=h_new/norm(h_new);
    inclination=acos(h_new_hat(3))
    incli=inclination*180/pi

    %%%%%%%%%%%%%%%%%%%%%
    %Minimum Conditions
    %%%%%%%%%%%%%%%%%%%%%

    a_min=0.5*s;                            %Minimum semi-major axis [Km]
    energy_min=-mu/(2*a_min);               %Energy associated with a_min [Km^2/sec^2]
    alpha_min=pi;                           %[rad]
    beta_min=2*asin(sqrt((s-c)/(2*a_min))); %[rad]
    P_min=(4*a_min*(s-r1)*(s-r2)*(sin(0.5*(alpha_min+beta_min)))^2)/c^2;
    e_min=sqrt(1-P_min/a_min);               %Eccentricity associated with a_min
    TOF_min=((a_min^1.5)/sqrt(mu))*((alpha_min-beta_min)-(sin(alpha_min)-sin(beta_min)));

    %%%%%%%%%%%%
    %Finding "a"
    %%%%%%%%%%%%
    TA(counter)=TA(counter)*180/pi;

    if TA(counter)<180
        TOF_par=(sqrt(2/mu)*(s^1.5-(s-c)^1.5))/3;   %Type 1
    else
        TOF_par=(sqrt(2/mu)*(s^1.5+(s-c)^1.5))/3;   %Type 2
    end

    TOF_Vector=24*3600*[TOF];
    a0=a_min;

    for j=1:length(TOF_Vector),
        TOFh=TOF_Vector(j)
        if TA(counter)<180
            if (TOFh<TOF_par)
                fprintf('\n\nHyperbola-1H\n\n');
                a0h=-a0;
                FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
(2*asinh(sqrt(s/(2*abs(a))))))-(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                a(j)=fsolve(FUN,a0h,[],s,TOFh,c,mu);
            else
                if (TOFh<TOF_min)
                    fprintf('\n\Ellipse-1A\n\n');
                    FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
sin(2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN2,a0,[],s,TOFh,c,mu);
                else
```

```
                    fprintf('\n\Ellipse-1B\n\n');
                    FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN3,a0,[],s,TOFh,c,mu);
                end
            end
        else
            if (TOFh<TOF_par)
                fprintf('\n\nHyperbola-2H\n\n');
                a0h=-a0;
                FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
(2*asinh(sqrt(s/(2*abs(a))))))+(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                a(j)=fsolve(FUN,a0h,[],s,TOFh,c,mu);
            else
                if (TOFh<TOF_min)
                    fprintf('\n\Ellipse-2A\n\n');
                    FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
sin(2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN2,a0,[],s,TOFh,c,mu);
                else
                    fprintf('\n\Ellipse-2B\n\n');
                    FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
c)/(2*a))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN3,a0,[],s,TOFh,c,mu);
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%
    %Type 1 parameteres
    %%%%%%%%%%%%%%%%%%%
    if TA(counter)<180
        if (TOFh<TOF_par)
            fprintf('\n\nType-1H\n\n');
            alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
            betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
            P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
            P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

            if P_plus>P_minus
                P=P_plus;
            else
                P=P_minus;
            end
        else
            if (TOFh<TOF_min)
                fprintf('\n\nType-1A\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                fprintf('\n\nType-1B\n\n')
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                if P_plus<P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            end
        end
    else
        if (TOFh<TOF_par)
            fprintf('\n\nType-2H\n\n');
            alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
            betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
            P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
            P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

            if P_plus<P_minus
                P=P_plus;
            else
                P=P_minus;
            end
        else
            if (TOFh<TOF_min)
                fprintf('\n\nType-2A\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus<P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                fprintf('\n\nType-2B\n\n');
                alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            end
        end
    end
    TA(counter)=TA(counter)*pi/180;

    %time of flight check against stk values
    tof_day(counter)=TOF;
    a_a(counter)=a;

    %%%%%%%%%%%%%%%%%%%%%%%%%
    %Transfer Characteristics
    %%%%%%%%%%%%%%%%%%%%%%%%%

    %Orbit
    r1_per=200+6378.14;
    r2_per=350+3397.00;
    e=sqrt(1-(P/a_a(counter)));
    vD=sqrt(2*((mu/r1)-(mu/(2*a_a(counter)))));
    vA=sqrt(2*((mu/r2)-(mu/(2*a_a(counter)))));
    rD=r1; rA=r2;

    %delta_V for inclination change
    theta_star_halfway=pi/2;
    r_halfway=a_a(counter)*(1-e);
    velocity_at_halfway=sqrt(2*((mu/r_halfway)-(mu/(2*a_a(counter)))));
    deltav_inc_change(counter)=2*velocity_at_halfway*sin(inclination/2)

    %Angle
```

```
        theta_star_D=acos((1/e)*((P/r1-1)));
        theta_star_A=acos((1/e)*((P/r2-1)));

        if theta_star_A-theta_star_D==TA(counter)
            theta_star_A_new=theta_star_A;
            theta_star_D_new=theta_star_D;
        elseif -theta_star_A-theta_star_D==TA(counter)
            theta_star_A_new=-theta_star_A;
            theta_star_D_new=theta_star_D;
        elseif -theta_star_A+theta_star_D==TA(counter)
            theta_star_A_new=-theta_star_A;
            theta_star_D_new=-theta_star_D;
        else
            theta_star_A_new=theta_star_A;
            theta_star_D_new=-theta_star_D;
        end

        theta_star_D;
        theta_star_A;

        gamma_D=acos(sqrt(mu*P)/(r1*vD))*sign(theta_star_D_new);
        gamma_A=acos(sqrt(mu*P)/(r2*vA))*sign(theta_star_A_new);

        %r, h, theta relationship
        vD_vector_1=[vD*sin(gamma_D) vD*cos(gamma_D) 0];
        vD_vector= vD_vector_1*trans_matrix';
        v_infinity_dep=vD_vector'-v1_vector;
        v_infinity_mag=norm(v_infinity_dep);
        delta_vD(counter)=sqrt(v_infinity_mag^2+((2*muE)/r1_per)) - sqrt(muE/r1_per);

        vA_vector_1=[vA*sin(gamma_A) vA*cos(gamma_A) 0];
        vA_vector= vA_vector_1*trans_matrix_M';
        v_infinity_arr=v2_vector-vA_vector';
        v_infinity_mag1=norm(v_infinity_arr);
        delta_vA(counter)=sqrt(v_infinity_mag1^2+((2*muM)/r2_per)) - sqrt(muM/r2_per) ;

        deltav_tot(counter)=abs(delta_vD(counter))+abs(delta_vA(counter))+abs(deltav_inc_change(counter));

        %Burn Time Analysis

        g=9.81;             %[m/s^2]
        Isp=1000;           %Assumed for now
        mdot=1.6;           %based on current engine [kg/s]
        mass_f=82543.31;    %Current MHV mass [kg]

        mass_prop_A(counter)=mass_f*exp(delta_vA(counter)*10^3/(Isp*g)) - mass_f;
        time_burn_A(counter)=mass_prop_A(counter)/mdot;

        mass_prop_inc(counter)=(mass_prop_A(counter) +
    mass_f)*exp(deltav_inc_change(counter)*10^3/(Isp*g)) - (mass_prop_A(counter) + mass_f);
        time_burn_inc(counter)=mass_prop_inc(counter)/mdot;

        mass_prop_D(counter)=(mass_prop_inc(counter) + mass_prop_A(counter) +
    mass_f)*exp(delta_vD(counter)*10^3/(Isp*g)) - (mass_prop_inc(counter) + mass_prop_A(counter) +
    mass_f);
        time_burn_D(counter)=mass_prop_D(counter)/mdot;

        mass_prop_total(counter)=mass_prop_A(counter)+mass_prop_inc(counter)+mass_prop_D(counter);
    end

    best_counter=find(deltav_tot==min(deltav_tot));

    out=[jdate, TA(best_counter), tof_day(best_counter), delta_vD(best_counter),
    delta_vA(best_counter),...
            deltav_tot(best_counter), a_a(counter),deltav_inc_change(counter), mass_prop_A(counter),...
            time_burn_A(counter),mass_prop_inc(counter),time_burn_inc(counter),mass_prop_D(counter),...
            time_burn_D(counter),mass_prop_total(counter)];
```

The following tables Table 31-1 though Table 31-7 give data for each trajectory about delta v's for each maneuver, as well as propellant mass, and burn times.

Table 31-1: Range of launch dates for year 2014

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Dec	19	2013	245	3.52	50225.00	31390.00	0.69	7885.80	4928.60	2.63	26053.00	16283.00	6.80	31390.00	179.628	1.8239E+08
	20	2013	246	3.52	49788.00	31118.00	0.66	7549.20	4718.20	2.63	25349.00	15843.00	6.81	82687.00	179.743	1.8267E+08
	22	2013	248	3.51	50010.00	31256.00	0.71	8130.70	5081.70	2.62	25614.00	16009.00	6.83	83755.00	179.995	1.8212E+08
	24	2013	249	3.51	50526.00	31579.00	0.76	8810.40	5506.50	2.63	26212.00	16383.00	6.85	85548.00	179.680	1.8147E+08
	26	2013	251	3.50	50328.00	31455.00	0.78	8955.80	5597.40	2.62	25904.00	16190.00	6.87	85188.00	179.955	1.8131E+08
	29	2013	253	3.49	50813.00	31758.00	0.83	9657.40	6035.80	2.62	26312.00	16445.00	6.90	86782.00	179.806	1.8061E+08
Jan	8	2014	260	3.48	50604.00	31627.00	0.92	10655.00	6659.40	2.60	25572.00	15982.00	6.97	86830.00	179.668	1.7951E+08
	18	2014	267	3.47	50412.00	31507.00	0.99	11390.00	7118.60	2.56	24870.00	15544.00	7.01	86672.00	179.783	1.7859E+08
	28	2014	274	3.45	50079.00	31299.00	1.03	11830.00	7393.80	2.52	24256.00	15160.00	7.00	86165.00	180.115	1.7790E+08
Feb	7	2014	280	3.45	50373.00	31483.00	1.05	11992.00	7494.80	2.48	24079.00	15049.00	6.98	86443.00	179.995	1.7728E+08
	17	2014	286	3.45	50858.00	31786.00	1.01	11555.00	7222.00	2.46	23838.00	14899.00	6.94	86251.00	180.006	1.7707E+08
	27	2014	292	3.45	49618.00	31011.00	0.97	11037.00	6898.10	2.45	23371.00	14607.00	6.88	84027.00	180.092	1.7736E+08
Mar	9	2014	302	3.49	49829.00	31143.00	0.87	9770.40	6106.50	2.44	23212.00	14508.00	6.81	82812.00	182.739	1.7787E+08
	19	2014	311	3.54	50183.00	31365.00	0.72	8061.40	5038.40	2.42	23002.00	14376.00	6.71	81246.00	184.699	1.7876E+08

Table 31-1: Data for departure year 2014

Table 31-2: Range of launch dates for year 2016

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Feb	5	2016	235	3.45	50419.00	31512.00	1.05	12021.00	7512.90	2.49	24209.00	15130.00	6.99	86649.00	180.01	177370000.00
	6	2016	236	3.45	51199.00	32000.00	1.05	12050.00	7531.30	2.49	24503.00	15315.00	6.99	87753.00	180.26	177140000.00
	8	2016	237	3.45	50879.00	31799.00	1.04	11995.00	7496.70	2.48	24286.00	15179.00	6.98	87160.00	180.12	177150000.00
	10	2016	238	3.45	50596.00	31623.00	1.04	11937.00	7460.60	2.48	24092.00	15058.00	6.97	86626.00	179.99	177160000.00
	13	2016	240	3.45	50248.00	31405.00	1.04	11846.00	7403.90	2.47	23847.00	14905.00	6.96	85942.00	180.12	177180000.00
	15	2016	241	3.45	50062.00	31289.00	1.03	11783.00	7364.40	2.46	23714.00	14822.00	6.95	85559.00	179.99	177200000.00
	25	2016	247	3.46	49762.00	31101.00	0.99	11234.00	7021.10	2.45	23417.00	14636.00	6.90	84413.00	180.07	177290000.00
Mar	6	2016	256	3.48	49898.00	31186.00	0.89	10067.00	6291.90	2.44	23249.00	14531.00	6.83	83214.00	182.09	177710000.00
	16	2016	265	3.52	50216.00	31385.00	0.75	8437.40	5273.40	2.43	23039.00	14400.00	6.74	81693.00	184.06	178510000.00
	26	2016	275	3.59	50643.00	31652.00	0.58	6459.90	4037.50	2.41	22765.00	14228.00	6.62	79868.00	186.54	179670000.00
Apr	5	2016	284	3.64	50858.00	31786.00	0.39	4267.50	2667.20	2.39	22406.00	14003.00	6.48	77240.00	188.19	181130000.00
	15	2016	294	3.72	51021.00	31888.00	0.19	1994.70	1246.70	2.35	21953.00	13721.00	6.32	74969.00	190.19	182830000.00
	25	2016	303	3.78	51354.00	32096.00	0.02	239.79	149.87	2.31	21398.00	13374.00	6.14	72992.00	191.32	184700000.00
May	5	2016	298	3.62	49748.00	31092.00	0.12	1321.40	825.87	2.42	21579.00	13487.00	6.04	72648.00	184.42	186220000.00

Table 31-2: Data for departure year 2016

Table 31-3: Range of launch dates for year 2018

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
May	15	2018	254	3.59	47554.00	29721.00	0.09	1011.90	632.42	2.51	24063.00	15039.00	6.19	72629.00	180.25	186510000.00
	16	2018	255	3.60	47600.00	29750.00	0.10	1119.60	699.73	2.49	23879.00	14925.00	6.19	72599.00	180.80	186610000.00
	18	2018	256	3.60	47890.00	29931.00	0.14	1557.00	973.10	2.51	24017.00	15010.00	6.24	73464.00	180.10	187040000.00
	20	2018	257	3.60	48090.00	30056.00	0.17	1880.10	1175.10	2.50	23916.00	14948.00	6.27	73886.00	179.81	187360000.00
	22	2018	259	3.61	48370.00	30231.00	0.21	2309.00	1443.10	2.49	23874.00	14921.00	6.31	74553.00	180.05	187780000.00
	28	2018	262	3.62	48739.00	30462.00	0.27	2938.10	1836.30	2.47	23608.00	14755.00	6.36	75286.00	180.38	188400000.00
Jun	4	2018	270	3.65	49717.00	31073.00	0.43	4741.20	2963.20	2.41	23032.00	14395.00	6.49	77490.00	180.26	190270000.00
	14	2018	279	3.67	50476.00	31548.00	0.59	6450.40	4031.50	2.33	22168.00	13855.00	6.59	79095.00	180.37	192170000.00
	24	2018	288	3.71	51274.00	32046.00	0.72	7929.10	4955.70	2.25	21260.00	13288.00	6.68	80464.00	180.22	193930000.00
Jul	4	2018	297	3.73	51750.00	32344.00	0.84	9145.00	5715.60	2.15	20247.00	12654.00	6.72	81142.00	179.82	195530000.00
	14	2018	308	3.75	52037.00	32523.00	0.94	10178.00	6361.20	2.05	19162.00	11976.00	6.73	81377.00	180.12	197060000.00
	24	2018	319	3.77	52328.00	32705.00	1.01	10881.00	6800.70	1.95	18136.00	11335.00	6.73	81346.00	180.18	198300000.00
Aug	3	2018	330	3.79	52338.00	32711.00	1.05	11272.00	7045.10	1.87	17312.00	10820.00	6.70	80922.00	180.05	199240000.00
	13	2018	342	3.79	52159.00	32599.00	1.06	11357.00	7098.20	1.80	16665.00	10415.00	6.66	80180.00	180.20	199870000.00

Table 31-3: Data for departure year 2018

Table 31-4: Range of launch dates for year 2020

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Jul	26	2020	278	3.77	52293.00	32683.00	1.02	11003.00	6876.70	1.93	17932.00	11208.00	6.72	81228.00	180.07	198550000.00
	27	2020	279	3.78	52322.00	32701.00	1.02	11047.00	6904.40	1.92	17857.00	11160.00	6.72	81225.00	180.01	198650000.00
	29	2020	281	3.78	52348.00	32718.00	1.03	11124.00	6952.40	1.90	17680.00	11050.00	6.72	81152.00	179.90	198830000.00
	31	2020	284	3.78	52360.00	32725.00	1.04	11204.00	7002.40	1.88	17469.00	10918.00	6.71	81033.00	180.22	199040000.00
Aug	2	2020	286	3.79	52386.00	32741.00	1.05	11259.00	7037.00	1.87	17342.00	10839.00	6.71	80987.00	180.10	199190000.00
	5	2020	289	3.79	52367.00	32729.00	1.06	11319.00	7074.20	1.85	17136.00	10710.00	6.70	80822.00	179.89	199400000.00
	15	2020	301	3.79	52151.00	32594.00	1.06	11334.00	7083.70	1.80	16580.00	10362.00	6.65	80065.00	180.01	199950000.00
	25	2020	313	3.80	51991.00	32494.00	1.04	11043.00	6901.70	1.77	16328.00	10205.00	6.61	79361.00	180.01	200120000.00
Sep	4	2020	325	3.80	51750.00	32344.00	0.99	10460.00	6537.80	1.78	16417.00	10261.00	6.56	78628.00	179.95	199910000.00
	14	2020	337	3.78	51262.00	32039.00	0.90	9596.70	5998.00	1.82	16850.00	10531.00	6.51	77709.00	179.88	199330000.00
	24	2020	350	3.77	50856.00	31785.00	0.79	8401.60	5251.00	1.90	17651.00	11032.00	6.46	76909.00	180.28	198350000.00
Oct	4	2020	362	3.75	50355.00	31472.00	0.65	6985.90	4366.20	2.00	18685.00	11678.00	6.40	76026.00	180.34	197050000.00
	14	2020	374	3.71	49598.00	30999.00	0.4963	5317.40	3323.40	2.12	19934.00	12459.00	6.33	74849.00	180.54	195450000.00
	24	2020	385	3.69	48938.00	30587.00	0.33	3508.30	2192.70	2.24	21168.00	13230.00	6.25	73615.00	180.47	193660000.00

Table 31-4: Data for departure year 2020

Table 31-5: Range of launch dates for year 2022

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Sep	4	2022	281	3.80	51733.00	32333.00	0.99	10492.00	6557.50	1.78	16404.00	10252.00	6.56	78628.00	180.01	199930000.00
	5	2022	282	3.80	51710.00	32319.00	0.98	10427.00	6516.60	1.78	16430.00	10269.00	6.56	78567.00	179.92	199890000.00
	7	2022	284	3.80	51655.00	32285.00	0.97	10289.00	6430.90	1.79	16497.00	10310.00	6.55	78441.00	179.73	199810000.00
	9	2022	287	3.80	51607.00	32255.00	0.95	10101.00	6313.30	1.79	16571.00	10357.00	6.54	78279.00	179.98	199690000.00
	11	2022	290	3.79	51504.00	32190.00	0.93	9901.20	6188.20	1.81	16685.00	10428.00	6.53	78090.00	180.22	199560000.00
	14	2022	293	3.79	51382.00	32114.00	0.91	9641.10	6025.70	1.82	16822.00	10514.00	6.52	77845.00	179.94	199360000.00
	24	2022	305	3.77	50835.00	31772.00	0.80	8515.10	5321.90	1.89	17557.00	10973.00	6.46	76907.00	179.90	198440000.00
Oct	4	2022	317	3.75	50397.00	31498.00	0.67	7121.20	4450.80	1.99	18546.00	11591.00	6.41	76064.00	179.94	197160000.00
	14	2022	328	3.72	49780.00	31112.00	0.52	5556.50	3472.80	2.10	19735.00	12334.00	6.35	75071.00	179.67	195650000.00
	24	2022	340	3.69	48934.00	30583.00	0.34	3682.40	2301.50	2.22	20957.00	13098.00	6.25	73573.00	180.05	193830000.00
Nov	3	2022	351	3.66	48178.00	30112.00	0.16	1717.00	1073.10	2.34	22227.00	13892.00	6.16	72123.00	180.18	191860000.00
	13	2022	361	3.63	47486.00	29679.00	0.03	272.69	170.43	2.44	23281.00	14550.00	6.09	71040.00	180.08	189850000.00
	23	2022	371	3.59	48222.00	30139.00	0.21	2361.60	1476.00	2.53	24247.00	15155.00	6.33	74831.00	180.26	187770000.00
Dec	3	2022	380	3.56	48964.00	30602.00	0.39	4390.60	2744.10	2.58	24839.00	24839.00	6.54	78194.00	180.22	185780000.00

Table 31-5: Data for departure year 2022

Table 31-6: Range of launch dates for year 2024

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Oct	4	2024	275	3.74	50194.00	31371.00	0.62	6651.90	4157.40	2.00	19390.00	12119.00	6.40	76235.00	180.37	196740000.00
	5	2024	276	3.74	50133.00	31333.00	0.61	6571.60	4107.30	2.01	19341.00	12088.00	6.39	76046.00	180.30	196660000.00
	7	2024	278	3.74	50055.00	31284.00	0.60	6410.90	4006.80	2.03	19249.00	12030.00	6.38	75714.00	180.14	196510000.00
	9	2024	280	3.74	50091.00	31307.00	0.51	5527.50	3454.70	2.05	20759.00	12974.00	6.37	76377.00	179.99	195620000.00
	11	2024	283	3.73	49892.00	31182.00	0.50	5349.90	3343.70	2.08	20628.00	12893.00	6.36	75870.00	180.32	195460000.00
	14	2024	286	3.72	49673.00	31046.00	0.47	5083.20	3177.00	2.11	20443.00	12777.00	6.34	75199.00	180.12	195220000.00
	24	2024	297	3.69	48896.00	30560.00	0.31	3330.40	2081.50	2.23	21555.00	13472.00	6.25	73782.00	180.03	193490000.00
Nov	3	2024	308	3.65	48116.00	30072.00	0.09	940.57	587.85	2.35	23711.00	14819.00	6.15	72767.00	180.18	191090000.00
	13	2024	315	3.63	47713.00	29821.00	0.05	581.25	363.28	2.47	23753.00	14846.00	6.10	72048.00	178.58	189550000.00
	23	2024	327	3.59	48496.00	30310.00	0.24	2690.10	1681.30	2.53	24721.00	15451.00	6.33	75907.00	179.77	187460000.00
Dec	3	2024	336	3.56	50007.00	31254.00	0.48	5424.10	3390.10	2.59	26603.00	16627.00	6.54	82034.00	179.73	184820000.00
	13	2024	345	3.54	50994.00	31871.00	0.65	7508.90	4693.10	2.61	27182.00	16989.00	6.71	85685.00	179.99	182790000.00
	23	2024	353	3.51	51888.00	32430.00	0.80	9382.20	5863.90	2.62	27510.00	17194.00	6.84	88781.00	180.01	180920000.00
Jan	2	2025	360	3.49	52811.00	33007.00	0.93	10911.00	6819.40	2.61	27584.00	17240.00	6.93	91306.00	179.73	179280000.00

Table 31-6: Data for departure year 2024

Table 31-7: Range of launch dates for year 2027

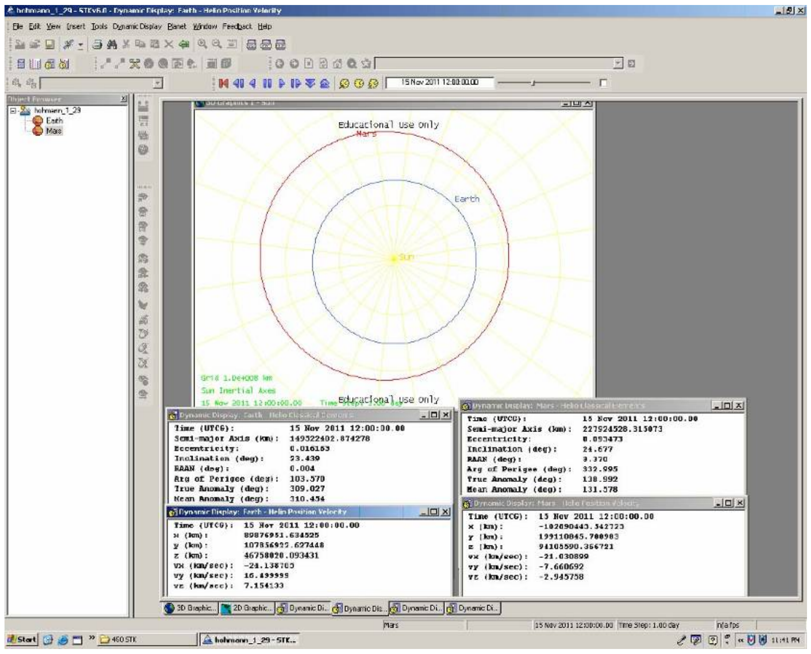
Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Nov	6	2026	266	3.64	48075.87	48075.87	0.11	1230.71	769.19	2.42	23112.24	14445.15	6.18	72418.81	179.66	191382456.80
	7	2026	267	3.64	48171.54	48171.54	0.10	1037.09	648.18	2.48	23702.01	14813.75	6.21	72910.64	179.64	191194922.05
	9	2026	270	3.63	48529.48	48529.48	0.05	541.43	338.39	2.62	25264.43	15790.27	6.30	74335.34	180.12	190768791.00
	11	2026	272	3.63	48920.29	48920.29	0.01	132.35	82.72	2.74	26602.41	16626.51	6.38	75655.05	180.10	190438716.49
	13	2026	274	3.63	49670.38	49670.38	0.03	289.52	180.95	2.87	28008.97	17505.61	6.52	77968.87	180.10	190128346.04
	16	2026	277	3.65	51176.35	51176.35	0.08	952.17	595.10	3.06	30228.82	18893.01	6.79	82357.34	180.13	189699028.82
	26	2026	287	3.75	57780.10	57780.10	0.28	3435.66	2147.29	3.74	38249.65	23906.03	7.76	99465.42	180.40	188571961.25
Dec	6	2026	296	3.91	66074.15	66074.15	0.46	6173.91	3858.69	4.36	46202.74	28876.71	8.73	118450.80	180.46	187855274.34
	16	2026	304	4.13	75930.54	75930.54	0.63	9028.34	5642.71	4.90	53503.12	33439.45	9.66	138462.00	180.26	187431645.36
	26	2026	312	4.39	87089.74	87089.74	0.79	11948.89	7468.06	5.35	59912.86	37445.54	10.53	158951.49	180.36	187301983.36
Jan	5	2027	319	4.65	98317.21	98317.21	0.92	14574.26	9108.92	5.69	64934.84	40584.27	11.27	177826.31	180.15	187368337.78
	15	2027	326	4.91	109066.36	109066.36	1.03	16798.97	10499.35	5.93	68502.05	42813.78	11.87	194367.38	180.18	187677920.59
	25	2027	333	5.17	118809.65	118809.65	1.11	18375.02	11484.38	6.05	70417.14	44010.71	12.33	207601.81	180.45	188232294.78
Feb	4	2027	339	5.40	126837.95	126837.95	1.15	19169.02	11980.64	6.09	71036.15	44397.59	12.65	217043.12	180.29	188882007.10

Table 31-7: Data for departure year 2027

31.3 MHV Hohmann Dates & ΔVs

Author: Kimberly Mrozek

To further examine the Hohmann transfer option for the Mars Habitat Vehicle (MHV), we created a model of Earth and Mars in Satellite Tool Kit (STK). This allowed us to obtain exact position data of Earth and Mars for particular dates. The STK model is shown below in Figure 31-2.



```
Dynamic Display: Earth - Helio Classical Elements
Time (UTCG):              15 Nov 2011 12:00:00.00
Semi-major Axis (km):     149522402.874278
Eccentricity:             0.016163
Inclination (deg):        23.439
RAAN (deg):               0.004
Arg of Perigee (deg):     103.570
True Anomaly (deg):       309.027
Mean Anomaly (deg):       310.454
```

```
Dynamic Display: Earth - Helio Position Velocity
Time (UTCG):     15 Nov 2011 12:00:00.00
x (km):          89876951.634525
y (km):          107856925.627448
z (km):          46758020.093431
vx (km/sec):     -24.138705
vy (km/sec):     16.499995
vz (km/sec):     7.154133
```

```
Dynamic Display: Mars - Helio Classical Elements
Time (UTCG):              15 Nov 2011 12:00:00.00
Semi-major Axis (km):     227924528.315073
Eccentricity:             0.093473
Inclination (deg):        24.677
RAAN (deg):               3.370
Arg of Perigee (deg):     332.995
True Anomaly (deg):       138.992
Mean Anomaly (deg):       131.578
```

```
Dynamic Display: Mars - Helio Position Velocity
Time (UTCG):     15 Nov 2011 12:00:00.00
x (km):          -102090443.342723
y (km):          199110845.700983
z (km):          94105590.366721
vx (km/sec):     -21.030899
vy (km/sec):     -7.660692
vz (km/sec):     -2.945758
```

Figure 31-2: STK model

An initial Google search found several dates that were claimed to be Hohmann transfer launch dates. These dates were found at Ref. [1]. We then took these dates, set the STK simulation to each arrival and departure date and found the corresponding position data for Mars and Earth.

In order to reconcile the differences between the true anomalies of Earth and Mars orbits, we looked up opposition dates for Mars to determine when they would be in the same radial line stretching from the Sun to Mars. These opposition dates were found at Ref. [2]. These dates can be found in Table 31-8 below, and the resulting average difference in true anomaly which I then used to reconcile the differences between the two planet orbits.

date	th* earth	th* mars	difference
13-Jun-01	158.18	286.39	128.21
28-Aug-03	233.86	358.44	124.58
7-Nov-05	302.32	68.58	126.26
28-Dec-07	353.55	117.93	124.38

29-Jan-10	26.76	153.21	126.45
3-Mar-12	60.1	187.04	126.94
			avg is 126.13

Table 31-8: Opposition dates and true anomaly data

Data on dates and times of flight for each Hohman transfer can be found below in Table 31-9.

Transfer	Date leave	Jdate	date arrive	Jdate	tof
1	Aug 13 2005	2453596	May 22 2006	2453878	282
2	Sep. 17 2007	2454361	June 23 2008	2454641	280
3	oct 16 2009	2455121	jul 15 2010	2455393	272
4	nov 15 2011	2455881	aug 1 2012	2456141	260
5	dec 19 2013	2456646	aug 22 2014	2456892	246
6	feb 5 2016	2457424	sep 29 2016	2457661	237
7	may 15 2018	2458254	jan 24 2019	2458508	254
8	jul 26 2020	2459057	may 2 2021	2459337	280
9	sep 4 2022	2459827	june13 2023	2460109	282
10	oct 4 2024	2460588	july 7 2025	2460864	276
11	nov 6 2026	2461351	jul 30 2027	2461617	266
12	dec 13 2028	2462119	aug 27 2029	2462376	257

Table 31-9: Dates, Julian dates, and times of flight for each Hohmann trajectory

Data for each Earth departure can be found below in Table 31-10.

	departures					
	th* (earth in earth)	th* (e in m)	pos x (km)	pox y (km)	pos z (km)	magnitude
1	218.682	344.812	117382762.7	87953629.64	38131009.51	151553712.70
2	252.529	18.66	149581690.8	14129374.15	6125414.015	150372344.94
3	282.962	49.272	137141391	53734331.06	23295185.42	149123455.99
4	309.027	75.337	89876951.63	107856922.6	46758020.09	147977345.01
5	344.384	110.69	6286215.124	134916592.8	58488991.6	147183442.21
6	32.295	158.605	105899247.3	94190308.58	40832543.14	147491563.79
7	128.95	255.26	88113418.93	112768165.4	48884471.65	151229379.73
8	202.267	328.57	84252794.25	116002250.5	50287378.71	151933787.94
9	238.733	5.04	143191026.6	43600071.71	18900912.66	150870410.78
10	271.03	37.34	146672386.5	27168831.97	11776605.55	149631623.79
11	303.059	69.36	107190547.3	94030006.76	40759721.66	148299731.99
12	340.319	106.62	21492869.52	133675220.1	57945010.71	147270608.70

Table 31-10: True anomaly (theta star) and position information from STK for each departure from earth

Data for each Mars arrival can be found below in Table 31-11.

	arrivals				
	th* (m in m)	pos x (km)	pox y (km)	pos z (km)	magnitude
1	167.081	198957291	133353136.8	66541395.28	248585639.23
2	198.138	246636330.8	20003105.7	15837408.48	247952470.10
3	227.533	220860621	90138077.61	35378762.96	241155434.95
4	257.281	137808446.2	169556936.9	74049046.98	230703238.06
5	291.677	8589663.273	198450249.5	90791725.93	218401971.89
6	340.981	152148054.2	126778398.7	62257389.33	207600036.85
7	78.405	128451151.5	165641161	72508383.55	221797561.34
8	148.477	139491627.7	182166049.8	87319022.5	245493371.81
9	185.957	236905158	67302195.16	37262024.62	249082512.11
10	216.08	238802516.1	49432398.95	16232267.74	244404767.28
11	247.13	170431288.8	148039151.2	63306724.82	234457151.52
12	284.876	34176948.55	198500937	90126764.84	220666081.52

Table 31-11: True anomaly (theta star) and position information from STK for each arrival at Mars

Data on the true anomaly differences between Earth departure and Mars arrival can be seen below in Table 31-12. Since a Hohmann Transfer is a 180 degree transfer, this table of true anomaly differences is one check that validates the Hohmann Transfer dates we originally found.

	th* diff
1	177.73
2	179.478
3	178.26
4	181.94
5	180.98
6	182.37
7	183.14
8	179.9
9	180.91
10	178.74
11	177.77
12	178.25

Table 31-12: Transfer angles for each trajectory

We then used the above information in MATLAB to calculate delta v values and times of flight for each trajectory. The process this code uses is identical to that in the Appendix section titled Initial MHV Trajectory Analysis, and will not be described here.

MATLAB code:

```
%Kimberly Mrozek
%Hohmann transfer calculations with distance numbers taken from STK

close all
clear all
clc

%distance between sun and earth on each departure date in kilometers
earthdist=[151553712.70 150372344.94 149123455.99 147977345.01 147183442.21 147491563.79 151229379.73
151933787.94 150870410.78 149631623.79 148299731.99 147270608.70];
%initial orbit is 200 km altitude, circular
r1=earthdist + 200;
%distance between sun and mars on each arrival date in kilometers
marsdist=[248585639.23 247952470.10 241155434.95 230703238.06 218401971.89 207600036.85 221797561.34
245493371.81 249082512.11 244404767.28 234457151.52 220666081.52];
%final orbit is 79796.04 km at apoapsis, elliptical, assume apoapsis is at far side of mars from sun
r2=marsdist + 79796.04;
ramars=79796.04; %(apoapsis of final orbit in km)
rpmars=350 %(periapsis of final orbit in km)
%a of final mars orbit (km)
amars=41771.52; %(km)
mumars=4.282828e4; %gravitational constant of mars (km^3/sec^2)
musun=1.3271244e11; %gravitational constant of sun (km^3/sec^2)

%velocity at first thrust point before maneuver (km/s)
v1=sqrt((musun./r1));

%transfer ellipse parameters
a=.5*(r1+r2);  %semi-major axis in kilometers
e=1-(r1./a);   %eccentricity

%velocity at first thrust point after maneuver (km/s)
vp=sqrt((2*musun./r1)-(musun./a));

%delta v at first thrust point (km/s)
deltav1=vp-v1

%velocity at 2nd thrust point before maneuver (km/s)
va=sqrt((2*musun./r2)-(musun./a));

%velocity at 2nd thrust point after maneuver (km/s)
v2=sqrt((2*mumars./rpmars)-(mumars./amars));

%delta v at second thrust point (km/s)
deltav2=abs(v2-va)

%total delta v (km/s)
totdeltav=deltav1 + abs(deltav2)

%time of flight check against stk values
tofsec=pi.*sqrt((a.^3)/musun); %(sec)
tofday=tofsec/(60*60*24)
```

This MATLAB code gave the following delta v and time of flight data for each trajectory which can be seen below in Table 31-13.

	Date Leave	Date Arrive	ΔV 1 (km/s)	ΔV 2 (km/s)	Tot. ΔV (km/s)	TOF (days)
4	Nov 15 2011	Aug 1 2012	3.11	5.59	8.70	260
5	Dec 19 2013	Aug 22 2014	2.79	6.50	9.30	246
6	Feb 5 2016	Sep 29 2016	2.44	7.43	9.87	237
7	May 15 2018	Jan 24 2019	2.68	6.41	9.09	254

8	Jul 26 2020	May 2 2021	3.30	4.71	8.01	280
9	Sep 4 2022	June13 2023	3.44	4.43	7.87	282
10	Oct 4 2024	Jul 7 2025	3.39	4.69	8.08	276

Table 31-13: Hohmann trajectory parameters

The TOF values that were computed by MATLAB were another check on the Hohmann transfer dates we originally found. These numbers do not match exactly (data can be found in Table 31-9 and Table 31-13), but they are close enough that they validate the departure and arrival dates.

References:

[1] http://uplink.space.com/showflat.php?Cat=&Board=missions&Number=25725&page=10&view=collapsed&sb=7&o=0&fpart=1&vc=1

[2] http://www.indiahams.com/astronomy/marsdate.htm

31.4 Non-free Return Trajectory

Author: Kimberly Mrozek

Contributors: Meredith Evans, Eric Gustafson

The final trajectory for the Mars Habitat vehicle (MHV) is an elliptical Lambert Arc Trajectory. This trajectory is determined using the same position data taken from Satellite Tool Kit (STK) that we used in the Hohmann Transfer Trajectory (see Initial MHV Trajectory Appendix for more details). From this position data, the shape of the transfer arc is determined using Lambert's Equation, Equation 31-10

$$\sqrt{\frac{\mu}{a^3}}(t_2 - t_1) = 2m\pi + \begin{cases} (\alpha_o - \sin\alpha_o) - (\beta_o - \sin\beta_o) & (1A) \\ 2\pi - (\alpha_o - \sin\alpha_o) - (\beta_o - \sin\beta_o) & (1B) \\ (\alpha_o - \sin\alpha_o) + (\beta_o - \sin\beta_o) & (2A) \\ 2\pi - (\alpha_o - \sin\alpha_o) + (\beta_o - \sin\beta_o) & (2B) \end{cases}$$

Equation 31-10

In these equations, c= the chord between the position vectors of Earth and Mars,

$s = semi - perimeter\ \frac{r_1 + r_2 + c}{2}$

$\alpha = 2\sin^{-1}\sqrt{\frac{s}{2a}}$

$\beta = 2\sin^{-1}\sqrt{\frac{s-c}{2a}}$

and a is the semi-major axis.

We created a MATLAB code to iterate on the value of a, and determine the type of the trajectory (1A, 1B, 2A, or 2B). Then with the value of semi-major axis, the semi-latus rectum iss calculated with Equation 31-11.

$$p = \frac{4a(s - r_1)(s - r_2)}{c^2}\sin^2\left(\frac{\alpha \pm \beta}{2}\right)$$

Equation 31-11

The eccentricity of the arc is calculated using Equation 31-12.

$$e = \sqrt{1 - \frac{p}{a}}$$

Equation 31-12

The velocities are calculated with the elliptical velocity equation (Equation 31-13), where μ is the gravitational parameter, and r is the position vector magnitude.

$$v = \sqrt{\frac{2\mu_s}{r} - \frac{\mu_s}{a}}$$

Equation 31-13

The inclination delta v is determined with Equation 31-14, where v is the velocity at the point of the inclination change maneuver, and i is the inclination angle.

$$\Delta v_i = 2v\sin\left(\frac{i}{2}\right)$$

Equation 31-14

The delta v values at Earth parking orbit departure and Mars parking orbit arrival are calculated with the Law of Cosines, where a and b are two sides of the velocity triangle, C is the included angle between them, and c is the delta v (Equation 31-15).

$$c^2 = a^2 + b^2 - 2ab\cos C$$

Equation 31-15

Assumptions about the MHV and its engine were that the MHV has a dry mass of 76862 kg, the main engine has a specific impulse of 1007 seconds, and a mass flow rate of 2.13 kg per second.

Below is the main MATLAB code (main.m) created to output numbers generated by the code orbit3_minimize.m. Main.m takes an input of a Julian date, and outputs transfer angle, time of flight, Δv_1, Δv_2, total Δv, semi-major axis, Δv_i, masses of propellant required for each burn, and burn times for each burn.

```
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This is an improved version of the code that calculates the
%optimal delta V (and TOF) for a given range of launch dates
%for the HAB trajectory.  The optimization process was improved so that the
%code now runs much faster.
%This code calls on "orbit3" and runs for a specified range of julian
%dates.
clear all; close all; clc; warning off;

tic
```

```
format long
counter=0;
jdate_vector = 2461353+[0,1,3,5,7,10:10:90];
%jdate_vector = 2457300:10:2457600;
for jdate= jdate_vector
    fprintf('%5.2f%% done...\n', (jdate-jdate_vector(1))/(jdate_vector(end)-jdate_vector(1))*100);
    counter=counter+1;
    best_TOF = fminbnd('orbit3_minimize', 100, 400, optimset('TolX', .5), jdate);
    output=orbit3(best_TOF, jdate);
    %     [jdateofcounter, TA, tof_day, delta_v1, delta_v2, deltav_tot]=output
    jdateofcounter(counter)=output(1);
    TA(counter)=output(2);
    tof_day(counter)=output(3);
    delta_v1(counter)=output(4);
    delta_v2(counter)=output(5);
    deltav_tot(counter)=output(6);
    a_a(counter)=output(7);
    deltav_inc_change(counter)=output(8);
    mass_prop_A(counter)=output(9);
    time_burn_A(counter)=output(10);
    mass_prop_inc(counter)=output(11);
    time_burn_inc(counter)=output(12);
    mass_prop_D(counter)=output(13);
    time_burn_D(counter)=output(14);
    mass_prop_total(counter)=output(15);
    v_infinity_D(counter)=output(16);
    v_infinity_A(counter)=output(17);
end

  TA=TA';
  tof_day=tof_day';
  delta_v1=delta_v1';
  delta_v2=delta_v2';
  deltav_tot=deltav_tot';
  a_a=a_a';
  deltav_inc_change=deltav_inc_change';
  mass_prop_A = mass_prop_A';
  time_burn_A = time_burn_A';
  mass_prop_inc = mass_prop_inc';
  time_burn_inc = time_burn_inc';
  mass_prop_D = mass_prop_D';
  time_burn_D = time_burn_D';
  mass_prop_total = mass_prop_total';
  v_infinity_D = v_infinity_D';
  v_infinity_A = v_infinity_A';

%%%%%%%%%
%Output
%%%%%%%%%
dataoutput=[delta_v1 mass_prop_D time_burn_D deltav_inc_change mass_prop_inc...
        time_burn_inc delta_v2 mass_prop_A time_burn_A deltav_tot mass_prop_total v_infinity_D
v_infinity_A TA a_a tof_day];

save(['window_', num2str(jdateofcounter(1)), '.txt'],'dataoutput','-ASCII')

toc

subplot(2,1,1); plot(jdateofcounter-jdateofcounter(1), tof_day); grid on
ylabel('TOF (day)')
title(num2str(jdateofcounter(1)))
subplot(2,1,2); plot(jdateofcounter-jdateofcounter(1), deltav_tot); grid on
ylabel('Total delta V')
xlabel('Julian Date (Days after first date)')
```

The code orbit3_minimize.m takes the input of a time of flight, and optimizes through the equations stated above to calculate the data that main.m outputs.

```
%Meredith Evans, Kimberly Mrozek, Eric Gustafson
%This code calls on "planet data" and is used to find the best delta_V and TOF
%for a specified Julian date using a built in MATLAB optimizer, fminbnd.
```

```
function out=orbit3_minimize(TOF, jdate)

counter=0;
%for TOF=180:200
    counter=counter+1;
    [r1_vector, v1_vector, theta_starE, r2_vector_junk, v2_vector_junk, theta_star_junk]
=planet_data(jdate);
    [RE_vector_junk, VE_vector_junk, theta_star_junk, r2_vector, v2_vector, theta_starM]
=planet_data(jdate+TOF);
    r2_vector_actual=r2_vector;
    r1_vector(3)=0;
    r2_vector(3)=0;
    v1_vector(3)=0;
    v2_vector(3)=0;
    r1=norm(r1_vector); r2=norm(r2_vector);
    v1=norm(v1_vector); v2=norm(v2_vector);
    c_vector=r2_vector - r1_vector;
    c=norm(c_vector);
    s=0.5*(c+r1+r2);

    r1_hat=r1_vector/r1;
    r2_hat=r2_vector/r2;
    h=cross(r1_vector,r2_vector);
    del=acos((dot(r1_vector,r2_vector))/(r1*r2));
    if h(3)<0
        h=-h;
        TA(counter)=(2*pi)-del;
    else
        h=h;
        TA(counter)=del;
    end
    TA(counter);
    muE=398600.485043; mu=1.3271244E+11; muM=4.28282868534e+04;
    h_mag=norm(h);
    h_hat=h/h_mag;

    %Earth transformation matrix
    theta_hat=cross(h_hat,r1_hat);
    trans_matrix=[r1_hat(1) theta_hat(1) h_hat(1);r1_hat(2) theta_hat(2) h_hat(2);r1_hat(3)
theta_hat(3) h_hat(3)];
    inc=acos(h_hat(3));
    omega1=acos(h_hat(2)/-sin(inc));
    omega2=asin(h_hat(1)/sin(inc));
    theta1=acos(theta_hat(3)/sin(inc));
    theta2=asin(r1_hat(3)/sin(inc));

    %Mars transformation matrix
    theta_hat_M=cross(h_hat,r2_hat);
    trans_matrix_M=[r2_hat(1) theta_hat_M(1) h_hat(1);r2_hat(2) theta_hat_M(2) h_hat(2);r2_hat(3)
theta_hat_M(3) h_hat(3)];

    %inclination change
    r2_hat_actual=r2_vector_actual/norm(r2_vector_actual);
    r1_hat_new=cross(r2_hat_actual,h_hat);
    h_new=cross(r1_hat_new,r2_hat_actual);
    h_new_hat=h_new/norm(h_new);
    inclination=acos(h_new_hat(3));
    incli=inclination*180/pi;

    %%%%%%%%%%%%%%%%%%%%%
    %Minimum Conditions
    %%%%%%%%%%%%%%%%%%%%%

    a_min=0.5*s;                            %Minimum semi-major axis [Km]
    energy_min=-mu/(2*a_min);               %Energy associated with a_min [Km^2/sec^2]
    alpha_min=pi;                           %[rad]
    beta_min=2*asin(sqrt((s-c)/(2*a_min))); %[rad]
    P_min=(4*a_min*(s-r1)*(s-r2)*(sin(0.5*(alpha_min+beta_min)))^2)/c^2;
    e_min=sqrt(1-P_min/a_min);               %Eccentricity associated with a_min
    TOF_min=((a_min^1.5)/sqrt(mu))*((alpha_min-beta_min)-(sin(alpha_min)-sin(beta_min)));
```

```
        %%%%%%%%%%%%
        %Finding "a"
        %%%%%%%%%%%%
        TA(counter)=TA(counter)*180/pi;

        if TA(counter)<180
            TOF_par=(sqrt(2/mu)*(s^1.5-(s-c)^1.5))/3;   %Type 1
        else
            TOF_par=(sqrt(2/mu)*(s^1.5+(s-c)^1.5))/3;   %Type 2
        end

        TOF_Vector=24*3600*[TOF];
        a0=a_min;

        for j=1:length(TOF_Vector),
            TOFh=TOF_Vector(j);
            if TA(counter)<180
                if (TOFh<TOF_par)
                    %fprintf('\n\nHyperbola-1H\n\n');
                    a0h=-a0;
                    FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
    (2*asinh(sqrt(s/(2*abs(a))))))-(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
    c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN,a0h,optimset('Display','off'),s,TOFh,c,mu);
                else
                    if (TOFh<TOF_min)
                    %    fprintf('\n\Ellipse-1A\n\n');
                        FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
    sin(2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN2,a0,optimset('Display','off'),s,TOFh,c,mu);
                    else
                    %    fprintf('\n\Ellipse-1B\n\n');
                        FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
    2*asin(sqrt(s/(2*a)))))-(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN3,a0,optimset('Display','off'),s,TOFh,c,mu);
                    end
                end
            else
                if (TOFh<TOF_par)
                    %fprintf('\n\nHyperbola-2H\n\n');
                    a0h=-a0;
                    FUN=inline('(sqrt(mu)*TOFh)-(abs(a)^1.5)*((sinh(2*asinh(sqrt(s/(2*abs(a)))))-
    (2*asinh(sqrt(s/(2*abs(a))))))+(sinh(2*asinh(sqrt((s-c)/(2*abs(a)))))-(2*asinh(sqrt((s-
    c)/(2*abs(a)))))))','a','s','TOFh','c','mu');
                    a(j)=fsolve(FUN,a0h,optimset('Display','off'),s,TOFh,c,mu);
                else
                    if (TOFh<TOF_min)
                        %fprintf('\n\Ellipse-2A\n\n');
                        FUN2=inline('(sqrt(mu)*TOFh)-(a^1.5)*((2*asin(sqrt(s/(2*a)))-
    sin(2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN2,a0,optimset('Display','off'),s,TOFh,c,mu);
                    else
                        %fprintf('\n\Ellipse-2B\n\n');
                        FUN3=inline('(sqrt(mu)*TOFh)-(a^1.5)*(((2*pi)-2*asin(sqrt(s/(2*a)))-sin((2*pi)-
    2*asin(sqrt(s/(2*a)))))+(2*asin(sqrt((s-c)/(2*a)))-sin(2*asin(sqrt((s-
    c)/(2*a))))))','a','s','TOFh','c','mu');
                        a(j)=fsolve(FUN3,a0,optimset('Display','off'),s,TOFh,c,mu);
                    end
                end
            end
        end

        %%%%%%%%%%%%%%%%%%%
        %Type 1 parameteres
        %%%%%%%%%%%%%%%%%%%
        if TA(counter)<180
            if (TOFh<TOF_par)
                %fprintf('\n\nType-1H\n\n');
                alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus>P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                if (TOFh<TOF_min)
                    %fprintf('\n\nType-1A\n\n');
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                    P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                    if P_plus>P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                else
                    %fprintf('\n\nType-1B\n\n')
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                    P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                    if P_plus<P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                end
            end
        else
            if (TOFh<TOF_par)
                %fprintf('\n\nType-2H\n\n');
                alphaa=2*asin(sqrt(s/(2*abs(a)))); %[rad]
                betaa=2*asin(sqrt((s-c)/(2*abs(a)))); %[rad]
                P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                P_minus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                if P_plus<P_minus
                    P=P_plus;
                else
                    P=P_minus;
                end
            else
                if (TOFh<TOF_min)
                    %fprintf('\n\nType-2A\n\n');
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                    P_plus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]

                    if P_plus<P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                else
                    %fprintf('\n\nType-2B\n\n');
                    alphaa=2*asin(sqrt(s/(2*a))); %[rad]
                    betaa=2*asin(sqrt((s-c)/(2*a))); %[rad]
                    P_minus=(4*a*(s-r1)*(s-r2)*(sin(0.5*(alphaa-betaa)))^2)/c^2; %[Km]
                    P_plus=(4*abs(a)*(s-r1)*(s-r2)*(sin(0.5*(alphaa+betaa)))^2)/c^2;  %[Km]

                    if P_plus>P_minus
                        P=P_plus;
                    else
                        P=P_minus;
                    end
                end
            end
        end
        TA(counter)=TA(counter)*pi/180;

        %time of flight check against stk values
        tof_day(counter)=TOF;
        a_a(counter)=a;

        %%%%%%%%%%%%%%%%%%%%%%%%%
        %Transfer Characteristics
        %%%%%%%%%%%%%%%%%%%%%%%%%

        %Orbit
        r1_per=200+6378.14;
        r2_per=500+3397.00;
        e=sqrt(1-(P/a_a(counter)));
        vD=sqrt(2*((mu/r1)-(mu/(2*a_a(counter)))));
        vA=sqrt(2*((mu/r2)-(mu/(2*a_a(counter)))));
        rD=r1; rA=r2;

        %delta_V for inclination change
        theta_star_halfway=pi/2;
        r_halfway=a_a(counter)*(1-e);
        velocity_at_halfway=sqrt(2*((mu/r_halfway)-(mu/(2*a_a(counter)))));
        deltav_inc_change(counter)=2*velocity_at_halfway*sin(inclination/2);

        %Angle
        theta_star_D=acos((1/e)*((P/r1-1)));
        theta_star_A=acos((1/e)*((P/r2-1)));

        if theta_star_A-theta_star_D==TA(counter)
            theta_star_A_new=theta_star_A;
            theta_star_D_new=theta_star_D;
        elseif -theta_star_A-theta_star_D==TA(counter)
            theta_star_A_new=-theta_star_A;
            theta_star_D_new=theta_star_D;
        elseif -theta_star_A+theta_star_D==TA(counter)
            theta_star_A_new=-theta_star_A;
            theta_star_D_new=-theta_star_D;
        else
            theta_star_A_new=theta_star_A;
            theta_star_D_new=-theta_star_D;
        end

        theta_star_D;
        theta_star_A;

        gamma_D=acos(sqrt(mu*P)/(r1*vD))*sign(theta_star_D_new);
        gamma_A=acos(sqrt(mu*P)/(r2*vA))*sign(theta_star_A_new);

        %r, h, theta relationship
        vD_vector_1=[vD*sin(gamma_D) vD*cos(gamma_D) 0];
        vD_vector= vD_vector_1*trans_matrix';
        v_infinity_dep=vD_vector'-v1_vector;
        v_infinity_mag(counter)=norm(v_infinity_dep);
        delta_vD(counter)=sqrt(v_infinity_mag(counter)^2+((2*muE)/r1_per)) - sqrt(muE/r1_per);
        v_infinity_D(counter)=v_infinity_mag(counter);

        vA_vector_1=[vA*sin(gamma_A) vA*cos(gamma_A) 0];
        vA_vector= vA_vector_1*trans_matrix_M';
        v_infinity_arr=v2_vector-vA_vector';
        v_infinity_mag1(counter)=norm(v_infinity_arr);
        delta_vA(counter)=sqrt(v_infinity_mag1(counter)^2+((2*muM)/r2_per)) - sqrt(muM/r2_per);
        v_infinity_A(counter)=v_infinity_mag1(counter);

        %TOTAL
```

```
    deltav_tot(counter)=abs(delta_vD(counter))+abs(delta_vA(counter))+abs(deltav_inc_change(counter));

    %Burn Time Analysis

    g=9.81;             %[m/s^2]
    Isp=1007.1;           %Assumed for now
    mdot=2.13;           %based on current engine [kg/s]
    mass_f=76862.21805; %Current MHV mass [kg]

    mass_prop_A(counter)=mass_f*exp(delta_vA(counter)*10^3/(Isp*g)) - mass_f;
    time_burn_A(counter)=mass_prop_A(counter)/mdot;

    mass_prop_inc(counter)=(mass_prop_A(counter) +
mass_f)*exp(deltav_inc_change(counter)*10^3/(Isp*g)) - (mass_prop_A(counter) + mass_f);
    time_burn_inc(counter)=mass_prop_inc(counter)/mdot;

    mass_prop_D(counter)=(mass_prop_inc(counter) + mass_prop_A(counter) +
mass_f)*exp(delta_vD(counter)*10^3/(Isp*g)) - (mass_prop_inc(counter) + mass_prop_A(counter) +
mass_f);
    time_burn_D(counter)=mass_prop_D(counter)/mdot;

    mass_prop_total(counter)=mass_prop_A(counter)+mass_prop_inc(counter)+mass_prop_D(counter);
%end

%best_counter=find(deltav_tot==min(deltav_tot));

out= deltav_tot;
```

The code planet_data.m is the code that is used to output position and velocity vectors, and also true anomaly for Earth and Mars position vectors for a given Julian date. Orbit3_minimize calls upon this code. The position data this program calls upon is generated by a STK model of the orbits of Earth and Mars.

```
%This function outputs the heliocentric position and velocity and the true anomaly
% of Earth and Mars for a given Julian Date.
%Positions are in km, and velocities are in km/s, theta_stars are in radians
%
%The planetary data is taken from STK in the Sun Centered Mean Ecliptic of J2000 frame
%All output vectors are column vectors
%This function needs the binary data files created by running "make_binary_data" - earth_data.mat and
mars_data.mat
%(Using the binary files just makes the function run a bit faster)
%
%example use: [pos_earth, vel_earth, thst_earth, pos_mars, vel_mars, thst_mars] = planet_data(2456659)
% will return the data for noon on January 1st, 2014
%
%Eric Gustafson - 2/13/2005

function [pos_earth, vel_earth, thst_earth, pos_mars, vel_mars, thst_mars] = planet_data(JD_interp)

if JD_interp<2451758 | JD_interp>2462867
    disp 'The JD being input to planet_data is out of range!'
    return
end

load earth_data
load mars_data

%variables loaded into workspace are:
%JD ,x_earth, y_earth, z_earth, x_vel_earth, y_vel_earth, z_vel_earth,
%x_mars, y_mars, z_mars, x_vel_mars, y_vel_mars, z_vel_mars

%store all the STK data in an array, that will be used to interpolate
pos_earth_vector = [x_earth, y_earth, z_earth];
vel_earth_vector = [x_vel_earth, y_vel_earth, z_vel_earth];
pos_mars_vector = [x_mars, y_mars, z_mars];
vel_mars_vector = [x_vel_mars, y_vel_mars, z_vel_mars];
%interpolate to find requested Julian date
pos_earth = interp1(JD, pos_earth_vector, JD_interp)';
vel_earth = interp1(JD, vel_earth_vector, JD_interp)';
pos_mars = interp1(JD, pos_mars_vector, JD_interp)';
```

```
vel_mars = interp1(JD, vel_mars_vector, JD_interp)';

a_earth = 1.49597807e8; %km
e_earth = .01670545;
p_earth = a_earth*(1-e_earth^2);
thst_earth = 180/pi*acos(1/e_earth*(p_earth/norm(pos_earth) - 1)) * sign(dot(pos_earth,vel_earth)) ;
%                                                             ---------- quad check --------

a_mars = 2.27936636e8; %km
e_mars = .09341233;
p_mars = a_mars*(1-e_mars^2);
thst_mars = 180/pi*acos(1/e_mars*(p_mars/norm(pos_mars) - 1)) * sign(dot(pos_mars,vel_mars)) ;
%                                                         -------- quad check --------
```

Below in Table 31-14 through Table 31-19 are the resulting outputs for each trajectory window.

Table 31-14: Range of departure dates for year 2016

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Feb	5	2016	327	3.63	42244.14	19832.93	0.00	0.04	0.02	2.10	18239.32	8563.06	5.74	60483.50	3.71	1.8915E+08
	6	2016	326	3.63	42173.72	19799.87	0.00	4.34	2.04	2.10	18222.03	8554.94	5.73	60400.09	3.69	1.8915E+08
	8	2016	324	3.63	42265.62	19843.02	0.00	5.78	2.71	2.10	18190.53	8540.16	5.73	60461.93	3.66	1.8916E+08
	10	2016	322	3.65	42425.74	19918.19	0.00	5.86	2.75	2.10	18169.44	8530.26	5.74	60601.04	3.63	1.8915E+08
	13	2016	320	3.66	42627.96	20013.13	0.00	0.39	0.18	2.10	18158.27	8525.01	5.76	60786.62	3.59	1.8912E+08
	15	2016	317	3.69	43003.41	20189.40	0.00	5.70	2.67	2.09	18153.13	8522.60	5.78	61162.25	3.54	1.8908E+08
	25	2016	307	3.81	44797.97	21031.91	0.00	10.17	4.78	2.11	18270.86	8577.87	5.92	63079.00	3.37	1.8877E+08
Mar	6	2016	314	3.85	46003.74	21598.00	0.18	1735.78	814.92	2.06	17794.57	8354.26	6.09	65534.09	3.37	1.9104E+08
	16	2016	321	3.92	47532.84	22315.89	0.34	3332.28	1564.45	2.03	17523.71	8227.09	6.29	68388.83	3.36	1.9322E+08
	26	2016	329	3.98	49245.54	23119.97	0.50	4860.88	2282.10	2.02	17479.94	8206.54	6.50	71586.36	3.35	1.9543E+08
Apr	5	2016	335	4.05	51099.01	23990.14	0.63	6205.23	2913.25	2.05	17753.78	8335.11	6.73	75058.01	3.32	1.9742E+08
	15	2016	340	4.14	53519.99	25126.76	0.74	7447.47	3496.47	2.12	18399.44	8638.23	7.01	79366.91	3.29	1.9928E+08
	25	2016	347	4.25	56421.54	26488.99	0.85	8624.35	4048.99	2.23	19496.18	9153.13	7.33	84542.07	3.26	2.0109E+08
May	5	2016	353	4.36	59730.12	28042.31	0.94	9727.78	4567.03	2.40	21134.21	9922.16	7.69	90592.11	3.22	2.0277E+08

Table 31-14 : Data for 2016 departure

Table 31-15: Range of departure dates for year 2018

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
May	15	2018	246	3.58	42913.33	20147.10	0.00	3.04	1.43	2.43	21410.44	10051.85	6.01	64326.82	3.07	1.8565E+08
	16	2018	245	3.58	42943.48	20161.26	0.00	12.91	6.06	2.42	21366.85	10031.38	6.01	64323.23	3.06	1.8566E+08
	18	2018	243	3.59	43028.61	20201.22	0.00	3.04	1.43	2.42	21314.90	10007.00	6.01	64346.55	3.02	1.8569E+08
	20	2018	241	3.60	43145.16	20255.94	0.00	0.45	0.21	2.41	21260.99	9981.69	6.01	64406.60	2.99	1.8569E+08
	22	2018	239	3.61	43300.29	20328.78	0.00	13.77	6.46	2.41	21212.79	9959.06	6.02	64526.85	2.95	1.8567E+08
	28	2018	236	3.63	43585.53	20462.69	0.00	11.31	5.31	2.40	21178.26	9942.85	6.04	64775.10	2.90	1.8564E+08
Jun	4	2018	226	3.73	44969.58	21112.48	0.00	7.89	3.70	2.41	21258.02	9980.29	6.14	66235.50	2.74	1.8527E+08
	14	2018	216	3.86	47124.32	22124.09	0.00	0.79	0.37	2.46	21682.61	10179.63	6.32	68807.71	2.57	1.8448E+08
	24	2018	295	3.77	46651.10	21901.92	0.77	7488.68	3515.81	1.87	16029.42	7525.55	6.41	70169.20	3.22	1.9528E+08
Jul	4	2018	298	3.84	47908.16	22492.09	0.84	8248.25	3872.42	1.83	15627.84	7337.02	6.52	71784.24	3.16	1.9645E+08
	14	2018	299	3.94	49624.91	23298.08	0.90	8865.99	4162.44	1.83	15622.67	7334.59	6.67	74113.57	3.08	1.9719E+08
	24	2018	300	4.08	52322.50	24564.56	0.96	9480.31	4450.85	1.87	16019.15	7520.73	6.91	77821.96	3.01	1.9773E+08
Aug	3	2018	303	4.26	56010.00	26295.78	1.01	10143.85	4762.37	1.97	16957.51	7961.27	7.24	83111.36	2.94	1.9822E+08
	13	2018	305	4.47	60794.55	28542.04	1.06	10849.35	5093.59	2.14	18609.33	8736.77	7.67	90253.23	2.87	1.9861E+08

Table 31-15 : Data for 2018 departure

Table 31-16: Range of departure dates for year 2020

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Jul	26	2020	212	3.75	46018.96	21605.15	0.65	6381.71	2996.11	1.90	16295.83	7650.63	6.31	68696.50	2.68	1.9740E+08
	27	2020	211	3.75	46004.58	21598.40	0.66	6444.42	3025.55	1.90	16255.27	7631.58	6.31	68704.27	2.67	1.9740E+08
	29	2020	210	3.75	46023.55	21607.30	0.66	6457.31	3031.60	1.90	16276.67	7641.63	6.31	68757.53	2.64	1.9747E+08
	31	2020	209	3.75	46034.92	21612.64	0.67	6525.45	3063.59	1.89	16251.34	7629.74	6.32	68811.71	2.62	1.9747E+08
Aug	2	2020	207	3.75	46073.98	21630.98	0.68	6592.43	3095.04	1.89	16230.50	7619.95	6.32	68896.91	2.59	1.9746E+08
	5	2020	206	3.76	46206.53	21693.21	0.68	6673.61	3133.15	1.89	16223.62	7616.72	6.34	69103.76	2.55	1.9741E+08
	15	2020	200	3.82	47224.42	22171.09	0.72	7034.88	3302.76	1.89	16205.06	7608.01	6.43	70464.36	2.42	1.9680E+08
	25	2020	195	3.94	49347.65	23167.91	0.75	7393.97	3471.35	1.91	16382.31	7691.23	6.60	73123.94	2.29	1.9556E+08
Sep	4	2020	281	3.95	50551.34	23733.02	1.08	10625.25	4988.38	1.80	15379.13	7220.25	6.83	76555.71	2.90	1.9887E+08
	14	2020	275	4.17	54440.99	25559.15	1.09	10857.27	5097.31	1.87	16033.63	7527.52	7.13	81331.89	2.76	1.9774E+08
	24	2020	270	4.48	60419.88	28366.14	1.11	11203.52	5259.87	2.00	17223.82	8086.30	7.59	88847.22	2.62	1.9632E+08
Oct	4	2020	266	4.92	69458.12	32609.45	1.14	11694.39	5490.32	2.20	19195.47	9011.96	8.25	100347.98	2.50	1.9459E+08
	14	2020	266	5.46	82454.05	38710.82	1.15	12334.03	5790.62	2.55	22625.94	10622.51	9.16	117414.01	2.41	1.9305E+08
	24	2020	272	6.02	99436.07	46683.60	1.15	13037.70	6120.99	3.12	28490.92	13376.02	10.29	140964.68	2.36	1.9219E+08

Table 31-16 : Data for 2020 departures

Table 31-17: Range of departure dates for year 2022

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Sep	4	2022	287	3.79	47377.77	22243.09	0.98	9611.32	4512.36	1.76	14979.07	7032.43	6.53	71968.16	3.22	1.9994E+08
	5	2022	286	3.79	47384.41	22246.20	0.98	9594.86	4504.63	1.76	14993.95	7039.41	6.53	71973.22	3.21	1.9994E+08
	7	2022	284	3.79	47443.13	22273.77	0.98	9587.43	4501.14	1.76	15000.74	7042.60	6.53	72031.30	3.18	1.9992E+08
	9	2022	282	3.80	47551.66	22324.72	0.98	9594.99	4504.69	1.76	14999.56	7042.05	6.54	72146.21	3.14	1.9989E+08
	11	2022	280	3.79	47471.11	22286.91	0.98	9596.92	4505.59	1.76	15012.56	7048.15	6.54	72080.58	3.11	1.9984E+08
	14	2022	278	3.79	47422.88	22264.26	0.98	9606.56	4510.12	1.77	15040.64	7061.33	6.54	72070.07	3.06	1.9974E+08
	24	2022	269	3.82	48042.91	22555.36	0.98	9654.89	4532.81	1.79	15269.35	7168.71	6.59	72967.15	2.90	1.9910E+08
Oct	4	2022	259	3.96	50460.72	23690.48	1.00	9808.00	4604.69	1.83	15679.25	7361.15	6.79	75947.96	2.73	1.9795E+08
	14	2022	249	4.25	55484.00	26048.83	1.01	10055.99	4721.12	1.91	16392.23	7695.88	7.17	81932.22	2.55	1.9620E+08
	24	2022	239	4.69	63710.24	29910.91	1.03	10412.40	4888.45	2.04	17583.98	8255.39	7.76	91706.62	2.39	1.9382E+08
Nov	3	2022	231	5.31	76455.79	35894.74	1.06	10910.18	5122.15	2.24	19552.33	9179.50	8.61	106918.30	2.23	1.9080E+08
	13	2022	227	6.13	95820.83	44986.30	1.08	11535.44	5415.70	2.58	22938.78	10769.38	9.80	130295.05	2.09	1.8738E+08
	23	2022	228	7.06	123056.64	57773.07	1.08	12178.48	5717.60	3.16	29016.48	13622.76	11.29	164251.60	2.00	1.8418E+08
Dec	3	2022	342	5.96	127663.18	59935.76	0.00	2.90	1.36	6.88	77336.12	36308.04	12.84	205002.19	2.84	1.9631E+08

Table 31-17 : Data for 2022 departures

Table 31-18: Range of departure dates for year 2024

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Oct	4	2024	280	3.73	46039.31	21614.70	0.64	6316.44	2965.46	1.99	17124.24	8039.55	6.36	69479.99	3.21	1.9698E+08
	5	2024	279	3.73	46053.42	21621.32	0.64	6302.60	2958.97	1.99	17150.55	8051.90	6.36	69506.56	3.20	1.9696E+08
	7	2024	277	3.74	46115.07	21650.27	0.64	6296.96	2956.32	1.99	17148.96	8051.15	6.37	69560.99	3.16	1.9693E+08
	9	2024	275	3.74	46158.31	21670.57	0.64	6283.77	2950.13	1.99	17184.58	8067.88	6.37	69626.66	3.13	1.9688E+08
	11	2024	273	3.73	46063.79	21626.19	0.64	6284.97	2950.69	1.99	17194.43	8072.50	6.37	69543.18	3.10	1.9682E+08
	14	2024	270	3.73	46034.47	21612.43	0.64	6288.89	2952.53	2.00	17233.64	8090.91	6.37	69557.00	3.05	1.9669E+08
	24	2024	260	3.78	46903.72	22020.52	0.65	6368.19	2989.76	2.02	17447.64	8191.38	6.45	70719.55	2.87	1.9603E+08
Nov	3	2024	248	3.94	49624.07	23297.68	0.66	6578.45	3088.47	2.06	17785.68	8350.08	6.66	73988.19	2.68	1.9492E+08
	13	2024	236	4.27	55161.86	25897.59	0.69	6907.64	3243.02	2.12	18348.68	8614.40	7.07	80418.18	2.49	1.9321E+08
	23	2024	223	4.79	64660.99	30357.27	0.73	7406.54	3477.25	2.21	19242.24	9033.92	7.74	91309.77	2.29	1.9082E+08
Dec	3	2024	209	5.54	79402.48	37278.16	0.79	8080.75	3793.78	2.35	20686.43	9711.94	8.68	108169.66	2.08	1.8759E+08
	13	2024	197	6.54	102256.97	48007.97	0.84	8896.95	4176.97	2.61	23270.13	10924.94	9.99	134424.05	1.89	1.8335E+08
	23	2024	193	7.70	135931.16	63817.45	0.84	9363.51	4396.01	3.16	28999.61	13614.84	11.70	174294.28	1.76	1.7842E+08
Jan	2	2025	400	4.68	102401.32	48075.74	1.09	17691.65	8305.94	6.70	74594.58	35020.93	12.47	194687.55	3.58	1.9920E+08

Table 31-18 : Data for 2024 departures

Table 31-19: Range of departure dates for year 2027

Departure Date			TOF [days]	ΔV 1 [km/s]	Mass Propellant burn 1 [kg]	Burn Time [sec]	ΔV inc [km/s]	Mass Propellant burn inc.	Burn Time [sec]	ΔV 2 [km/s]	Mass Propellant burn 2 [kg]	Burn Time [sec]	Tot. ΔV [km/s]	Total Mass Propellant [kg]	Transfer Angle [rad]	Semi_major Axis [km]
Nov	6	2026	261	3.64	44037.21	20674.75	0.14	1366.47	641.54	2.34	20549.83	9647.81	6.12	65953.52	3.08	1.9155E+08
	7	2026	260	3.64	44029.58	20671.16	0.14	1364.21	640.47	2.34	20564.50	9654.69	6.12	65958.28	3.06	1.9151E+08
	9	2026	258	3.64	44058.95	20684.95	0.14	1368.10	642.30	2.34	20580.64	9662.27	6.12	66007.69	3.03	1.9143E+08
	11	2026	256	3.65	44162.51	20733.57	0.14	1364.23	640.48	2.35	20635.88	9688.21	6.14	66162.62	2.99	1.9132E+08
	13	2026	254	3.66	44325.67	20810.17	0.14	1377.21	646.58	2.35	20657.08	9698.16	6.15	66359.96	2.95	1.9122E+08
	16	2026	251	3.68	44714.91	20992.92	0.14	1389.79	652.48	2.36	20740.30	9737.23	6.18	66845.00	2.90	1.9102E+08
	26	2026	240	3.84	47247.27	22181.82	0.15	1514.93	711.24	2.39	21053.31	9884.18	6.38	69815.51	2.71	1.9011E+08
Dec	6	2026	228	4.13	52043.49	24433.56	0.17	1723.90	809.34	2.45	21589.15	10135.75	6.75	75356.55	2.52	1.8871E+08
	16	2026	215	4.61	60196.79	28261.40	0.20	2045.91	960.52	2.53	22419.92	10525.78	7.34	84662.61	2.32	1.8670E+08
	26	2026	202	5.30	73078.93	34309.36	0.24	2456.57	1153.32	2.66	23741.65	11146.31	8.19	99277.15	2.12	1.8394E+08
Jan	5	2027	189	6.20	92229.64	43300.30	0.28	2999.80	1408.36	2.86	25773.29	12100.14	9.34	121002.73	1.91	1.8031E+08
	15	2027	400	3.91	66696.04	31312.69	1.00	13262.04	6226.31	4.72	47108.81	22116.81	9.64	127066.89	3.94	1.9630E+08
	25	2027	400	4.05	68679.47	32243.88	0.95	12487.20	5862.53	4.65	46178.97	21680.27	9.65	127345.64	3.87	1.9693E+08
Feb	4	2027	400	4.20	70791.47	33235.43	0.89	11573.42	5433.53	4.58	45298.17	21266.74	9.67	127663.05	3.81	1.9758E+08

Table 31-19: Data for 2027 departures

32 Niziolek, Paul

32.1 CTV Atmosphere Supply

Author: Paul Niziolek

32.1.1 Significance and Purpose

The atmospheric supply parameters of the CTV need to be determined. This system provides the astronauts with an atmosphere that contains enough oxygen to allow astronauts to be productive. It removes the carbon dioxide and processes it to reclaim the oxygen. This requires an input of hydrogen from the main fuel tanks and a Sabatier/electrolysis system.

32.1.2 Inputs and Outputs

An atmospheric pressure of 101 kPa is maintained, with a partial pressure of 80 kPa nitrogen and 21 kPa of oxygen. First, we determine the volume of one mole of gas in spacecraft condition of 101 kPa and 298 K. Using Charles' Law in Equation 32-1 results in a volume of 0.02445 m^3/mole.

$$V_{final} = V_{initial} * T_{final} / T_{initial}$$

Equation 32-1

With a pressurized volume of 230 m^3, **Equation 32-2** calculates the mass of the ambient gas. For oxygen and nitrogen, this equation results in 62.6 kg oxygen and 208.6 kg nitrogen.

$$\frac{\text{Partial Pressure}}{\text{Total Pressure}} \cdot \text{Molar Mass (g/mol)} \cdot \frac{1\text{ kg}}{1000\text{ g}} \cdot \frac{1\text{ mol}}{0.02445\text{ m}^3} \cdot 230\text{ m}^3 = \text{Ambient Gas Mass (kg)}$$

Equation 32-2

Assuming a leakage rate of 0.14% mass per day, Equation 32-3 determines the actual amount of the ambient gas masses that must be carried for 1100 days. We find that a total of 974 kg nitrogen and 292 kg oxygen must be carried.

$$\frac{\text{Ambient Gas Mass}}{(1 - 0.0014)^{\text{Days}}} = \text{Needed Gas Mass (kg)}$$

Equation 32-3

An analysis of the Sabatier/electrolysis system is performed to determine if it can provide all the oxygen the astronauts need. The net reaction of this system is given in Equation 32-4.

$$CO_2 + H_2 \rightarrow CH_4 + O_2$$

Equation 32-4

Next, we compute the total amount of oxygen the four astronauts consume over 1100 days assuming a rate of 0.835 kg/p/d. [1,2] Equation 32-5 shows this computation which indicates that the astronauts require 3674 kg of oxygen.

$$\frac{O_2 \text{ consumed (kg)}}{\text{person * day}} \cdot \text{person * days} = \frac{0.835 \text{ kg}}{\text{p * d}} \cdot 4 * 1100 \text{ p * d} = 3674 \text{ kg } O_2$$

Equation 32-5

During this same duration, each astronaut generates 1 kg of carbon dioxide per day. [1, 2]. The total carbon dioxide generated is 4400 kg as shown in Equation 32-6.

$$\frac{CO_2 \text{ generated (kg)}}{\text{person * day}} \cdot \text{person * days} = \frac{1.00 \text{ kg}}{\text{p * d}} \cdot 4 * 1100 \text{ p * d} = 4400 \text{ kg } CO_2$$

Equation 32-6

Assuming that the Sabatier/electrolysis system can react all this carbon dioxide, the total mass of oxygen that is reclaimed is only 3200 kg, as shown in Equation 32-7. Thus we must supplement the astronauts' oxygen supply with the additional 476 kg in an on-board tank.

$$\frac{CO_2 \text{ generated (4400 kg)}}{1} \cdot \frac{\text{1kmole } CO_2}{\text{44 kg } CO_2} \cdot \frac{\text{1 kmole } O_2}{\text{1 kmole } CO_2} \cdot \frac{\text{32 kg } O_2}{\text{1 kmole } O_2} = 3200 \text{ kg } O_2 \text{ Reclaimed}$$

Equation 32-7

We round this supplemental oxygen to 500 kg to allow for inefficiencies and emergencies. We store this oxygen in the same tank as that of the pressurization oxygen for a total oxygen mass of 792 kg. Using tankage values from [1], we compute oxygen and nitrogen tank masses in Equation 32-8 and Equation 32-9.

$$792\,\text{kg O}_2\ \cdot \frac{0.364\,\text{kg tank}}{1\,\text{kg O}_2} = \cdot 288\,\text{kg O}_2\ \text{Tankage}$$

Equation 32-8

$$974\,\text{kg N}_2\ \cdot \frac{0.556\,\text{kg tank}}{1\,\text{kg N}_2} = \cdot 542\,\text{kg N}_2\ \text{Tankage}$$

Equation 32-9

We assume that the density of both gases in the tanks to be 1440 kg/m^3. This density value results in an oxygen tank volume of 0.55 m^3 and a nitrogen tank volume of 0.68 m^3.

Finally, we must determine the amount of hydrogen the Sabatier/electrolysis system consumes in order to reclaim the 3200 kg oxygen. Since one mole of oxygen corresponds to two moles of hydrogen, Equation 32-10 computes the mass of hydrogen needed.

$$\frac{\text{O}_2\ \text{reclaimed}\,(3200\,\text{kg})}{1} \cdot \frac{1\text{kmole O}_2}{32\,\text{kg O}_2} \cdot \frac{2\,\text{kmole H}_2}{1\,\text{kmole O}_2} \cdot \frac{2\,\text{kg H}_2}{1\,\text{kmole H}_2} = 400\,\text{kg H}_2\ \text{Consumed}$$

Equation 32-10

We see that this system requires 400 kg of hydrogen from the main fuel tanks in order to supply the astronauts with oxygen for the mission duration.

References

[1] Hanford, Anthony J., ed. NASA Johnson Space Center. Advanced Life Support Baseline Values and Assumptions Document. Aug. 2004. 1 Feb. 2005. <http://ston.jsc.nasa.gov/collections/TRS/_techrep/CR-2004-208941.pdf>

[2] Stilwell, Don, Ramzy Boutros, and Janis H. Connolly. "Crew Accommodations." Human Spaceflight: Mission Analysis and Design. Ed. Wiley J. Larson and Linda K. Prank. New York: McGraw-Hill, 1999. 575-606.

32.2 CTV Radiation Bunker Contents

Author: Paul Niziolek

32.2.1 Significance and Purpose

We design the CTV radiation bunker to protect the astronauts from solar particle events. It also protects against galactic cosmic radiation while the astronauts sleep in the bunker. Because the bunker is sealed during a solar event, the crew must have enough supplies and consumables to live during that period.

32.2.2 Analysis

Initially, a closed system was considered. Lithium hydroxide canisters would remove carbon dioxide and oxygen candles would produce oxygen using values calculated from [1]. This results in the system parameters in Table 32-1.

Table 32-1 Closed system for radiation bunker occupancy for three days (d) with four people (p). The final system we choose does not include the LiOH cartridges or the O_2 canisters

Locker Contents Item	Mass (kg/p/d)	Volume (m^3/p/d)	Mass (kg) (3 days, 4 people)	Volume (m^3) (3 days, 4 people)
LiOH cartridge	1.75	0.00125	21	0.015
O_2 Canister	2	0.0008	24	0.0095
Food	2.3	0.008	27.6	0.096
Water	1.6	0.0016	19.2	0.0192
Waste Bags	0.23	0.0008	2.76	0.0096
Data Link (PDA)	-	-	0.2	0.0005

However, [1] also notes that oxygen candles burn at a temperature of 700 K. This high temperature requires us to route heat pipes into the bunker to reduce the temperature. This increases the mass beyond what we desire. So we choose to eliminate the LiOH and O_2 candles and simply include a vent that delivers the CTV ambient air into the bunker. We assume that if this vent follows a circuitous route through the shielding, then the integrity of the radiation shielding is maintained and the crew is protected. The food in the bunker comes from the freezer and can be replenished at any time along with the water.

References

[1] Stilwell, Don, Ramzy Boutros, and Janis H. Connolly. "Crew Accommodations." Human Spaceflight: Mission Analysis and Design. Ed. Wiley J. Larson and Linda K. Prank. New York: McGraw-Hill, 1999. 575-606.

32.3 Stored Consumables System

Author: Paul Niziolek

Contributors: Paul Gramm, Tim Szamborski

32.3.1 Significance and Purpose

The Crew Transport Vehicle (CTV) Stored Consumables System provides the energy and nutrients astronauts need during interplanetary travel. The system is sized using values from [1] and [2].

Table 32-2 provides these values. The units of p and d represent person and day respectively. The freezer and refrigerator are sized as a function of the mass of the frozen food they contain.

Table 32-2 Sizing Values for the CTV Stored Consumables System.

Item	Mass	Volume	Power (kW)
Frozen Food	2.3 kg/person/day	0.008 m^3/p/d	-
Long-Term Freezer	0.75 kg/$kg_{\text{frozen food}}$	$3.93 \times 10^{-3} m^3/kg_{\text{frozen food}}$	$0.33 \times 10^{-3}/kg_{\text{frozen food}}$
Refrigerator	0.75 kg/$kg_{\text{frozen food}}$	$3.93 \times 10^{-3} m^3/kg_{\text{frozen food}}$	$0.33 \times 10^{-3}/kg_{\text{frozen food}}$
Food Warmer	5 kg	0.005 m^3	0.015
Cleaning Supplies	0.0083 kg/p/day	0.0067 m^3/kg	-
Utensils	0.0017 kg/p/day	-	-

For a mission duration of 1100 days with four astronauts, the system results in the values in Table 32-3. The calculations are not shown in detail due to their simplicity.

Table 32-3 System Parameter Values for the CTV Stored Consumables System for a mission length of 1100 days and four astronauts.

Item	Mass (kg)	Volume (m^3)	Power (kW)
Frozen Food	10120	35.2	-
Long-Term Freezer	7590	39.8	3.3
Refrigerator	412	2	0.2
Food Warmer (2x)	10	0.01	0.03
Cleaning Supplies	37	0.2	-
Utensils	7	-	-
Total System	18176	42.01	3.53

We considered using a plant system on the CTV, but it was determined that implementation of plants would result in a massive system that was not worth the costs. Since we plan on loading food into the CTV immediately prior to mission launch, food spoilage is not a major factor if it is frozen. Also, the requirement to be able to operate in zero gravity complicated any plant system that might be implemented. For these reasons, only frozen food is carried on the CTV.

References

[1] Hanford, Anthony J., ed. NASA Johnson Space Center. Advanced Life Support Baseline Values and Assumptions Document. Aug. 2004. 1 Feb. 2005. <http://ston.jsc.nasa.gov/collections/TRS/_techrep/CR-2004-208941.pdf>

[2] Stilwell, Don, Ramzy Boutros, and Janis H. Connolly. "Crew Accommodations." Human Spaceflight: Mission Analysis and Design. Ed. Wiley J. Larson and Linda K. Prank. New York: McGraw-Hill, 1999. 575-606.

32.4 NCRP Report No. 98 Review

Author: Paul Niziolek

32.4.1 Significance and Purpose

We perform a radiation risk assessment to determine the survivability of the crew during and after the mission. We use the NCRP 98 report to give us the risk of cancer death from prolonged exposure and acute exposure.

32.4.2 NCRP Report No. 98 Review

Galactic cosmic radiation (GCR) is the major radiation in the outer space outside of Earth's magnetosphere. GCR would primarily be the source of long-term protracted exposure in a Mars mission. Solar particle events (SPE) are emissions of protons and other heavier particles form the sun. They are unpredictable and can cause acute effects, as opposed to the protracted exposure from GCR.

Acute effects are not expected to result from the exposure to the radiation environment from space, except in the case of large SPE. [p. 4] As such, the major concern is going to be long-term effects, primarily that of cancer. Cataracts are also mentioned as a concern, but for our mission we will concentrate on minimizing the risk of cancer to the astronauts.

The S.I. unit of absorbed dose is the gray (Gy), such that 1 Gy is equal to the net absorption of one joule in one kilogram of any material, assumed to be water in this report. A rad is also used and 100 rads are equal to 1 gray. Different types of radiation cause different amounts of biological damage per gray. The amount of energy loss per unit of track length (as it travels through tissue) is proportional to the amount of damage. This energy loss is called linear energy transfer (LET). Higher LET particles cause more damage than lower LET particles. To convert high LET radiation to normal LET radiation, a quality factor (Q) is used. The resulting amount is called the dose equivalent. This dose equivalent has units of Sieverts (Sv) or rem, where 100 rem equal 1 Sv. The report stated that a mission to Mars was estimated to deliver 1 Sv total to the blood forming organs for a 3-year mission. This was computed with a 2 g/cm^2 Al shielding on the spacecraft and habitat and 10 g/cm^2 CO_2 shielding while on Mars.

An average value of Q was given as 2.9 for all the GCR and secondaries (Bremsstrahlung radiation) for a craft with approximately 4 g/cm^2 Aluminum shielding that is outside of the magnetosphere. [p. 5]

"Current risk estimates for radiation depend, to a significant extent, on the data from atomic bomb survivors and, therefore, the accuracy of these data, the dosimetry, and the appropriateness of extrapolating risk from a Japanese to a U.S. population are important." [p.8] So we must take the report's risk calculations as being rough estimates that may or may not apply to the selected astronauts.

"In the years of lower solar activity, the solar wind is not so strong and the intensity of GCR is high." [p.19] Thus if we traveled during the solar minimum there would be a lower chance for a large SPE, but at the same time there would be a definite increase in GCR. Because it seems that proper shielding and the lesser effects of protracted exposures on excess cancers, it is recommended that the crews travel during solar minimum to lessen their chance of encountering a mission-failure-class SPE.

"There is an average delay time from the maximum flare emission to the arrival of the first significantly energetic particles (radiation) of 15 hours, with a range of 15 minutes to 60 hours." [p.25] Thus there exists the possibility of a long enough time to get the astronauts into a "radiation shelter" after the SPE has been first identified.

There is an effective mortality threshold of 1.5 Gray in a single, high-dose exposure. Note that on average, radiation induced skin cancers do not appear for over 20 years after the exposure [p. 107]. So our 15 year criteria is generally going to be satisfied because as long as the astronauts do not experience an SPE that kills them during the mission, they will most likely develop cancers many years (15+) after they get back from the mission. However, the report also stated that Leukemia is the most commonly reported cancer following exposure [p.108]. This cancer does not have a latent period and peaks around 6-8 years after exposure, thus this appears to be the cancer that the astronauts would get within 15 years (if they were to get a radiation-related cancer in the first place).

"It is somewhat surprising that while we apparently know more about radiation as a carcinogen than any other environmental exposure agent, the evidence necessary to make age, sex, and time-specific risk extrapolation is sufficiently weak to warrant great caution in the interpretation of these tabular data on projected radiation risks." [p 116] Take these risk extrapolations with a grain of salt and realize that there is not enough information on human exposure to make strong extrapolations and predictions.

Risk estimates were presented for excess cancers per 1000 persons. Risk percentage was calculated by the formula: Risk = excess cancers / 1000 persons *100%. Please refer to the uploaded spreadsheet "NCRP98_radiation_exposure.xls" to calculate risks. Below is the adjusted risk for a protracted exposure of 0.5 Gy/year for 5 years.

Table 4: 0.5 Gy / year for 5 years, excess cancers per 1000 persons

Sex	Age at Exposure	Total Cancers	Risk
Male	25	89.3	8.93%
	35	50.8	5.08%
	45	36.3	3.63%
	55	30.0	3.00%
Female	25	173.4	17.34%
	35	104.2	10.42%
	45	58.4	5.84%
	55	46.7	4.67%

Note that as a result it was chosen to have women of age 45+ be allowed for consideration for the mission. It would be possible to have a woman 35+, but not recommended. Below is the risk assessment for an acute exposure of 1 Gy.

Table 5: Acute exposure to 1 Gy, excess cancers per 1000 persons

Sex	Age at Exposure	Total Cancers	**Risk**
Male	25	74.74	7.47%
	35	41.14	4.11%
	45	27.94	2.79%
	55	23.14	2.31%
Female	25	138.34	13.83%
	35	85.37	8.54%
	45	44.57	4.46%
	55	35.83	3.58%

Because of the drop in risk for men from age 25 to 35, it is recommended that men of age 35+ be considered for the mission. Also, the radiation shielding used must be able to limit the absorbed dose to any astronaut to roughly 1 gray (100 rem) and lower the protracted dosage to less than 0.5 gray per year.

References

[1] National Council on Radiation Protection and Measurements. Guidance on Radiation Received in Space Activities. NCRP No. 98. Bethesda, MD. 1989.

32.5 Radiation Risk Assessment

Author: Paul Niziolek

32.5.1 Significance and Purpose

We perform a radiation risk assessment to determine the survivability of the crew during and after the mission. We use the NCRP 98 report to give us the risk of cancer death from prolonged exposure and acute exposure. The NCRP report is detailed in NCRP Report No. 98 Review Appendix section. Only the particulars to our missions are detailed in this section.

32.5.2 Inputs and Outputs

The equations and values provided in [1] provide the methodology to assess the crew radiation risk. We calculate these risks for each mission. Mission A lands on the surface of Mars and stays for a significant duration. Mission B lands on the surface for several days only. Mission C stays in space the entire mission duration. Only Mission A gains a radiation shielding benefit from the Martian atmosphere. Table 32-6 presents the cancer mortality risks to the crew.

Table 32-6 Total cancer mortality risk due to amount of radiation absorbed in each mission type. Mission A lands on the surface of Mars. Missions B and C do not gain any benefit from Martian atmospheric shielding.

		Mission A	**Mission B & C**
		300 rem	**500 rem**
Sex	**Age at Exposure**	**Risk**	**Risk**
Male	25	11%	19%
	35	7%	11%
	45	5%	8%
	55	4%	7%
Female	25	16%	27%
	35	10%	16%
	45	6%	10%
	55	5%	9%

References

[1] National Council on Radiation Protection and Measurements. Guidance on Radiation Received in Space Activities. NCRP No. 98. Bethesda, MD. 1989.

[2] Mettler, Fred and Arthur Upton. Medical Effects of Ionizing Radiation. W.B. Saunders Company. 2nd Ed. Philadelphia. 1995.

[3] Nicogossian, Arnauld, editors et all. Space Biology and Medicine: III Humans in Spaceflight Book 2. AIAA. Reston, VA. 1996.

33 Parkison, Ben

33.1.1 Appendix – Ben Parkison

33.1.1.1 ELV Structural Design Primary Code

The following code, ran in Matlab, conducts the entire structural modeling and design of the Earth Launch Vehicle. The inputs for the code are the four volumes of fuels and oxidizers needed. The output is the variable "summary" which details the mass and dimensions of the vehicle.

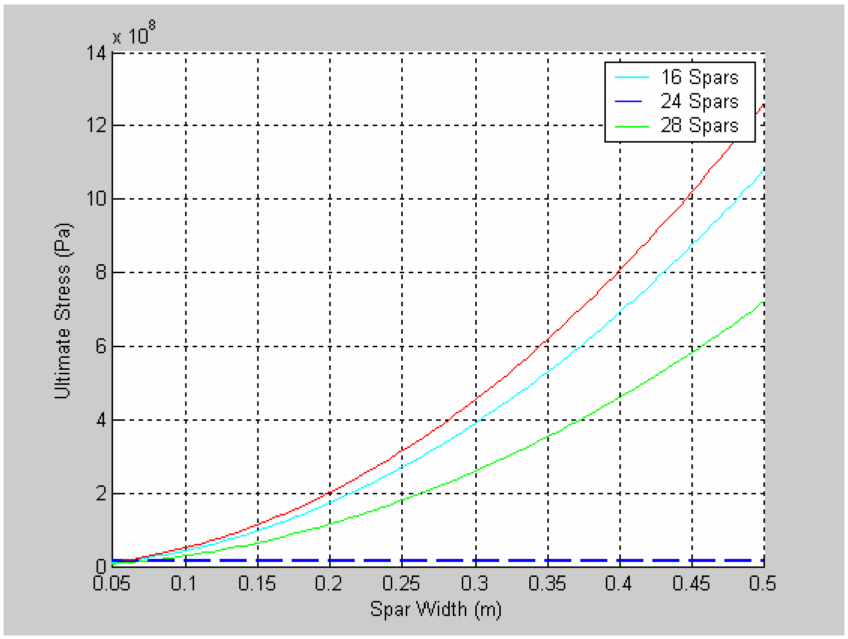
```
% FUEL VOLUMES
%
%            Stage 1         Stage 2
%           ____|___        ____|____
%           |      |        |       |
%          RP1    LOX      LH2     LOX
volumes = [695    1340     640     240];   % m^3




alm_prop = [2.705*1000 69 110*1000000];
car_prop = [1.2*1000   1000 2530*1000000];
honey_prop = [74];
graph_prop = [167];

diameter   = 10.5;
radius_out = diameter/2;
radius_in  = radius_out-0.06;

g = 9.81;
aero_stress = 75000; %Pa
aero_force  = aero_stress/(pi*radius_out^2);
max_accel   = 6*g;
stage1_prop = 4*5393;
stage2_prop = 3*3177;
stage1_prop_h = 3.56;

stringer_number = 30;
rib_spacing     = 0.5;
rib_base        = 0.02;
tank_skin       = 0.001;
stringer_base   = 0.02;

% --------------------------------------
% Fairing Modeling
% --------------------------------------

%Dimensions
fairing_cyl_h = 16.5;
fairing_con_h = 6;
fairing_cap_r = 0.75;
fairing_cyl_wall_t = 0.5;
fairing_con_wall_t = 0.5;
fairing_cap_wall_t = 0.5;
fairing_skin_t     = 0.05;

%Calculations
fairing_vol_out.cyl        = pi*(radius_out^2)*fairing_cyl_h;
fairing_vol_out.total_cone = (pi*(radius_out^2)*(fairing_con_h+fairing_cap_r))/3;
fairing_vol_out.extra_cone = (pi*(fairing_cap_r^2)*fairing_cap_r)/3;
fairing_vol_out.cap = (2/3)*pi*fairing_cap_r^3;
fairing_vol_out.total      = fairing_vol_out.cyl + fairing_vol_out.total_cone +
fairing_vol_out.extra_cone + fairing_vol_out.cap;

fairing_vol_in.cyl = pi*(radius_in^2)*fairing_cyl_h;
fairing_vol_in.total_cone = (pi*(radius_in^2)*(fairing_con_h+fairing_cap_r))/3;
fairing_vol_in.extra_cone = (pi*((fairing_cap_r-fairing_cap_wall_t)^2)*(fairing_cap_r-
fairing_cap_wall_t))/3;
fairing_vol_in.cap = (2/3)*pi*(fairing_cap_r-fairing_cap_wall_t)^3;
fairing_vol_in.total = fairing_vol_in.cyl + fairing_vol_in.total_cone+fairing_vol_in.extra_cone +
fairing_vol_in.cap;

fairing_vol_step.cyl = pi*((radius_in+fairing_skin_t)^2)*fairing_cyl_h;
fairing_vol_step.total_cone = (pi*((radius_in+fairing_skin_t)^2)*(fairing_con_h+fairing_cap_r))/3;
fairing_vol_step.extra_cone = (pi*((fairing_cap_r-fairing_cap_wall_t)^2)*(fairing_cap_r-
fairing_cap_wall_t))/3;
fairing_vol_step.cap = (2/3)*pi*(fairing_cap_r+fairing_skin_t)^3;
fairing_vol_step.total = fairing_vol_step.cyl +
fairing_vol_step.total_cone+fairing_vol_step.extra_cone + fairing_vol_step.cap;

fairing_str_vol = fairing_vol_out.total-fairing_vol_in.total;
fairing_skn_vol = fairing_vol_step.total-fairing_vol_out.total;
fairing_mass    = fairing_str_vol*honey_prop(1)+fairing_skn_vol*graph_prop(1);

%fairing_mass = 12520; %16.5
fairing_mass = 13679; %18.5

% -------------------------------------
% Total Mass Update
% -------------------------------------

total_mass = fairing_mass;

% =====================================
% Second Stage
% =====================================


% -------------------------------------
% 2nd Stage LOX Tank
% -------------------------------------

%Dimensions
tank_skin       = 0.019;
tank1.radius          = radius_in;
tank1.vol             = volumes(4);
tank1_stringer.number = stringer_number;
tank1_rib.spacing     = rib_spacing;
tank1_rib.base        = rib_base;
tank1_skin            = tank_skin;
tank1_stringer.base   = stringer_base;

%Stress Calculations
tank1.force  = (total_mass*max_accel+aero_force)/tank1_stringer.number;
tank1.height = tank1.vol/((4/3)*pi*tank1.radius^2);
tank1_stringer.width = ((12*tank1.force+tank1.height)/(pi*car_prop(3)*tank1_stringer.base))^(1/3);
tank1_rib.width = tank1_stringer.width;

%Mass Calculations
tank1_rib.number = tank1.height/tank1_rib.spacing;
tank1_vol.stringer = tank1_stringer.base*tank1_stringer.width*tank1.height*tank1_stringer.number;
tank1_vol.ribs = tank1_rib.width*tank1_rib.base*(2*pi*tank1.radius)*(tank1_rib.number);
tank1_vol.intersections =
tank1_rib.number*tank1_stringer.number*tank1_rib.base*tank1_stringer.base*tank1_rib.width;
tank1_vol.skin = -(tank1.height*pi*tank1.radius^2)+(tank1.height*pi*(tank1.radius+tank1_skin)^2);

tank1_vol.total = tank1_vol.stringer+tank1_vol.ribs+tank1_vol.skin-tank1_vol.intersections;
tank1_mass = tank1_vol.total*car_prop(1);

tank1_mass = 0;
% -------------------------------------
% Total Mass Update
% -------------------------------------

total_mass = fairing_mass+tank1_mass;

% -------------------------------------
% 2nd Stage Intertank
% -------------------------------------

intertank1.height = radius_out+tank1.height/2;
intertank1.skin   = tank1_skin;

intertank1_vol.skin = -
(intertank1.height*pi*tank1.radius^2)+(intertank1.height*pi*(tank1.radius+tank1_skin)^2);
intertank1_vol.stringer =
tank1_stringer.base*tank1_stringer.width*intertank1.height*tank1_stringer.number;
intertank1_vol.total = intertank1_vol.skin+intertank1_vol.stringer;

intertank1_mass = intertank1_vol.total*car_prop(1);

%intertank1_mass = 0;
% -------------------------------------
% Total Mass Update
% -------------------------------------

total_mass = fairing_mass+tank1_mass+intertank1_mass;

% -------------------------------------
% 2nd Stage LH2 Tank
% -------------------------------------

%Dimensions
tank2.radius          = radius_in;
tank2.vol             = volumes(3) +volumes(4);
tank2_stringer.number = stringer_number;
tank2_rib.spacing     = rib_spacing;
tank2_rib.base        = rib_base;
tank2_skin            = tank_skin;
tank2_stringer.base   = stringer_base;

%Stress Calculations
tank2.force  = (total_mass*max_accel+aero_force)/tank2_stringer.number;
tank2.cyl_height = (tank2.vol-(4/3)*pi*tank2.radius^3)/(pi*tank2.radius^2);
tank2.height = tank2.cyl_height+2*tank2.radius;
tank2_stringer.width = ((12*tank2.force+tank2.height)/(pi*car_prop(3)*tank2_stringer.base))^(1/3);
tank2_rib.width = tank2_stringer.width;

%Mass Calculations
tank2_rib.number = tank2.height/tank2_rib.spacing;
tank2_vol.stringer = tank2_stringer.base*tank2_stringer.width*tank2.height*tank2_stringer.number;
tank2_vol.ribs = tank2_rib.width*tank2_rib.base*(2*pi*tank2.radius)*(tank2_rib.number);
tank2_vol.intersections =
tank2_rib.number*tank2_stringer.number*tank2_rib.base*tank2_stringer.base*tank2_rib.width;
tank2_vol.skin = -(tank2.height*pi*tank2.radius^2)+(tank2.height*pi*(tank2.radius+tank2_skin)^2);

tank2_vol.total = tank2_vol.stringer+tank2_vol.ribs+tank2_vol.skin-tank2_vol.intersections;
tank2_mass = tank2_vol.total*car_prop(1);

% -------------------------------------
% Total Mass Update
% -------------------------------------

total_mass = fairing_mass+tank1_mass+intertank1_mass+tank2_mass;

% -------------------------------------
% Interstage
% -------------------------------------

interstage.height = 4.24;
interstage.skin   = tank2_skin;
```

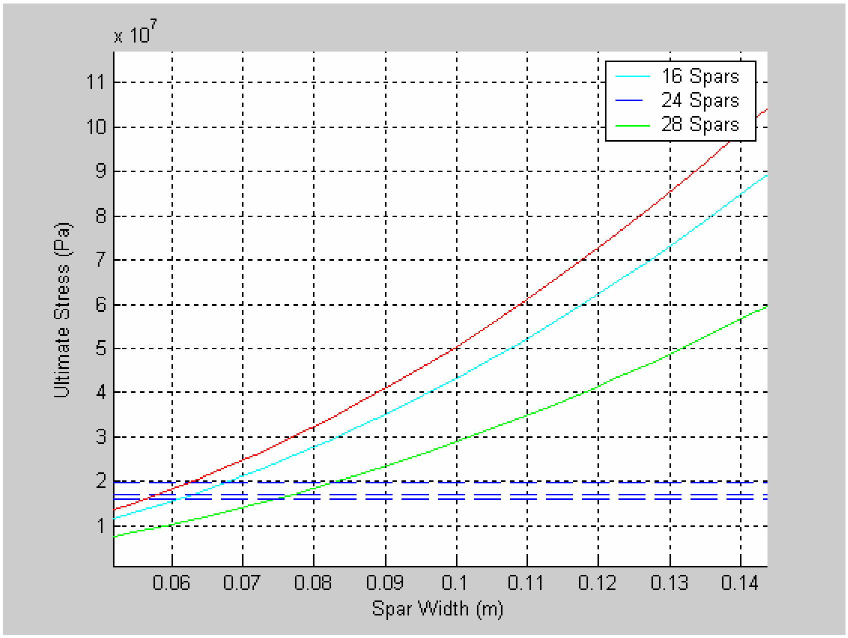
```
interstage_vol.skin = -
(interstage.height*pi*tank2.radius^2)+(interstage.height*pi*(tank2.radius+tank2_skin)^2);
interstage_vol.stringer =
tank2_stringer.base*tank2_stringer.width*interstage.height*tank2_stringer.number;
interstage_vol.total = interstage_vol.skin+interstage_vol.stringer;

interstage_mass = interstage_vol.total*car_prop(1);

% ----------------------------------------
% Total Mass Update
% ----------------------------------------

total_mass = fairing_mass+tank1_mass+intertank1_mass+tank2_mass+interstage_mass;

% ========================================
% First Stage
% ========================================

% ----------------------------------------
% 1st Stage LOX Tank
% ----------------------------------------

%Dimensions
tank3.radius          = radius_in;
tank3.vol             = volumes(2);
tank3_stringer.number = stringer_number;
tank3_rib.spacing     = rib_spacing;
tank3_rib.base        = rib_base;
tank3_skin            = tank_skin;
tank3_stringer.base   = stringer_base;

%Stress Calculations
tank3.force  = (total_mass*max_accel+aero_force)/tank3_stringer.number;
tank3.cyl_height = (tank3.vol-(4/3)*pi*tank3.radius^3)/(pi*tank3.radius^2);
tank3.height = tank3.cyl_height+2*tank3.radius;
tank3_stringer.width = ((12*tank3.force+tank3.height)/(pi*car_prop(3)*tank3_stringer.base))^(1/3);
tank3_rib.width = tank3_stringer.width;

%Mass Calculations
tank3_rib.number = tank3.height/tank3_rib.spacing;
tank3_vol.stringer = tank3_stringer.base*tank3_stringer.width*tank3.height*tank3_stringer.number;
tank3_vol.ribs = tank3_rib.width*tank3_rib.base*(2*pi*tank3.radius)*(tank3_rib.number);
tank3_vol.intersections =
tank3_rib.number*tank3_stringer.number*tank3_rib.base*tank3_stringer.base*tank3_rib.width;
tank3_vol.skin = -(tank3.height*pi*tank3.radius^2)+(tank3.height*pi*(tank3.radius+tank3_skin)^2);

tank3_vol.total = tank3_vol.stringer+tank3_vol.ribs+tank3_vol.skin-tank3_vol.intersections;
tank3_mass = tank3_vol.total*car_prop(1);

% ----------------------------------------
% Total Mass Update
% ----------------------------------------

total_mass = fairing_mass+tank1_mass+intertank1_mass+tank2_mass+interstage_mass+tank3_mass;

% ----------------------------------------
% 2nd Stage Intertank
% ----------------------------------------

intertank2.height = radius_out+tank3.height/2;
intertank2.skin   = tank3_skin;

intertank2_vol.skin = -
(intertank2.height*pi*tank3.radius^2)+(intertank2.height*pi*(tank3.radius+tank3_skin)^2);
intertank2_vol.stringer =
tank3_stringer.base*tank3_stringer.width*intertank2.height*tank3_stringer.number;
intertank2_vol.total = intertank2_vol.skin+intertank2_vol.stringer;

intertank2_mass = intertank2_vol.total*car_prop(1);

% ----------------------------------------
% Total Mass Update
% -------------------------------------

total_mass =
fairing_mass+tank1_mass+intertank1_mass+tank2_mass+interstage_mass+tank3_mass+intertank2_mass;

% -------------------------------------
% 2nd Stage RP-1 Tank
% -------------------------------------

%Dimensions
tank4.radius          = radius_in;
tank4.vol             = volumes(1);
tank4_stringer.number = stringer_number;
tank4_rib.spacing     = rib_spacing;
tank4_rib.base        = rib_base;
tank4_skin            = tank_skin;
tank4_stringer.base   = stringer_base;

%Stress Calculations
tank4.force  = (total_mass*max_accel+aero_force)/tank4_stringer.number;
tank4.cyl_height = (tank4.vol-(4/3)*pi*tank4.radius^3)/(pi*tank4.radius^2);
tank4.height = tank4.cyl_height+2*tank4.radius;
tank4_stringer.width = ((12*tank4.force+tank4.height)/(pi*car_prop(3)*tank4_stringer.base))^(1/3);
tank4_rib.width = tank4_stringer.width;

%Mass Calculations
tank4_rib.number = tank4.height/tank4_rib.spacing;
tank4_vol.stringer = tank4_stringer.base*tank4_stringer.width*tank4.height*tank4_stringer.number;
tank4_vol.ribs = tank4_rib.width*tank4_rib.base*(2*pi*tank4.radius)*(tank4_rib.number);
tank4_vol.intersections =
tank4_rib.number*tank4_stringer.number*tank4_rib.base*tank4_stringer.base*tank4_rib.width;
tank4_vol.skin = -(tank4.height*pi*tank4.radius^2)+(tank4.height*pi*(tank4.radius+tank4_skin)^2);

tank4_vol.total = tank4_vol.stringer+tank4_vol.ribs+tank4_vol.skin-tank4_vol.intersections;
tank4_mass = tank4_vol.total*car_prop(1);

% -------------------------------------
% Interstage
% -------------------------------------

interstagetop.height = 2.12;
interstagetop.skin   = tank2_skin;

interstagetop_vol.skin = -
(interstagetop.height*pi*tank2.radius^2)+(interstagetop.height*pi*(tank2.radius+tank2_skin)^2);
interstagetop_vol.stringer =
tank2_stringer.base*tank2_stringer.width*interstagetop.height*tank2_stringer.number;
interstagetop_vol.total = interstagetop_vol.skin+interstagetop_vol.stringer;

interstagetop_mass = interstagetop_vol.total*car_prop(1);

% -------------------------------------
% Final Mass Update
% -------------------------------------

mass.fairing = fairing_mass;
mass.stage2 = tank1_mass+intertank1_mass+tank2_mass+stage2_prop+interstagetop_mass;
mass.stage1 = interstage_mass+tank3_mass+intertank2_mass+tank4_mass+stage1_prop;
mass.total  = mass.fairing+mass.stage2+mass.stage1;

% -------------------------------------
% Other Mass
% -------------------------------------

thermal = 1550;
comm = 100;
dnc = 33;
power = 10;
```

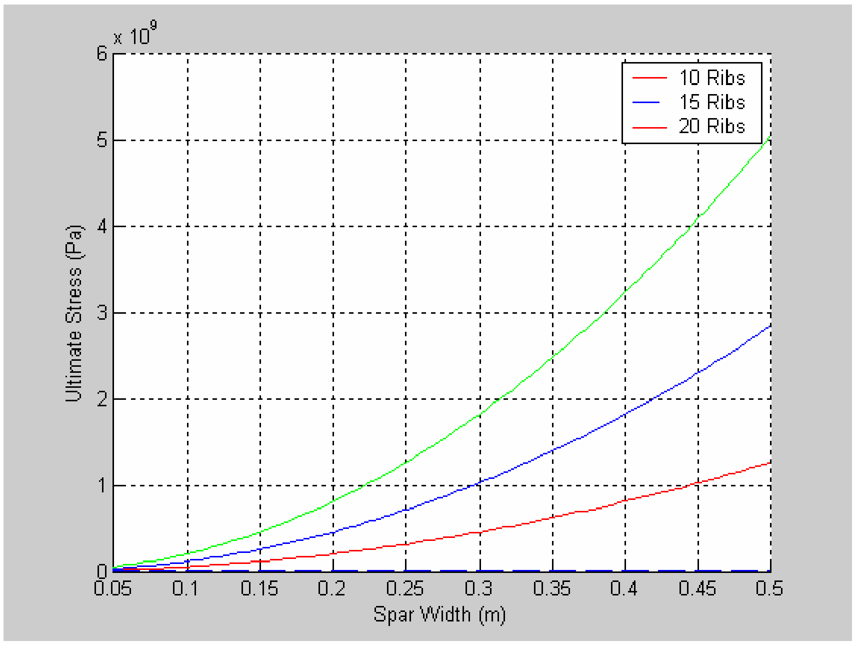
```
% -------------------------------------
% Summary
% -------------------------------------

summary.stage1_mass = mass.stage1;
summary.stage2_mass = mass.stage2;
summary.fairing_mass = mass.fairing;
summary.total_mass = mass.total+thermal+comm+dnc+power;
%summary.fairing_height = fairing_cyl_h+fairing_con_h+fairing_cap_r;
summary.fairing_height = 18.5;
summary.stage2_height = tank1.height+tank2.height+interstage.height;
summary.stage1_height = tank3.height+tank4.height;
summary.total_height =
summary.fairing_height+summary.stage1_height+summary.stage2_height+stage1_prop_h;

summary
```

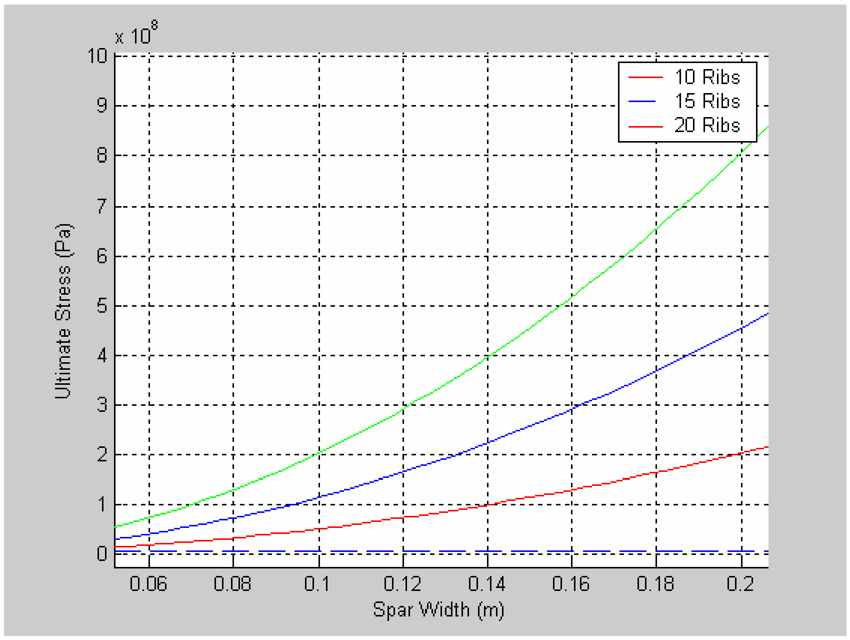
33.1.1.2 Rib/Spar Optimization Study

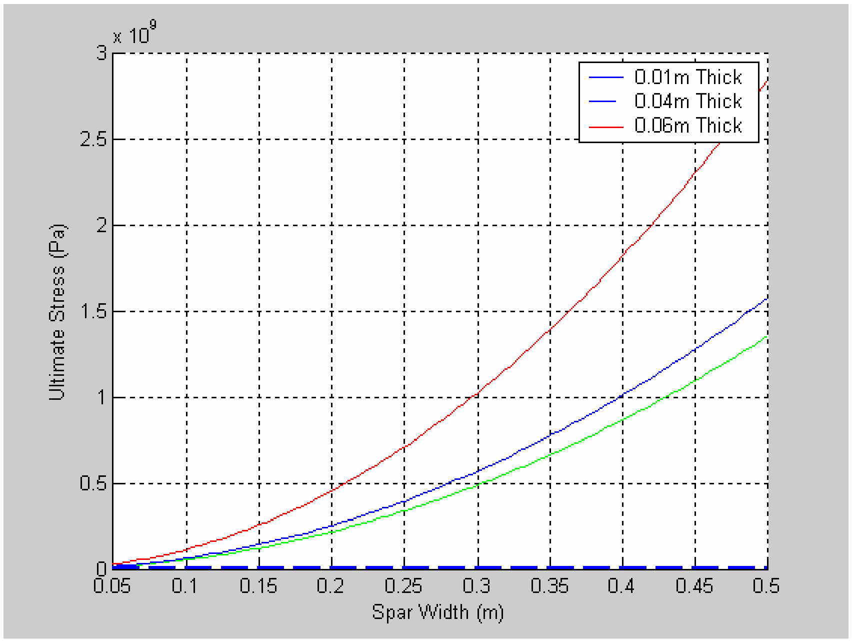
The following are the primary results which let to the determination of the number of ribs and spars to use within the ELV fuel tanks. The input is left up to the user. Examples of inputs would include number of spars, number of ribs, thickness of either, and the acting forces.

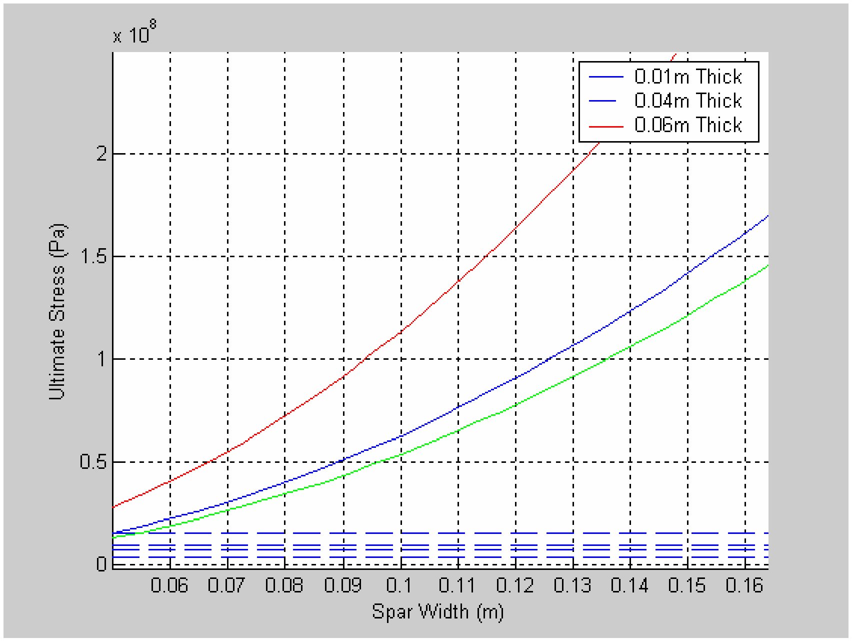

x 10^8
14
12
10
8
6
4
2
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Ultimate Stress (Pa)
Spar Width (m)
16 Spars
24 Spars
28 Spars


x 10^7
11
10
9
8
7
6
5
4
3
2
1
0.06
0.07
0.08
0.09
0.1
0.11
0.12
0.13
0.14
Ultimate Stress (Pa)
Spar Width (m)
16 Spars
24 Spars
28 Spars


x 10^9
Ultimate Stress (Pa)
Spar Width (m)
10 Ribs
15 Ribs
20 Ribs
0
1
2
3
4
5
6
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5


x 10^8
Ultimate Stress (Pa)
Spar Width (m)
10 Ribs
15 Ribs
20 Ribs
0
1
2
3
4
5
6
7
8
9
10
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2





33.1.1.3 Fuel Tank Optimization Code

The following code, run in Matlab, conducts a mass optimization comparison study for the needed fuel tank. The input to the code is the volume and type of fuel. The output is the mass of the tank. The fuel volume can also be input as an array, creating a comparative graph.

```
% ==========================================
% The following code conducts a study of the
% number and size of spars and ribs in the
% ELV. The purpose of this study is to create
% a baseline solution of the number and size
% of each component to begin the ELV structural
% analysis.
%
% All units in SI
%
% Written by: Ben Parkison
% Updated: 03/05/2005
% ==========================================

clear all
clc

height = 100;                   % Vehicle height estimation
ribs = 10;                      % Number of ribs
spars = 24;                     % Number of spars

% -------------------
% Material Properties
% -------------------

E   = 6.8900e+010;              % E (pa)
rho = 2710;                     % Density (kg/m^3)
L   = height/ribs;              % Spar length (meters)
r   = 5.1;                      % Vehicle Radius (m)
t   = 0.1;                      % Spar Thickness (m)
w   = linspace(0.05,.5,100);    % Spar Width (m)
v = 0.33;                       % Poisson's Ratio


I = w.^4./12;                   % Spar Moment of Intertia
A = w.^2;                       % Spar Cross Section Area

p = sqrt(I./A);

% ------------------
% Forces
% ------------------

drag =   59146000;              % Newtons
thrust = 34000000;              % Newtons

force = drag+thrust;
cross = spars* mean(A)+pi*(r+mean(t))^2-pi.*r.^2;   % Required Cross Section
P = force/cross;

% ------------------
% Buckling Stresses
% ------------------

P_col = spars.*(pi.*E)./(L./p).^2;                   % Column Buckling Stess
P_cyl = 0.807.*((E.*t.^2)./(L.*r))…
…*sqrt(((1./(1-v.^3)).^3).*(t.^2./r.^2));    %Cylinder Buckling Stress
cross2 = spars.*A+pi.*(r+t).^2-pi.*r.^2;     % Required Cross Section
mass = rho.*height.*cross2+spar_m*spars;

% ------------------
% Plotting
% ------------------

hold on
plot(w,P_col,'c')
hold on
plot(w,P.*ones(length(w),1),'--')
```

34 Pollock, George E.

34.1 Mars Parking Orbit Period Trade Study Appendix

Author: George Pollock

34.1.1 MPO Period Trade Study Description

Based upon maneuvers for the September 2028 non-free return launch opportunity, we investigate the impact of Mars Parking Orbit (MPO) period on total system mass at Earth departure. From the results of this analysis, we choose the period of the CTV's parking orbit at Mars.

34.1.2 Trade Study Inputs

Inputs to this trade study are: the CTV maneuver profile (magnitude of all major propulsive burns) and empty masses of the CTV and the MLV. This code generates plots of CTV and MLV gross masses as a function of MPO period, as well as CTV ΔV and propellant mass as functions of MPO period.

34.1.3 Trade Study Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: MPOscript_final.m
% Mars Parking Orbit Sizing Script - Final Version for completed design
% Author: George E. Pollock
% Last Modified on March 27, 2005 by George E. Pollock
%
% THIS SCRIPT ANALYZES THE TOTAL SYSTEM MASS AT EARTH DEPARTURE AS A
% FUNCTION OF MARS PARKING ORBIT PERIOD, BASED UPON MANEUVERS FROM
% THE SEPT. 2028 NON-FREE-RETURN LAUNCH OPPORTUNITY,  CTV EMPTY MASS
% OF 134115 kg, MLV EMPTY MASS OF 3953 kg.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all; clear all; clc;

% Gravitational Parameter for the Earth (mu = GM)
mu_earth = 3.98600485e5;    % km^3/s^2

% Gravitational Parameter for Mars (mu = GM)
mu_mars = 4.282828e4;       % km^3/s^2

% September 2028 Non-Free-Return, Worst-Case Launch Opportunity
% Delta-Vs, vehicle masses, and propulsion system performance data
% Earth departure Delta-V [km/s]
delV_Edep = 4.59;
% Earth capture Delta-V [km/s]
delV_Ecapt = 1.11;

% Final mass of CTV [kg]
m_f_CTV = 134115;

% Isp of CTV's Nuclear Thermal Rocket Engines [sec]
Isp_CTV = 1000;

% Gross mass of ARV [kg]
m_0_ARV = 5000;

% Final Mass of MLV [kg]
m_f_MLV = 3953;

% Isp's of Descent and Ascent stages of MLV [sec]
Isp_1 = 440;
Isp_2 = 465.5;

% V_infinity at Mars arrival [km/s]
Vinf_Marr = 3.32;

% Semi-major axis of arrival hyperbola [km]
a_arr = -mu_mars/Vinf_Marr^2;

% V_infinity at Mars departure [km/s]
Vinf_Mdep = 2.90;

% Semi-major axis of departure hyperbola [km]
a_dep = -mu_mars/Vinf_Mdep^2;

% compute 350km circular orbit period
Pc = 2*pi*sqrt((3397+350)^3/4.282828e4)/3600/24;    % [days]

% Apo-twist inclination change requirement [degrees]
incl = 65;



% Mars Parking Orbit Period [days]
P = [linspace(Pc,1,15), [1:.5:7]];

% initialize vectors
delV_capt = zeros(1,length(P));
```

```
delV_at1 = zeros(1,length(P));
delV_at2 = zeros(1,length(P));
delVdep = zeros(1,length(P));
delV_deorb = zeros(1,length(P));
delV_rndvs = zeros(1,length(P));

for j = 1:length(P)

  % compute CTV delta-Vs and deorbit and rendezvous delta-Vs for MLV
  % (input arrival and departure conditions, define entry interface
  % conditions in MPO function)
  [delV_capt(1,j), delV_at1(1,j), delV_at2(1,j), delV_dep(1,j), ...
        delV_deorb(1,j), delV_rndvs(1,j)] = MPOfinal(P(j), a_arr, incl, a_dep);

  % Compute initial mass of the lander (prior to powered descent, following
  % the aeroentry and parachute phases of landing--after which the TPS and
  % parachutes have been jettisoned), based upon delta-V requirements for
  % specified parking orbit period
  [m_0_MLV(j) m_in1(j), m_prop1(j), m_in2(j), m_prop2(j)] = ...
    MLVmass_final(m_f_MLV,delV_deorb(j)*10^3,delV_rndvs(j)*10^3,Isp_1, Isp_2);

  % Propellant mass for MLV [kg]
  m_prop_MLV(j) = m_0_MLV(j) - m_f_MLV;

  % TOTAL MLV MASS [kg] (including a factor of 3.1% of total mass for
  % parachutes, and 1/12 total mass for TPS/lifting entry body)
  m_MLV_total(j) = m_0_MLV(j)/.8177;

  % Parachute mass [kg]
  m_chutes(j) = 0.031*m_MLV_total(j);

  % Thermal Protection System Mass [kg]
  m_TPS(j) = 0.1513*m_MLV_total(j);

  m_0_CTV(j) = CTVmass_final(m_f_CTV,Isp_CTV,m_MLV_total(j),m_0_ARV,...
    delV_Edep,delV_capt(j),delV_at1(j), delV_at2(j), delV_dep(j),...
    delV_Ecapt);
  m_prop_CTV(j) = m_0_CTV(j) - m_f_CTV;
end

optimal = find(m_0_CTV-min(m_0_CTV) ==0);

%%%% Output Section
fprintf('\nOptimal MPO Period (of those considered): %3.1f Days \n', P(optimal))
fprintf('Earth Departure Delta-V: %6.4f km/s \n',delV_Edep)
fprintf('Mars Capture Delta-V: %6.4f km/s \n',delV_capt(optimal))
fprintf('Apo-Twist 1 Delta-V: %6.4f km/s \n',delV_at1(optimal))
fprintf('Apo-Twist 2 Delta-V: %6.4f km/s \n',delV_at2(optimal))
fprintf('Mars Departure Delta-V: %6.4f km/s \n',delV_dep(optimal))
fprintf('Earth Capture Delta-V: %6.4f km/s \n', delV_Ecapt)
fprintf('Total CTV Delta-V: %6.4f km/s \n', [delV_capt(optimal)+delV_at1(optimal)+delV_at2(optimal)+delV_dep(optimal)+delV_Edep+delV_Ecapt])
fprintf('CTV Propellant Mass: %10.2f kg \n', m_prop_CTV(optimal))
fprintf('GROSS CTV Mass: %10.2f kg \n\n', m_0_CTV(optimal))
fprintf('MLV Deorbit Delta-V: %6.4f km/s \n', delV_deorb(optimal))
fprintf('MLV Rendezvous Delta-V: %6.4f km/s \n', delV_rndvs(optimal))
fprintf('Total MLV Delta-V: %6.4f km/s \n', (delV_deorb(optimal)+delV_rndvs(optimal)))
fprintf('MLV Propellant Mass: %10.2f kg \n', m_prop_MLV(optimal))
fprintf('GROSS MLV Mass: %10.2f kg \n\n\n', m_MLV_total(optimal))
fprintf('MLV TPS Mass: %10.2f kg \n', m_TPS(optimal))
fprintf('MLV Parachute Mass: %10.2f kg \n', m_chutes(optimal))
fprintf('MLV Descent Stage Inert Mass: %10.2f kg \n', m_in1(optimal))
fprintf('MLV Descent Stage Propellant Mass: %10.2f kg \n', m_prop1(optimal))
fprintf('MLV Ascent Stage Inert Mass: %10.2f kg \n', m_in2(optimal))
fprintf('MLV Ascent Stage Propellant Mass: %10.2f kg \n', m_prop2(optimal))
fprintf('MLV Crew Compartment Mass: %10.2f kg \n\n\n\n', m_f_MLV)

nonopt = find(P-1 ==0);
fprintf('Near Optimal MPO Period: 1 Day\n')
fprintf('Earth Departure Delta-V: %6.4f km/s \n',delV_Edep)
fprintf('Mars Capture Delta-V: %6.4f km/s \n',delV_capt(nonopt(1)))
fprintf('Apo-Twist 1 Delta-V: %6.4f km/s \n',delV_at1(nonopt(1)))
fprintf('Apo-Twist 2 Delta-V: %6.4f km/s \n',delV_at2(nonopt(1)))
fprintf('Mars Departure Delta-V: %6.4f km/s \n',delV_dep(nonopt(1)))
fprintf('Earth Capture Delta-V: %6.4f km/s \n', delV_Ecapt)
fprintf('Total CTV Delta-V: %6.4f km/s \n',
[delV_capt(nonopt(1))+delV_at1(nonopt(1))+delV_at2(nonopt(1))+delV_dep(nonopt(1))+delV_Edep+delV_Ecapt])
fprintf('CTV Propellant Mass: %10.2f kg \n', m_prop_CTV(nonopt(1)))
fprintf('GROSS CTV Mass: %10.2f kg \n\n', m_0_CTV(nonopt(1)))
fprintf('MLV Deorbit Delta-V: %6.4f km/s \n', delV_deorb(nonopt(1)))
fprintf('MLV Rendezvous Delta-V: %6.4f km/s \n', delV_rndvs(nonopt(1)))
fprintf('Total MLV Delta-V: %6.4f km/s \n', (delV_deorb(nonopt(1))+delV_rndvs(nonopt(1))))
fprintf('MLV Propellant Mass: %10.2f kg \n', m_prop_MLV(nonopt(1)))
fprintf('GROSS MLV Mass: %10.2f kg \n\n\n', m_MLV_total(nonopt(1)))
fprintf('MLV TPS Mass: %10.2f kg \n', m_TPS(nonopt(1)))
fprintf('MLV Parachute Mass: %10.2f kg \n', m_chutes(nonopt(1)))
fprintf('MLV Descent Stage Inert Mass: %10.2f kg \n', m_in1(nonopt(1)))
fprintf('MLV Descent Stage Propellant Mass: %10.2f kg \n', m_prop1(nonopt(1)))
fprintf('MLV Ascent Stage Inert Mass: %10.2f kg \n', m_in2(nonopt(1)))
fprintf('MLV Ascent Stage Propellant Mass: %10.2f kg \n', m_prop2(nonopt(1)))
fprintf('MLV Crew Compartment Mass: %10.2f kg \n\n\n\n', m_f_MLV)

figure(1)
```

```
    h=plot(P,m_MLV_total,'b-');
    set(h,'linewidth',2); hold on; grid on;
    title('MLV Mass vs. Mars Parking Orbit Period')
    xlabel('MPO Period [Martian days]')
    ylabel('MLV Gross Mass [kg]')

    figure(2)
    h=plot(P,m_0_CTV,'r-');
    set(h,'linewidth',2); hold on; grid on;
    title('CTV Launch Mass vs. Mars Parking Orbit Period')
    xlabel('MPO Period [Martian days]')
    ylabel('CTV Mass [kg]')

    figure(3)
    h=plot(P, (delV_capt+delV_at1+delV_at2+delV_dep+delV_Edep+delV_Ecapt),'g-', P, m_prop_CTV/10000, 'k--');
    set(h,'linewidth',2); hold on; grid on;
    title('CTV \Delta V & m_p_r_o_p vs. Mars Parking Orbit Period')
    xlabel('MPO Period [Martian days]')
    ylabel('CTV \Delta V [km/s], CTV m_p_r_o_p (x10^-^4) [kg]')
    legend('CTV \Delta V', 'CTV m_p_r_o_p')



function [delV_capt, delV_at1, delV_at2, delV_dep, ...
  delV_deorb, delV_rndvs] = MPOfinal(P, a_c, incl, a_d)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: MPOfinal.m
% Mars Parking Orbit Delta V Requirements
% Author: George E. Pollock
% Last Modified on March 27, 2005 by George E. Pollock
%
% THIS FUNCTION COMPUTES THE DELTA-V'S FOR BOTH THE CTV AND MLV FOR THE
% MISSION MANEUVERS DETAILED BELOW.
%
% OVERALL MISSION DESCRIPTION:
% Hyperbolic arrival at Mars, capture into a "Mars Parking Orbit" of
% period P Martian Days then apo-twist into the Martian Equatorial Plane,
% perform lander deorbit maneuver, rendezvous following completion of lander
% mission, pre-departure apo-twist maneuver, Mars departure.
%
% NOTE: This treatment of the problem
% assumes that the arrival trajectory can be targeted such that the
% argument of periapsis is zero and that the maximum inclination change
% required is modeled in the inclination changes given by incl (an input to
% the funtion). Additionally, impulsive, tangential maneuvers are
% assumed (except for plane changes, which are inherently not tangential).
%
% INPUTS: Period of the Mars Parking Orbit (in Martian Days, 24.6229 hours)
% hyperbolic arrival semi-major axis (km), inclination change relative to the
% Martian equatorial plane (deg) (used for the magnitude of both plane changes
% for the apo-twist maneuvers), and hyperbolic departure semi-major axis
% (km).
%
% OUTPUT: CTV Delta V's for Capture into MPO of specified period, Apo-Twist #1,
% Apo-Twist #2, and Departure; Lander Delta V's for deorbit burn and
% rendezvous burn to get on elliptical parking orbit from 350 km altitude
% circular orbit.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% UNIT CONVERSIONS
P = P*24.6229*3600;         % sec
incl = deg2rad(incl);   % rad

% CONSTANTS
mu_mars = 4.282828e4;   % km^3/s^2
R_mars = 3397;          % equatorial radius of Mars [km]

% MARS PARKING ORBIT
% periapsis radius of parking orbit [km]
r_p = 350 + R_mars;

% semi-major axis of Mars Parking Orbit [km]
a_po = [(P/(2*pi))^2*mu_mars]^(1/3);

% eccentricity of Mars Parking Orbit
e_po = 1 - r_p/a_po;

% periapsis and apoapsis radii of Mars Parking Orbit (km)
r_p = a_po*(1-e_po);
r_a = a_po*(1+e_po);

% velocities on Mars Parking Orbit at periapsis and apoapsis (km/s)
v_a = sqrt(2*mu_mars/r_a - mu_mars/a_po);
v_p = sqrt(2*mu_mars/r_p - mu_mars/a_po);

% HYPERBOLIC ARRIVAL AND DEPARTURE ORBITS
% velocities of CTV at arrival and departure (periapsis of both the
% hyperbolic approach/departure orbit and the Mars Parking Orbit).
v_arr = sqrt(2*mu_mars/r_p - mu_mars/a_c);
v_dep = sqrt(2*mu_mars/r_p - mu_mars/a_d);
```

```
% Capture Delta V from hyperbolic approach to Loose Mars Capture Orbit
delV_capt = abs(v_p - v_arr);

% Plane Change Delta V at apoapsis of Loose Mars Capture Orbit to crank
% LMCO into Martian Equatorial Plane
delV_at1 = 2*v_a*sin(abs(incl)/2);

% Plane Change Delta V at apoapsis of a Loose Mars Capture Orbit in the
% EQUATORIAL PLANE to change inclination to desired departure conditions
delV_at2 = 2*v_a*sin(abs(incl)/2);

% Delta V for Departure at Mars to return to Earth
delV_dep = v_dep - v_p;

% ENTRY ORBIT  and DELTA-Vs FOR THE MLV
% desired entry flight path angle (deg) at entry interface
%     (125 km altitude)
gam_e = -8;
gam_e = deg2rad(gam_e);

% conduct de-orbit burn at apoapsis of Equatorial Loose Mars Capture Orbit
r_a_e = r_a;

% initial guess at r_p (km)
r_p_e = 20 + R_mars;

% entry interface altitude (km)
alt_e = 103;

a_e = 1/2*(r_p_e+r_a_e);        % semi-major axis of entry orbit (km)
r_e = R_mars + alt_e;           % entry radius (km)
v_e = sqrt(2*mu_mars/r_e - mu_mars./a_e);      % entry vel. (km/s)

e_e = 1 - r_p_e/a_e;
p_e = a_e*(1-e_e^2);

gam_temp = -acos(sqrt(mu_mars.*p_e)./(r_e.*v_e));

% iterate on r_p_e (entry periapsis radius) to find orbit that satisfies
% desired entry interface conditions (altitude and flight path angle)
while abs(gam_temp - gam_e) > 0.000001
    S = sign(gam_temp - gam_e);
    r_p_e = r_p_e-S*0.00001*r_p_e;
    a_e = 1/2*(r_p_e+r_a_e);     % semi-major axis of entry orbit (km)
    v_e = sqrt(2*mu_mars/r_e - mu_mars./a_e);      % entry vel. (km/s)
    e_e = 1 - r_p_e/a_e;
    p_e = a_e*(1-e_e^2);
    gam_temp = -acos(sqrt(mu_mars.*p_e)./(r_e.*v_e));
end
gam_entry = gam_temp;

% Deorbit delta V, assuming tangential, in-plane, impulsive maneuver at
% apoapsis of Mars Parking Orbit
v_a_e = sqrt(2*mu_mars/r_a - mu_mars./a_e);
delV_deorb = v_a-v_a_e;

% ENTRY CONDITIONS OUTPUT SECTION
% fprintf('\n\nENTRY CONDITIONS FOR ARES LANDER:\n')
% fprintf('Entry altitude: %4.1f km\n', alt_e)
% fprintf('Entry velocity: %7.5f km/s\n', v_e)
% fprintf('Entry flight path angle: %7.4f degrees\n', rad2deg(gam_entry))
% fprintf('Deorbit Delta-V: %7.5f m/s\n', delV_deorb*1000)
% fprintf('Deorbit semi-major axis: %10.4f km\n', a_e)
% fprintf('Deorbit eccentricity: %8.6f \n', e_e)

% Delta V for rendezvous maneuver from 350km circular orbit to Mars Parking
% Orbit
v_c = sqrt(mu_mars/(R_mars+350));
delV_rndvs = v_p - v_c;
return
function [m_MLV, m_in1, mprop1, m_in2, mprop2] = MLVmass_final(m_payload, delVdeorbit, ...
   delVrend, Isp_1, Isp_2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AAE 450 - Spring 2005
% Filename: MLVmass_final.m
% Mars Lander Vehicle Mass Calculation Function
% Author: George E. Pollock
% Last Modified on March 27, 2005 by George E. Pollock
%
% Acknowledgments: this function is based upon Robert Anderson's ideal rocket
% equation analysis employed in his script 'staging.m' code for the Mars
% Lander descent and ascent propulsion system.
%
% THIS FUNCTION COMPUTES THE GROSS MASS OF THE MARS LANDER VEHICLE, BASED
% UPON THE IDEAL ROCKET EQUATION AND INPUT PROPULSION SYSTEM PERFORMANCE
% NUMBERS, PAYLOAD MASS, AND DELTA-V REQUIREMENTS.
%
% INPUTS: Crew compartment (payload) mass [kg], deorbit delta-V [m/s],
% rendezvous delta-V [m/s], Isp values [sec] for the descent stage, and
% ascent stage
```

```
%
% OUTPUT: Gross mass [kg] of the MLV at prior to undocking from CTV in Mars
% orbit (not counting Thermal Protection System, Reaction Control Systems,
% or Parachute mass!), inert and propellant masses for the descent and
% ascent stages, respectively [in kg].
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
del_v1=840+delVdeorbit;        % velocity increment of stage 1 [m/s]
del_v2=(4126+109+delVrend);    % velocity increment of stage 2 [m/s]
g0=9.81;                       % gravitational constant [m/s^2]

% CHANGED THE PROPELLANT MASS FRACTION ON 2/14/2005 TO A LESS AGGRESSIVE
% 0.80 and 0.85 (FROM ITS PREVIOUS VALUES OF 0.87 for both stages)
lambda_1=0.80;                 % propellant mass fraction for descent stage
lambda_2=0.85;                 % propellant mass fraction for ascent stage
m_pl=m_payload;                  % payload mass [kg]
descent_struct = 518;          % descent structure mass [kg]

% Single Stage Descent followed by Single Stage Ascent
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Ascent Stage
MR_2=exp((del_v2)/(Isp_2*g0));
mprop2=lambda_2.*m_pl.*(MR_2-1)./(1-MR_2.*(1-lambda_2));
m_in2=mprop2.*(1-lambda_2)./lambda_2;
m_stage2=m_pl+m_in2+mprop2;

% Descent Stage
MR_1=exp(del_v1/(Isp_1*g0));
mprop1=lambda_1.*(m_stage2+descent_struct).*(MR_1-1)./(1-MR_1.*(1-lambda_1));
m_in1=mprop1.*(1-lambda_1)./lambda_1;
m_stage1=m_stage2+descent_struct+m_in1+mprop1;

inert_stage2_2=m_in2';
inert_stage1_2=m_in1';
mprop_total_2=(mprop1+mprop2)';
m_MLV=m_stage1';

return
```

```
function m_0_CTV = CTVmass_final(m_f_CTV,Isp_CTV,m_0_MLV,m_0_ARV,delV_Edep,delV_capt,delV_at1, ...
    delV_at2,delV_dep,delV_Ecapt)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% CTV Propellant Mass Function
% Author: George E. Pollock
% Last Modified on March 27, 2005 by George E. Pollock
%
% THIS FUNCTION COMPUTES THE GROSS MASS OF CTV PRIOR TO EARTH DEPARTURE
% USING THE IDEAL ROCKET EQUATION, WORKING FROM THE FINAL CTV MASS AT EARTH
% CAPTURE (COMPLETION OF A NOMINAL MISSION) BACKWARDS TO OBTAIN THE INITIAL
% MASS OF THE CTV.
%
% INPUTS: Empty mass of CTV [kg], gross MLV mass [kg], gross ARV mass [kg],
% CTV Delta V's for Earth Departure, Mars Capture, Apo-Twist #1, Apo-Twist #2,
% Mars Departure, and Earth Capture [all km/s]
%
% OUTPUT: Gross mass of CTV prior to Earth departure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Assemble all Delta-V data into a single vector % m/s
DELTA_V = [delV_Edep,delV_capt,delV_at1,delV_at2,delV_dep,delV_Ecapt]*10^3;

% Initialize vector to store masses at each phase of mission
CTV_MASS = zeros(1,length(DELTA_V)+1);
CTV_MASS(length(DELTA_V)+1) = m_f_CTV;

% Vector denoting the payload mass due to the presence or absence of the
% MLV and ARV at the CTV at all phases of mission
M_PAYLOAD = m_0_MLV*[1,1,1,0,0,0,0] + m_0_ARV*[1,1,1,1,1,1,0];

% gravitational constant [m/s^2]
g0 = 9.81;

% Work backward from post-Earth-Capture on return from nominal mission
% using ideal Rocket Equation to determine masses.
for k = length(DELTA_V):-1:1
  CTV_MASS(k) = [CTV_MASS(k+1)+M_PAYLOAD(k)]*exp(DELTA_V(k)/(Isp_CTV*g0));
end

m_0_CTV = CTV_MASS(1);

return
```

34.2 Lambert Problem Appendix

Author: George Pollock

34.2.1 Lambert Problem Description

The Lambert time of flight problem is a classic problem in orbit mechanics. We develop this treatment of the problem to compute interplanetary trajectories from Earth to Mars and back. In particular, this algorithm is used to validate trajectories obtained using MIDAS and provide departure and arrival conditions for the various mission opportunities.

34.2.2 Lambert Problem Inputs

The functions in this code take departure and arrival dates as inputs. The choice of function (free_rtn.m or low_energy_rtn.m) indicates whether the trajectory arc being determined is Earth-to-Mars or Mars-to-Earth (respectively). From this code, we obtain the solution for the Lambert arc, with all orbital elements (in the Sun-Centered Mean Ecliptic of J2000 coordinate frame) along with planetary departure and arrival conditions for the CTV.

34.2.3 Lambert Problem Code

```
function [a,incl,v_inf_earth_launch,v_inf_mars_arr] = free_rtn(T1,T2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AAE 450 - Spring 2005
% Filename: free_rtn.m
% Lambert Problem Script for:
% ANALYSIS OF OUTBOUND LEG OF A MARS FREE-RETURN TRAJECTORY
% Author: George E. Pollock
% Last Modified on February 14, 2005 by George E. Pollock
%
% This script solves the Lambert Problem using code contained in
% the 8 functions: lambert.m, transfer.m, parab.m, ellps.m, hyprb.m,
% theta_star.m, orb2in.m, and body313_qc.m
% The main function is lambert.m, calling this will in turn call the other
% functions as needed in solving the Lambert problem.  This script requires
% the position and velocity vectors of the departure and arrival planets
% which it reads from the spreadsheet that is input as 'file'.  To analyze
% additional cases, the planet state vectors and time of flight must be
% changed in the spreadsheet to reflect the free return time of flight
% and Earth's state at launch & Mars' state at arrival.
%
% Notes on using this code:
%
```

```
% The original author strongly recommends using a
% Sun-Centered Mean Ecliptic of J2000 coordinate frame for input of state
% data for the planets of interest.  This system lends itself to an easier
% visualization of the resulting velocities which this script computes in
% the same inertial frame used for the planet data in the spreadsheet.
%
% Due to the brute-force iteration on 'a' in the ellps.m and hyprb.m
% functions, the code can take 30 seconds - 1 minute or more per arc.  This
% time can be reduced if the precision of the iteration process is lowered
% by altering that portion of the code.
%
% Finally, of great interest are the outputs printed to the screen in the
% lambert.m function, the inertial velocities at each end of the arc, and the
% V_infinity relative to each planet.
% These values are currently displayed by running the code.  Additional
% values of interest can be output by un-suppressing printing of these
% values in the functions where they are computed.
%
% Please comment all modifications to this set of code and change the
% header to record the date of last revision and who performed the
% revision.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long;

global mu
mu = 1.3271244e11;          % Sun [km^3/s^2]

% T1 = Departure time [Julian Date]
% T2 = Arrival time [Julian Date]

R1 = jdate(T1,1);    % Earth at launch
R2 = jdate(T2,3);    % Mars at arrival

V1 = jdate(T1,2);    % Earth at launch
V2 = jdate(T2,4);    % Mars at arrival

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FREE-RETURN EARTH-MARS OUTBOUND TRANSFER ARC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('OUTBOUND EARTH-MARS FREE-RETURN ARC')

% Dates must be expressed in Julian dates
TOF1 = T2 - T1;         % days
TOF1 = TOF1*24*3600;    % seconds
```

```
% orbital angular momentum vector
h_0 = cross(R1,V1);
h_hat1 = cross(R1,R2)/norm(cross(R1,R2));

% Quad Check for Transfer Angle
TA1_a = mod(asin(norm(cross(R1,R2))/(norm(R1)*norm(R2))),2*pi);
TA1_b = mod(pi-asin(norm(cross(R1,R2))/(norm(R1)*norm(R2))),2*pi);
TA1_c = mod(acos(dot(R1,R2)/(norm(R1)*norm(R2))),2*pi);
TA1_d = mod(-acos(dot(R1,R2)/(norm(R1)*norm(R2))),2*pi);

if (abs(TA1_a - TA1_c) < 0.001)
   TA1 = TA1_a;
elseif (abs(TA1_a - TA1_d) < 0.001)
   TA1 = TA1_a;
elseif (abs(TA1_b - TA1_c) < 0.001)
   TA1 = TA1_b;
elseif (abs(TA1_b - TA1_d) < 0.001)
   TA1 = TA1_b;
else
   disp('Error determining Transfer Angle')
end

% enforce ang. mom. vector to have same sign of z-component as initial
% angular momentum vector at Earth
if sign(h_hat1(3)) ~= sign(h_0(3))
   h_hat1 = -h_hat1;
   TA1 = 2*pi-TA1;
end

% Convert Transfer Angle to degrees and call lambert function to solve
% Lambert's problem.
TA1 = rad2deg(TA1);
[OUT1, arc1_type] = lambert(R1, R2, TA1, TOF1, h_hat1);

% Direction Cosine Matrix between orbit frame and inertial frame at Earth
% departure (after maneuver to get on transfer arc)
C_earth_launch = orb2in(OUT1(4), OUT1(5), OUT1(6)+OUT1(10));
% Velocity in orbit frame (r_hat, theta_hat, h_hat) [km/s]
V_earth_launch_rt = OUT1(8)*[sin(deg2rad(OUT1(9))); cos(deg2rad(OUT1(9))); 0];
% Velocity in inertial frame (x_hat, y_hat, z_hat) [km/s]
V_earth_launch_in = C_earth_launch*V_earth_launch_rt;
% V_infinity at Earth departure [km/s]
V_inf_earth_launch = V_earth_launch_in - V1';
% magnitude of V_infinity at arrival [km/s]
v_inf_earth_launch = norm(V_inf_earth_launch)
```

```
% Direction Cosine Matrix between orbit frame and inertial frame at MARS
% arrival (before any propulsive or gravitational delta V)
C_mars_arr = orb2in(OUT1(4), OUT1(5), OUT1(6)+OUT1(14));
% Velocity in orbit frame (r_hat, theta_hat, h_hat) [km/s]
V_mars_arr_rt = OUT1(12)*[sin(deg2rad(OUT1(13))); cos(deg2rad(OUT1(13))); 0];
% Velocity in inertial frame (x_hat, y_hat, z_hat) [km/s]
V_mars_arr_in = C_mars_arr*V_mars_arr_rt;
% V_infinity at planetary arrival [km/s]
V_inf_mars_arr = V_mars_arr_in - V2';
% magnitude of V_infinity at arrival [km/s]
v_inf_mars_arr = norm(V_inf_mars_arr)

% transfer arc inclination relative to the ecliptic plane [degrees]
incl = rad2deg(acos(dot(h_hat1,[0,0,1])));

% semi-major axis of hyperbolic arrival orbit at Mars [km]
a = -4.282828e4/v_inf_mars_arr^2;

return
```

```
function [a,incl,v_inf_mars_dep,v_inf_earth_arr] = low_energy_rtn(T3,T4)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: low_energy_rtn.m
% Lambert Problem Function for:
% ANALYSIS OF LOW ENERGY RETURN ARC FROM MARS TO EARTH
% Author: George E. Pollock
% Last Modified on February 19, 2005 by George E. Pollock
%
% Comments: same approach to solving the Lambert problem as free_rtn.m, but
% modified for the case of the low energy return arc from Mars to Earth.
% Note: this algorithm does not identify the optimal low-energy return, it
% simply solves the Lambert problem from the input dates for Mars departure
% and Earth arrival.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long;
global mu
mu = 1.3271244e11;          % Sun [km^3/s^2]

% T3 = Departure time [Julian Date]
% T4 = Arrival time [Julian Date]

R1 = jdate(T3,3);     % Mars at departure [km]
R2 = jdate(T4,1);     % Earth at arrival [km]

V1 = jdate(T3,4);     % Mars at departure [km/s]
V2 = jdate(T4,2);     % Earth at arrival [km/s]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOW-ENERGY MARS-EARTH RETURN TRANSFER ARC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('LOW ENERGY MARS-EARTH RETURN TRAJECTORY')

% Dates must be expressed in Julian dates
TOF1 = T4 - T3;       % days
TOF1 = TOF1*24*3600;   % seconds

% orbital angular momentum vector
h_0 = cross(R1,V1);
h_hat1 = cross(R1,R2)/norm(cross(R1,R2));

% Quad Check for Transfer Angle
```

```
TA1_a = mod(asin(norm(cross(R1,R2))/(norm(R1)*norm(R2))),2*pi);
TA1_b = mod(pi-asin(norm(cross(R1,R2))/(norm(R1)*norm(R2))),2*pi);
TA1_c = mod(acos(dot(R1,R2)/(norm(R1)*norm(R2))),2*pi);
TA1_d = mod(-acos(dot(R1,R2)/(norm(R1)*norm(R2))),2*pi);

if (abs(TA1_a - TA1_c) < 0.001)
   TA1 = TA1_a;
elseif (abs(TA1_a - TA1_d) < 0.001)
   TA1 = TA1_a;
elseif (abs(TA1_b - TA1_c) < 0.001)
   TA1 = TA1_b;
elseif (abs(TA1_b - TA1_d) < 0.001)
   TA1 = TA1_b;
else
   disp('Error determining Transfer Angle')
end

% enforce ang. mom. vector to have same sign of z-component as initial
% angular momentum vector at Earth
if sign(h_hat1(3)) ~= sign(h_0(3))
   h_hat1 = -h_hat1;
   TA1 = 2*pi-TA1;
end

% Convert Transfer Angle to degrees and call lambert function to solve
% Lambert's problem.
TA1 = rad2deg(TA1);
[OUT1, arc1_type] = lambert(R1, R2, TA1, TOF1, h_hat1);

% Direction Cosine Matrix between orbit frame and inertial frame at MARS
% departure (after maneuver to get on transfer arc)
C_mars_dep = orb2in(OUT1(4), OUT1(5), OUT1(6)+OUT1(10));
% Velocity in orbit frame (r_hat, theta_hat, h_hat) [km/s]
V_mars_dep_rt = OUT1(8)*[sin(deg2rad(OUT1(9))); cos(deg2rad(OUT1(9))); 0];
% Velocity in inertial frame (x_hat, y_hat, z_hat) [km/s]
V_mars_dep_in = C_mars_dep*V_mars_dep_rt;
% V_infinity at Earth departure [km/s]
V_inf_mars_dep = V_mars_dep_in - V1';
% magnitude of V_infinity at departure [km/s]
v_inf_mars_dep = norm(V_inf_mars_dep)

% Direction Cosine Matrix between orbit frame and inertial frame at EARTH
% arrival (before any propulsive or gravitational delta V)
C_earth_arr = orb2in(OUT1(4), OUT1(5), OUT1(6)+OUT1(14));
% Velocity in orbit frame (r_hat, theta_hat, h_hat) [km/s]
V_earth_arr_rt = OUT1(12)*[sin(deg2rad(OUT1(13))); cos(deg2rad(OUT1(13))); 0];
```

```
% Velocity in inertial frame (x_hat, y_hat, z_hat) [km/s]
V_earth_arr_in = C_earth_arr*V_earth_arr_rt;
% V_infinity at planetary arrival [km/s]
V_inf_earth_arr = V_earth_arr_in - V2';
% magnitude of V_infinity at arrival [km/s]
v_inf_earth_arr = norm(V_inf_earth_arr)

% transfer arc inclination relative to the ecliptic plane [degrees]
incl = rad2deg(acos(dot(h_hat1,[0,0,1])));

% semi-major axis of hyperbolic departure orbit at Mars [km]
a = -4.282828e4/v_inf_mars_dep^2;

return
```

```
function vector_out = jdate(JDATE, flag)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AAE 450 - Spring 2005
% Filename: jdate.m
% Planet State Data/Julian Date Function
% Author: George E. Pollock
% Last Modified on February 14, 2005 by George E. Pollock
%
% THIS FUNCTION PROVIDES PLANET STATE DATA FOR EARTH AND MARS (POSITION AND
% VELOCITY) FOR SPECIFIED JULIAN DATE.  Special thanks to Eric Gustafson for
% extracting the data from STK in 1-day intervals in the Sun Centered Mean
% Ecliptic of J2000 coordinate system.

% INPUTS: Julian date of interest, a flag denoting which vector is to be
% provided for the input date:
%   1 = Earth position
%   2 = Earth velocity
%   3 = Mars position
%   4 = Mars velocity
%
% OUTPUT: The requested vector, given in Sun Centered Mean Ecliptic of
% J2000 coordinates, at the input Julian date.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load earth_data.mat
load mars_data.mat
n = find(JD - JDATE ==0);
vector_out = zeros(1,3);
if flag == 1
   vector_out(1) = x_earth(n);
   vector_out(2) = y_earth(n);
   vector_out(3) = z_earth(n);
elseif flag == 2
   vector_out(1) = x_vel_earth(n);
   vector_out(2) = y_vel_earth(n);
   vector_out(3) = z_vel_earth(n);
elseif flag == 3
   vector_out(1) = x_mars(n);
   vector_out(2) = y_mars(n);
   vector_out(3) = z_mars(n);
elseif flag == 4
   vector_out(1) = x_vel_mars(n);
   vector_out(2) = y_vel_mars(n);
   vector_out(3) = z_vel_mars(n);
end
return
```

```
function [OUTPUT, type] = lambert(R1, R2, TA, TOF, h_hat)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: lambert.m
% Lambert Problem Main Function
% Author: George E. Pollock
% Last Modified on February 5, 2005 by George E. Pollock
%
% INPUTS: the vectors R1, R2 (in km), transfer angle (TA, in degrees),
% Time of Flight (TOF, in seconds), angular momentum unit vector
%
% OUTPUTS: A vector containing Transfer Angle [deg], semi-major axis [km],
% eccentricity, Right Ascension of the Ascending Node [deg], inclination
% [deg], argument of periapsis [deg], departure radius [km], velocity [km/s],
% flight path angle [deg], and true anomaly [deg], arrival radius [km],
% velocity [km/s], flight path angle [deg], and true anomaly [deg]; the
% transfer type (1A, 1B, 2A, 2B, 1H, or 2H)
%
% IMPORTANT NOTE: gravitational parameter (mu) should be a global variable
% defined in calling script and have units of km^3/s^2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global mu
type = char(2);

% Magnitudes of Position Vectors
r1 = norm(R1);    % [km]
r2 = norm(R2);    % [km]

% ************************************************************************
% WARNING: The logic of this code has NOT been verified for cases other
% than m = 0!  Do not use this without extensive verification for any case
% but the one given here: m = 0.
% ************************************************************************
% Number of Revolutions about Central Body
m = 0;

% Compute Parabolic Time of Flight for type determination (ellipse,
% hyperbola, or parabola)
[TOFpar, c, s, type(1)] = parab(r1,r2,TA);

if TOF >TOFpar
    species = 'ellips';
    fprintf('\nThe Transfer Orbit is an ELLIPSE.\n\n')
elseif TOF < TOFpar
    species = 'hyperb';
    fprintf('\nThe Transfer Orbit is a HYPERBOLA.\n\n')
elseif TOF == TOFpar
    species = 'parabo';
    fprintf('\nThe Transfer Orbit is a PARABOLA.\n\n')
end

% Continue computations based on ellipse or hyperbola, as determined above
if species == 'ellips'
    a_min = s/2;
    alpha_min = 2*asin(sqrt(s/(2*a_min)));
    beta_min = 2*asin(sqrt((s-c)/(2*a_min)));
    if type(1) == '1'
        TOFmin = sqrt(a_min^3/mu) * [2*pi*m+(alpha_min - sin(alpha_min)) - (beta_min - sin(beta_min))];
    elseif type(1) == '2'
        TOFmin = sqrt(a_min^3/mu) * [2*pi*m+(alpha_min - sin(alpha_min)) + (beta_min - sin(beta_min))];
    end

    if TOF > TOFmin
        type(2) = 'B';
    elseif TOF < TOFmin
        type(2) = 'A';
    end
end

if species == 'hyperb'
    type(2) = 'H';
    % set a_min and TOFmin to zero (variables not applicable to hyperbolic transfers)
    a_min = 0;
    TOFmin = 0;
end

[a, TOF_actual, p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = transfer(type, TOF, a_min, TOFmin, s, c, r1, r2, TA, m);

% Use r_hat, th_hat, h_hat orbit frame to recover the direction cosine
% matrix between the orbit and inertial frames.
r_hat = (R1/norm(R1));
th_hat = cross(h_hat,r_hat);
C(:,1) = r_hat';
C(:,2) = th_hat';
C(:,3) = h_hat';

% Use the DC matrix to obtain the inclination, Right Ascension of the
% Ascending Node, and Argument of Peripsis
```

```
incl = acos(C(3,3));
% value produced by acos will be in the range from 0<=incl<=180 [degrees],
% in accordance with convention for orbit inclination.
[RAAN, theta] = body313_qc(C(1,3), C(2,3), C(3,1), C(3,2), incl);
omega = theta-TSD;

OUTPUT = [TA, a, e, rad2deg(RAAN), rad2deg(incl), rad2deg(omega), ...
    rd, vd, rad2deg(gamd), rad2deg(TSD), ra, va, rad2deg(gama), rad2deg(TSA)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SCREEN OUTPUT SECTION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('The transfer angle (TA) is: %10.6f degrees\n', TA)
fprintf('The transfer is type: %c%c \n', type)
fprintf('The semimajor axis is: %10.4f km\n', a)
fprintf('The eccentricity is: %8.6f\n', e)
fprintf('The inclination is: %10.6f degrees\n', incl*180/pi)
fprintf('The RAAN is: %10.6f degrees\n', RAAN*180/pi)
fprintf('The argument of periapsis is: %10.6f degrees\n\n', omega*180/pi)

fprintf('The Time of Flight is: %10.4f sec\n', TOF_actual)
fprintf('The parameter p is: %10.4f km\n', p)

fprintf('Radius at departure: %10.4f km\n', rd)
fprintf('Velocity at departure: %10.4f km/s\n', vd)
fprintf('Flight Path Angle at departure: %10.4f degrees\n', gamd*180/pi)
fprintf('Theta Star at departure: %10.4f degrees\n\n', TSD*180/pi)
fprintf('Radius at arrival: %10.4f km\n', ra)
fprintf('Velocity at arrival: %10.4f km/s\n', va)
fprintf('Flight Path Angle at arrival: %10.4f degrees\n', gama*180/pi)
fprintf('Theta Star at arrival: %10.4f degrees\n\n', TSA*180/pi)

return
```

```
function [TOFpar, c, s, type] = parab(r1, r2, TA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: parab.m
% Function to compute the semi-perimeter of the space triangle and the
% parabolic TOF, given inputs of r1, r2 (magnitudes, in km) and transfer angle
% (TA) in degrees.
% Author: George E. Pollock
% Last Modified on January 14, 2005 by George E. Pollock
%
% Function to compute the semi-perimeter of the space triangle and the
% parabolic TOF, given inputs of r1, r2 (magnitudes, in km) and transfer angle
% (TA) in degrees.
%
% INPUTS: position vector magnitudes r1 and r2 [km], Transfer Angle [in degrees]
%
% OUTPUTS: parabolic Time of Flight [sec], chord (c) and semi-perimeter (s) of
% space triangle [in km], type 1 or 2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% gravitational parameter of attracting body, initialized in calling
% script.  must have units km^3/s^2.
global mu

c = sqrt(r1^2 + r2^2 - 2*r1*r2*cos(TA*pi/180));
s = 1/2 * (r1+r2+c);

if TA < 180
    TOFpar = 1/3* sqrt(2/mu)*[s^(3/2) - (s-c)^(3/2)];
    type = '1';
elseif TA > 180
    TOFpar = 1/3* sqrt(2/mu)*[s^(3/2) + (s-c)^(3/2)];
    type = '2';
end

return
```

```
function [a, TOF_actual, p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = transfer(type, TOF, a_min, TOFmin, s, c, r1, r2, TA, m)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: transfer.m
% Function to compute semi-major axis 'a' of transfer orbit.
% Author: George E. Pollock
% Last Modified on January 14, 2005 by George E. Pollock
%
% INPUTS: transfer type (1A, 1B, 2A, 2B, 1H, 2H), desired TOF [sec], minimum
% energy semi-major axis (a_min) [km] and TOF (TOFmin) [sec], semi-perimeter (s) and chord
% (c) of space triangle [km], position vector magnitudes r1 and r2 [km], and Transfer
% Angle [degrees]
%
% OUTPUTS: semi-major axis [km], actual Time of Flight [sec], semi-latus
% rectum (p) [km], eccentricity and position, velocity, flight path angle
% and true anomaly at both Departure and Arrival [km, km/s, rad, rad]
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Gravitational parameter of attracting body, initialized in calling
% script - must have units km^3/s^2.
global mu

if type == '1A'
   % guess initial value for 'a', then iterate to match desired TOF
   a_temp = a_min;
   TOF_temp = TOFmin;
   while TOF_temp > TOF
      alpha_0 = 2*asin(sqrt(s/(2*a_temp)));
      beta_0 = 2*asin(sqrt((s-c)/(2*a_temp)));
      alpha = alpha_0;
      beta = beta_0;
      TOF_temp = sqrt(a_temp^3/mu) * [2*m*pi + (alpha_0 - sin(alpha_0)) - (beta_0 - sin(beta_0))];
      if abs(TOF-TOF_temp)/TOF < 0.025
         a_temp = 1.000000001*a_temp;
      elseif ((abs(TOF-TOF_temp)/TOF > 0.025) && (abs(TOF-TOF_temp)/TOF < 0.15))
         a_temp = 1.00001*a_temp;
      elseif (abs(TOF-TOF_temp)/TOF > 0.15)
         a_temp = 1.001*a_temp;
      end
   end

   if abs(TOF-TOF_temp)/TOF >0.0000001
      disp('Tolerance Warning -- Possible Overshoot Error')
```

```
    end

    a = a_temp;
    TOF_actual = TOF_temp;

    [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = ellps(type, s, c, r1, r2, a, alpha, beta, TA);

elseif type == '1B'
    % guess initial value for 'a', then iterate to match desired TOF
    a_temp = a_min;
    TOF_temp = TOFmin;
    while TOF_temp < TOF
        alpha_0 = 2*asin(sqrt(s/(2*a_temp)));
        beta_0 = 2*asin(sqrt((s-c)/(2*a_temp)));
        alpha = 2*pi - alpha_0;
        beta = beta_0;
        TOF_temp = sqrt(a_temp^3/mu) * [2*m*pi + 2*pi - (alpha_0 - sin(alpha_0)) - (beta_0 - sin(beta_0))];
        if abs(TOF-TOF_temp)/TOF < 0.025
            a_temp = 1.000000001*a_temp;
        elseif ((abs(TOF-TOF_temp)/TOF > 0.025) && (abs(TOF-TOF_temp)/TOF < 0.15))
            a_temp = 1.00001*a_temp;
        elseif (abs(TOF-TOF_temp)/TOF > 0.15)
            a_temp = 1.001*a_temp;
        end
    end

    if abs(TOF-TOF_temp)/TOF >0.0000001
        disp('Tolerance Warning -- Possible Overshoot Error')
    end
    a = a_temp;
    TOF_actual = TOF_temp;

    [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = ellps(type, s, c, r1, r2, a, alpha, beta, TA);

elseif type == '2A'
    % guess initial value for 'a', then iterate to match desired TOF
    a_temp = a_min;
    TOF_temp = TOFmin;
    while TOF_temp > TOF
        alpha_0 = 2*asin(sqrt(s/(2*a_temp)));
        beta_0 = 2*asin(sqrt((s-c)/(2*a_temp)));
        alpha = alpha_0;
        beta = -beta_0;
        TOF_temp = sqrt(a_temp^3/mu) * [2*m*pi + (alpha_0 - sin(alpha_0)) + (beta_0 - sin(beta_0))];
        if abs(TOF-TOF_temp)/TOF < 0.025
            a_temp = 1.000000001*a_temp;
        elseif ((abs(TOF-TOF_temp)/TOF > 0.025) && (abs(TOF-TOF_temp)/TOF < 0.15))
            a_temp = 1.00001*a_temp;
        elseif (abs(TOF-TOF_temp)/TOF > 0.15)
            a_temp = 1.001*a_temp;
        end
    end

    if abs(TOF-TOF_temp)/TOF >0.0000001
        disp('Tolerance Warning -- Possible Overshoot Error')
    end

    a = a_temp;
    TOF_actual = TOF_temp;

    [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = ellps(type, s, c, r1, r2, a, alpha, beta, TA);

elseif type == '2B'
    % guess initial value for 'a', then iterate to match desired TOF
    a_temp = a_min;
    TOF_temp = TOFmin;
    while TOF_temp < TOF
        alpha_0 = 2*asin(sqrt(s/(2*a_temp)));
        beta_0 = 2*asin(sqrt((s-c)/(2*a_temp)));
        alpha = 2*pi - alpha_0;
        beta = -beta_0;
        TOF_temp = sqrt(a_temp^3/mu) * [2*m*pi + 2*pi - (alpha_0 - sin(alpha_0)) + (beta_0 - sin(beta_0))];
        if abs(TOF-TOF_temp)/TOF < 0.025
            a_temp = 1.000000001*a_temp;
        elseif ((abs(TOF-TOF_temp)/TOF > 0.025) && (abs(TOF-TOF_temp)/TOF < 0.15))
            a_temp = 1.00001*a_temp;
        elseif (abs(TOF-TOF_temp)/TOF > 0.15)
            a_temp = 1.001*a_temp;
        end
    end

    if abs(TOF-TOF_temp)/TOF >0.0000001
        disp('Tolerance Warning -- Possible Overshoot Error')
    end

    a = a_temp;
    TOF_actual = TOF_temp;

    [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = ellps(type, s, c, r1, r2, a, alpha, beta, TA);

elseif type == '1H'
```

```
    % guess initial value for 'a', then iterate to match desired TOF
    a_temp = -.0001;
    alpha1_0 = 2*asinh(sqrt(s/(2*abs(a_temp))));
    beta1_0 = 2*asinh(sqrt((s-c)/(2*abs(a_temp))));
    TOF_temp = sqrt(abs(a_temp)^3/mu) * [(sinh(alpha1_0) - alpha1_0) - (sinh(beta1_0) - beta1_0)];
    while TOF > TOF_temp
        alpha1_0 = 2*asinh(sqrt(s/(2*abs(a_temp))));
        beta1_0 = 2*asinh(sqrt((s-c)/(2*abs(a_temp))));
        alpha1 = alpha1_0;
        beta1 = beta1_0;
        TOF_temp = sqrt(abs(a_temp)^3/mu) * [(sinh(alpha1_0) - alpha1_0) - (sinh(beta1_0) - beta1_0)];
        if abs(TOF-TOF_temp)/TOF < 0.05
            a_temp = 1.000001*a_temp;
        elseif ((abs(TOF-TOF_temp)/TOF > 0.05) && (abs(TOF-TOF_temp)/TOF < 0.15))
            a_temp = 1.01*a_temp;
        elseif (abs(TOF-TOF_temp)/TOF > 0.15)
            a_temp = 1.1*a_temp;
        end

    end
    if abs(TOF-TOF_temp)/TOF >0.0000001
        disp('Tolerance Warning -- Possible Overshoot Error')
    end

    a = a_temp;
    TOF_actual = TOF_temp;

    [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = hyprb(type, s, c, r1, r2, a, alpha1, beta1, TA);


elseif type == '2H'

    % guess initial value for 'a', then iterate to match desired TOF
    a_temp = -.0001;
    alpha1_0 = 2*asinh(sqrt(s/(2*abs(a_temp))));
    beta1_0 = 2*asinh(sqrt((s-c)/(2*abs(a_temp))));
    alpha1 = alpha1_0;
    beta1 = -beta1_0;
    TOF_temp = sqrt(abs(a_temp)^3/mu) * [(sinh(alpha1_0) - alpha1_0) + (sinh(beta1_0) - beta1_0)];
    while TOF > TOF_temp
        alpha1_0 = 2*asinh(sqrt(s/(2*abs(a_temp))));
        beta1_0 = 2*asinh(sqrt((s-c)/(2*abs(a_temp))));
        alpha1 = alpha1_0;
        beta1 = -beta1_0;
        TOF_temp = sqrt(abs(a_temp)^3/mu) * [(sinh(alpha1_0) - alpha1_0) + (sinh(beta1_0) - beta1_0)];
        if abs(TOF-TOF_temp)/TOF < 0.05
```

```
            a_temp = 1.000001*a_temp;
        elseif ((abs(TOF-TOF_temp)/TOF > 0.05) && (abs(TOF-TOF_temp)/TOF < 0.15))
            a_temp = 1.01*a_temp;
        elseif (abs(TOF-TOF_temp)/TOF > 0.15)
            a_temp = 1.1*a_temp;
        end

    end
    if abs(TOF-TOF_temp)/TOF >0.0000001
        disp('Tolerance Warning -- Possible Overshoot Error')
    end

    a = a_temp;
    TOF_actual = TOF_temp;
    [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = hyprb(type, s, c, r1, r2, a, alpha1, beta1, TA);
end
return
```

```
function [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = ellps(type, s, c, r1, r2, a, alpha, beta, TA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: ellps.m
% Function to compute parameters of elliptical transfer orbits.
% Author: George E. Pollock
% Last Modified on January 14, 2005 by George E. Pollock
%
% INPUTS: transfer type (1A, 1B, 2A, 2B) and semi-perimeter (s) and chord (c)
% of space triangle, position vector magnitudes r1 and r2, semi-major axis (a),
% alpha, beta, Transfer Angle (TA)
%
% OUTPUTS: semi-latus rectum (p), eccentricity (e), radius, velocity,
% flight path angle, and true anomaly at Departure (rd, vd, gamd, TSD), radius,
% velocity, flight path angle, and true anomaly at Arrival (ra, va, gama, TSA)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global mu

% ellipse equations for p [km]
P(1) = [(4*a*(s-r1)*(s-r2))/c^2] * (sin((alpha + beta)/2))^2;
P(2) = [(4*a*(s-r1)*(s-r2))/c^2] * (sin((alpha - beta)/2))^2;

if (type == '1A')
   % 1A --> larger p value
   p = max(P);
elseif (type == '2B')
   % 2B --> larger p value
   p = max(P);
elseif (type == '1B')
   % 1B --> smaller p value
   p = min(P);
elseif (type == '2A')
   % 2A --> smaller p value
   p = min(P);
end

e = sqrt(1-p/a);
rd = r1;
vd = sqrt(2*mu/rd - mu/a);
ra = r2;
va = sqrt(2*mu/ra - mu/a);

% Obtain theta star values [radians]
```

```
[TSD, TSA] = theta_star(e, p, rd, ra, TA);

% Computation and quad check of flight path angles [radians] based on true
% anomaly values
gamd = acos(sqrt(mu*p)/(rd*vd));
if mod(TSD,2*pi) < pi
    % ascending
    gamd = gamd;
elseif mod(TSD,2*pi) > pi
    % descending
    gamd = -gamd;
end

gama = acos(sqrt(mu*p)/(ra*va));
if mod(TSA,2*pi) < pi
    % ascending
    gama = gama;
elseif mod(TSA,2*pi) > pi
    % descending
    gama = -gama;
end

return
```

```
function [p, e, rd, vd, gamd, TSD, ra, va, gama, TSA] = hyprb(type, s, c, r1, r2, a, alpha1, beta1, TA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: hyprb.m
% Function to compute parameters of hyperbolic transfer orbits.
% Author: George E. Pollock
% Last Modified on January 14, 2005 by George E. Pollock
%
% INPUTS: transfer type (1H, 2H) and semi-perimeter (s) and chord (c)
% of space triangle, position vector magnitudes r1 and r2, semi-major axis (a),
% alpha, beta, Transfer Angle (TA)
%
% OUTPUTS: semi-latus rectum (p), eccentricity (e), radius, velocity,
% flight path angle, and true anomaly at Departure (rd, vd, gamd, TSD), radius,
% velocity, flight path angle, and true anomaly at Arrival (ra, va, gama, TSA)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global mu

% hyperbola equations for p [km]
P(1) = [(4*abs(a)*(s-r1)*(s-r2))/c^2]*(sinh((alpha1 + beta1)/2))^2;
P(2) = [(4*abs(a)*(s-r1)*(s-r2))/c^2]*(sinh((alpha1 - beta1)/2))^2

if (type == '1H')
   % 1H --> larger p value
   p = max(P);
elseif (type == '2H')
   % 2H --> smaller p value
   p = min(P);
end

e = sqrt(p/abs(a) + 1);
rd = r1;
vd = sqrt(2*mu/rd + mu/abs(a));
ra = r2;
va = sqrt(2*mu/ra + mu/abs(a));

% Obtain True Anomaly Values [radians]
[TSD, TSA] = theta_star(e, p, rd, ra, TA);

% change True Anomalies in quadrants III and IV to have corresponding
% negative values
if TSD > pi
```

```
   TSD = TSD - 2*pi;
end
if TSA > pi
   TSA = TSA - 2*pi;
end

% Computation and quad check of flight path angles [radians] based on true
% anomaly values
gamd = acos(sqrt(mu*p)/(rd*vd));
if mod(TSD,2*pi) < pi
   % ascending
   gamd = gamd;
elseif mod(TSD,2*pi) > pi
   % descending
   gamd = -gamd;
end

gama = acos(sqrt(mu*p)/(ra*va));
if mod(TSA,2*pi) < pi
   % ascending
   gama = gama;
elseif mod(TSA,2*pi) > pi
   % descending
   gama = -gama;
end

return
```

```
function [TSD, TSA] = theta_star(e, p, rd, ra, TA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AAE 450 - Spring 2005
% Filename: theta_star.m
% Function to compute Theta Star (True Anomaly) values at departure and arrival
% of transfer orbits.
% Author: George E. Pollock
% Last Modified on January 14, 2005 by George E. Pollock
%
% INPUTS: eccentricity (e), semi-latus rectum (p) [km], radius at departure
% (rd) [km], radius at arrival (ra) [km], transfer angle (TA) [degrees]
%
% OUTPUTS: True Anomaly (theta star) values for Departure and Arrival (TSD,
% TSA respectively) [radians]
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% convert Transfer Angle to radians
TA_rad = pi/180*TA;

% Compute candidate Theta Star angles [radians]
% departure:
TSD1 = mod(acos(1/e * (p/rd - 1)), 2*pi);
TSD2 = mod(-TSD1,2*pi);

TSA1 = mod(acos(1/e * (p/ra - 1)), 2*pi);
TSA2 = mod(-TSA1,2*pi);

% Perform Quadrant Check:
if (abs(TSA1 - TSD1 - TA_rad) < 0.001)
   TSA = TSA1;
   TSD = TSD1;
else if (abs(TSA1 + (2*pi-TSD1) -TA_rad) < 0.001)
   TSA = TSA1;
   TSD = TSD1;
elseif (abs(TSA1 - TSD2 - TA_rad) < 0.001)
   TSA = TSA1;
   TSD = TSD2;
elseif (abs(TSA1 + (2*pi-TSD2) - TA_rad) < 0.001)
   TSA = TSA1;
   TSD = TSD2;
elseif (abs(TSA2 - TSD1 - TA_rad) < 0.001)
   TSA = TSA2;
   TSD = TSD1;
elseif (abs(TSA2 + (2*pi-TSD1) - TA_rad) < 0.001)
```

```
    TSA = TSA2;
    TSD = TSD1;
elseif (abs(TSA2 - TSD2 - TA_rad) < 0.001)
    TSA = TSA2;
    TSD = TSD2;
elseif (abs(TSA2 + (2*pi-TSD2) - TA_rad) < 0.001)
    TSA = TSA2;
    TSD = TSD2;
else
    disp('An Error Has Ocurred in Determining Theta Star Values!!!')
    TSA = NaN;
    TSD = NaN;
    end
end
```

```
function [t_1, t_3] = body313_qc(C13, C23, C31, C32, theta_2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AAE 450 - Spring 2005
% Filename: body313_qc.m
% Function to recover theta 1 and theta 3 for the body 3-1-3 Euler Angle Sequence
% Author: George E. Pollock
% Last Modified on January 14, 2005 by George E. Pollock
%
% INPUTS: C13, C23, C31, C32 (the direction cosine elements from the body 3-1-3
% rotation) and theta_2 is the middle rotation in RADIANS
% OUTPUTS: the first and third angles in the rotation sequence, in RADIANS
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t_1a = mod(asin(C13/(sin(theta_2))),2*pi);
t_1b = mod((pi - asin(C13/(sin(theta_2)))),2*pi);
t_1c = mod(acos(-C23/(sin(theta_2))),2*pi);
t_1d = mod(-acos(-C23/(sin(theta_2))),2*pi);
if (abs(t_1a - t_1c) < 0.001)
   t_1 = t_1a;
elseif (abs(t_1a - t_1d) < 0.001)
   t_1 = t_1a;
elseif (abs(t_1b - t_1c) < 0.001)
   t_1 = t_1b;
elseif (abs(t_1b - t_1d) < 0.001)
   t_1 = t_1b;
else
   disp('Close to singularity for theta_1 as theta_2 -> 0')
   t_1 = NaN;
end
t_3a = mod(asin(C31/(sin(theta_2))),2*pi);
t_3b = mod(pi-asin(C31/(sin(theta_2))),2*pi);
t_3c = mod(acos(C32/(sin(theta_2))),2*pi);
t_3d = mod(-acos(C32/(sin(theta_2))),2*pi);
if (abs(t_3a - t_3c) < 0.001)
   t_3 = t_3a;
elseif (abs(t_3a - t_3d) < 0.001)
   t_3 = t_3a;
elseif (abs(t_3b - t_3c) < 0.001)
   t_3 = t_3b;
elseif (abs(t_3b - t_3d) < 0.001)
   t_3 = t_3b;
else
   disp('Close to singularity for theta_3 as theta_2 -> 0')
   t_3 = NaN;
end
return
```

```
function C = orb2in(RAAN, incl, theta)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AAE 450 - Spring 2005
% Filename: orb2in.m
% Function to compute the Euler 3-1-3 direction cosine matrix relating the
% r-hat, theta-hat, h-hat (orbit frame) with the inertial x-hat, y-hat,
% z-hat frame.
% Author: George E. Pollock
% Last Modified on January 14, 2005 by George E. Pollock
%
% INPUTS: Right Ascension of the Ascending Node (RAAN), inclination (incl), and
% theta (the sum of the argument of periapsis [lowercase omega] and true
% anomaly [theta-star]  ALL ANGLES MUST BE INPUT IN DEGREES!!!
%
% OUTPUT: The direction cosine matrix, C.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% NOTE: theta is the angle given by the sum of the argument of periapsis
% (lowercase omega) and theta-star (true anomaly - the angle measured from the
% e-hat vector to the radial position vector in the CCW direction).

% conversion to radians for computations
RAAN = RAAN*pi/180;
incl = incl*pi/180;
theta = theta*pi/180;

C = zeros(3,3);
C(1,1) = cos(RAAN)*cos(theta) - sin(RAAN)*cos(incl)*sin(theta);
C(1,2) = -cos(RAAN)*sin(theta) - sin(RAAN)*cos(incl)*cos(theta);
C(1,3) = sin(RAAN)*sin(incl);
C(2,1) = sin(RAAN)*cos(theta)+cos(RAAN)*cos(incl)*sin(theta);
C(2,2) = -sin(RAAN)*sin(theta)+cos(RAAN)*cos(incl)*cos(theta);
C(2,3) = -cos(RAAN)*sin(incl);
C(3,1) = sin(incl)*sin(theta);
C(3,2) = sin(incl)*cos(theta);
C(3,3) = cos(incl);

return
```

34.3 Mars Lander Vehicle Mars Entry Orbit Appendix

Author: George Pollock

34.3.1 Mars Lander Vehicle Entry Orbit Description

Once the period of the MPO is selected at 1 Martian-day, we determine the orbit flown by the MLV from its deorbit burn at apoapsis of the CTV's MPO to entry interface.

34.3.2 MLV Entry Orbit Input/Output

This function requires the flight path angle and altitude at the targeted entry interface point. It determines the orbital elements for the entry orbit and outputs the entry velocity and magnitude of the deorbit maneuver.

34.3.3 MLV Entry Orbit Code

```
function [delV_deorb, v_entry] = entry_interface(gamma_entry, alt_entry)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AAE 450 - Spring 2005
% Filename: entry_interface.m
% Mars Parking Orbit Delta V Requirements
% Author: George E. Pollock
% Last Modified on March 21, 2005 by George E. Pollock
%
% THIS FUNCTION COMPUTES THE DEORBIT DELTA-V AND VELOCITY AT ENTRY
% INTERFACE FOR THE MLV.  THE FUNCTION ASSUMES A 1 DAY PERIOD MARS PARKING
% ORBIT WITH 350 km PERIAPSIS ALTITUDE.
%
% INPUTS: Desired entry flight path angle [a negative value, given in degrees]
% and entry altitude [in km].
%
% OUTPUTS: De-orbit Delta-V [in km/s] and velocity at entry interface
% [km/s]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% UNIT CONVERSIONS (1-Martian-day period parking orbit)
P = 24.6229*3600;             % sec

% CONSTANTS
mu_mars = 4.282828e4;   % km^3/s^2
R_mars = 3397;          % equatorial radius of Mars [km]

% MARS PARKING ORBIT
% periapsis radius of parking orbit [km]
r_p = 350 + R_mars;
```

```
% semi-major axis of Mars Parking Orbit [km]
a_po = [(P/(2*pi))^2*mu_mars]^(1/3);

% eccentricity of Mars Parking Orbit
e_po = 1 - r_p/a_po;

% periapsis and apoapsis radii of Mars Parking Orbit (km)
r_p = a_po*(1-e_po);
r_a = a_po*(1+e_po)

% velocities on Mars Parking Orbit at periapsis and apoapsis (km/s)
v_a = sqrt(2*mu_mars/r_a - mu_mars/a_po)
v_p = sqrt(2*mu_mars/r_p - mu_mars/a_po);

% ENTRY ORBIT  and DELTA-Vs FOR THE MLV
% desired entry flight path angle (deg) at entry interface
gamma_entry = deg2rad(gamma_entry);

% conduct de-orbit burn at apoapsis of Equatorial Loose Mars Capture Orbit
r_a_e = r_a;

% initial guess at r_p (km)
r_p_e = 20 + R_mars;

a_e = 1/2*(r_p_e+r_a_e);     % semi-major axis of entry orbit (km)
r_e = R_mars + alt_entry;     % entry radius (km)
v_e = sqrt(2*mu_mars/r_e - mu_mars./a_e);      % entry vel. (km/s)

e_e = 1 - r_p_e/a_e;
p_e = a_e*(1-e_e^2);

gam_temp = -acos(sqrt(mu_mars.*p_e)./(r_e.*v_e));

% iterate on r_p_e (entry periapsis radius) to find orbit that satisfies
% desired entry interface conditions (altitude and flight path angle)
while abs(gam_temp - gamma_entry) > 0.000001
    S = sign(gam_temp - gamma_entry);    % sign providing direction of iteration
    r_p_e = r_p_e-S*0.00001*r_p_e; % periapsis of entry orbit (km)
    a_e = 1/2*(r_p_e+r_a_e);     % semi-major axis of entry orbit (km)
    v_e = sqrt(2*mu_mars/r_e - mu_mars./a_e);      % entry vel. (km/s)
    e_e = 1 - r_p_e/a_e;           % eccentricity of entry orbit
    p_e = a_e*(1-e_e^2);           % the parameter p (km)
    gam_temp = -acos(sqrt(mu_mars.*p_e)./(r_e.*v_e));
end
gam_entry = gam_temp;
```

```
v_entry = v_e;
% Deorbit delta V, assuming tangential, in-plane, impulsive maneuver at
% apoapsis of Mars Parking Orbit
v_a_e = sqrt(2*mu_mars/r_a - mu_mars./a_e);
delV_deorb = v_a-v_a_e;

% ENTRY CONDITIONS OUTPUT SECTION
fprintf('\n\nENTRY CONDITIONS FOR MLV:\n')
fprintf('Entry interface altitude: %4.1f km\n', alt_entry)
fprintf('Entry velocity: %7.5f km/s\n', v_entry)
fprintf('Entry flight path angle: %7.4f degrees\n', rad2deg(gamma_entry))
fprintf('Deorbit Delta-V: %7.5f km/s\n', delV_deorb)

return
```

34.3.3.1 Apo-Twist Maneuver

Author: George Pollock

34.3.3.1.1 Maneuver Description

Apo-twist maneuvers (a special case of plane-change) rearrange the Crew Transport Vehicle (CTV) orbit geometry both for Mars Lander Vehicle (MLV) operations and the CTV's return to Earth. We perform two apo-twist maneuvers during the CTV's stay in Mars orbit—for which we have budgeted a total ΔV capability of 1.0 km/s. Without these maneuvers, the landing and departure portions of the mission require considerably more propellant.

34.3.3.1.2 Theory and Assumptions

According to Landau, Longuski, and Penzo [1,2], failure to account for the departure geometry can produce large underestimates in the CTV's Earth departure mass. In general, the parking orbit orientation does not (necessarily) permit a tangential burn to reach the departure asymptote of the Earth return transfer. Landau, Longuski, and Penzo present a method of parking orbit reorientation (the apo-twist maneuver) to establish the orbit geometry required for a tangential departure burn. Their method is discussed here, as it pertains to the Project Legend missions.

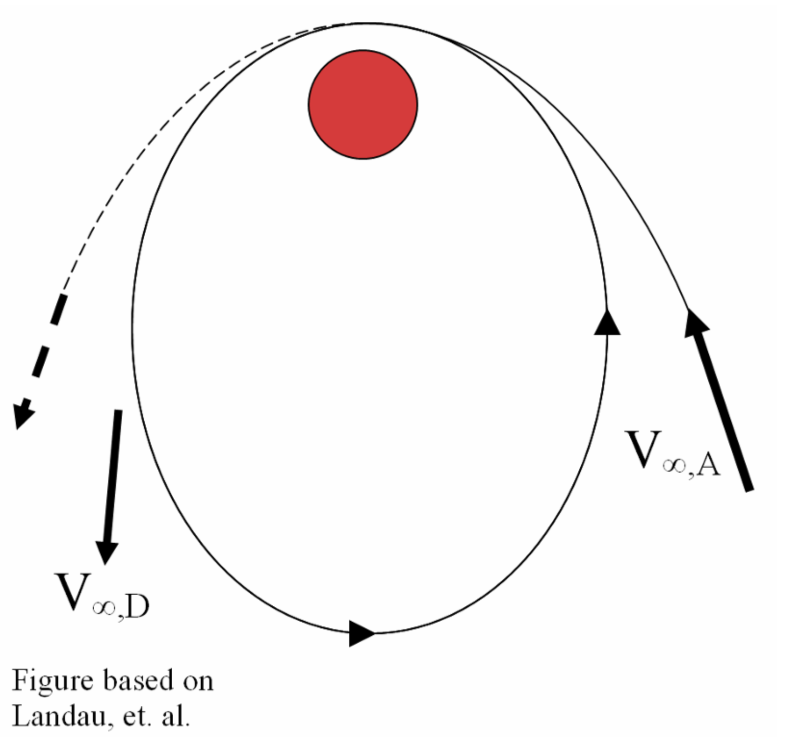


Figure 2-3: Misalignment of departure asymptote and parking orbit.

Figure 2-3 illustrates the need for Mars Parking Orbit (MPO) reorientation—as the departure asymptote $\left(V_{\infty,D}\right)$ depicted is not accessible from the given parking orbit. The apo-twist is a plane-change maneuver conducted at apoapsis which rotates the MPO about its major axis (the line of apsides). The altered geometry achieved with the apo-twist is shown in Figure 2-4. A vector diagram for the maneuver is given in Figure 2-5.

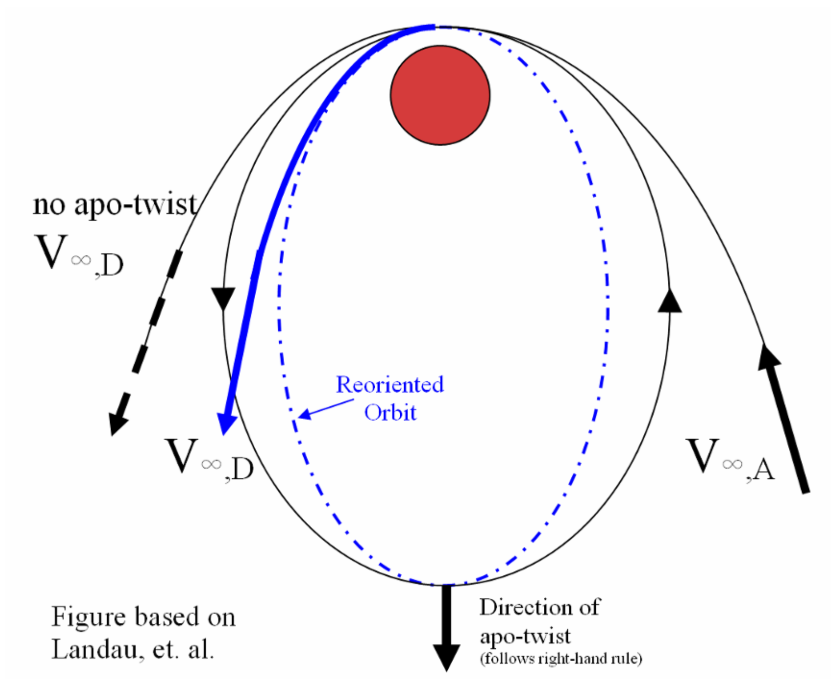


Figure 2-4: Result of apo-twist maneuver.

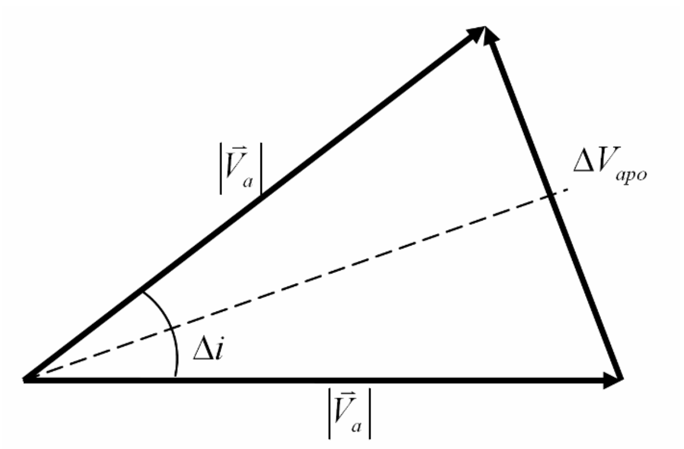


Figure 2-5: Vector diagram for apo-twist maneuver.

This maneuver (which we assume to be impulsive) requires a velocity increment given by

$$\Delta V_{apo} = 2V_a \sin\left(\frac{\Delta i}{2}\right) \quad .$$

(34-1).

The apo-twist maneuver requires the periapsis vectors for the orbits (before and after the burn) to be collinear. In general, we can select either the inclination or periapsis location of the capture orbit. We choose the periapsis location in order to enable apo-twist maneuvers. For Project Legend, the apo-twist maneuvers provide for inclination changes before the MLV landing mission and the CTV Mars departure.

34.3.3.1.3 Analysis

The availability and cost of the apo-twist maneuver varies with each mission opportunity and depends heavily on orbit perturbations and arrival conditions. A full characterization of Project Legend's apo-twist maneuvers is beyond the scope of this feasibility study. Therefore, we assume the availability of the apo-twist to achieve an equatorial orbit for the landing mission and orient the CTV orbit for a tangential departure burn. Further, we budget a 1.0 km/s velocity capability (corresponding to two 0.5 km/s, 65 degree apo-twist maneuvers) for each mission investigated. In considering five missions with similar parking orbits to the CTV but without lander considerations, Landau et. al. identify the largest (single) apo-twist maneuver as 0.245 km/s [1,2]. Accordingly, we consider this maneuver budget (total of 1.0 km/s for the two burns) to be a conservative estimate, ensuring that we avoid underestimating the mass launched to Earth orbit.

References

[1] Landau, Damon F., James M Longuski, and Paul A. Penzo. "A Method for Parking Orbit Reorientation for Human Missions to Mars." submitted to *Journal of Spacecraft and Rockets* for publication.

[2] Landau, Damon F., James M Longuski, and Paul A. Penzo. "Parking Orbits for Human Missions to Mars." American Astronautical Society, AAS 03-514.

34.3.3.2 Mars Orbit Configuration

Author: George Pollock

34.3.3.2.1 Orbit Description

During its stay at Mars, the Crew Transport Vehicle (CTV) orbits the planet with a one-Martian-day period and a periapsis (closest approach) altitude of 350 kilometers. We select this Mars Parking Orbit (MPO) from a trade study to assess the propellant mass cost of different orbital periods.

34.3.3.2.2 Theory and Assumptions

Our trade study investigates the impact of the MPO period on propellant mass cost. During its mission, the CTV performs six major propulsive maneuvers. The first and last of the CTV's burns (Earth departure and capture) are unaffected by the MPO period. Four maneuvers take place about Mars which depend heavily on the MPO period (capture, apo-twist #1, apo-twist #2, and departure). Two maneuvers performed by the MLV also depend on MPO period. Propellant requirements for both the CTV and MLV hinge on MPO period in a competing fashion—as period decreases, the MLV mass decreases but the CTV mass increases (and vice versa). We investigate the opposing mass effects of the MPO period on the two vehicles, including the MLV as payload on the CTV for the mission's outbound leg.

34.3.3.2.2.1 Crew Transport Vehicle Maneuvers at Mars

34.3.3.2.2.1.1 Mars Capture Maneuver

The CTV approaches Mars on a hyperbolic trajectory with a hyperbolic excess velocity of $\vec{V}^{-}_{\infty/♂}$ (Figure 2-6). We perform a single maneuver at periapsis (denoted by $\vec{r}_p$) to capture into the desired parking orbit at Mars. We first determine the velocity of the CTV at periapsis of the arrival hyperbola. From the energy equation,

$$E = \frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2a}$$

(34-2)

and the parameters for Mars: $\mu_{♂} = 4.2828\frac{\text{km}^3}{\text{s}^2}$, $R_{♂} = 3397\text{km}$, we arrive at the following expression for the arrival velocity at periapsis

$$V_{A,p} = \sqrt{\left(V^{-}_{\infty/♂}\right)^2 + \frac{2\mu_{♂}}{r_p}} = \sqrt{\left(V^{-}_{\infty/♂}\right)^2 + 22.86\frac{\text{km}^2}{\text{s}^2}}$$

(34-3).

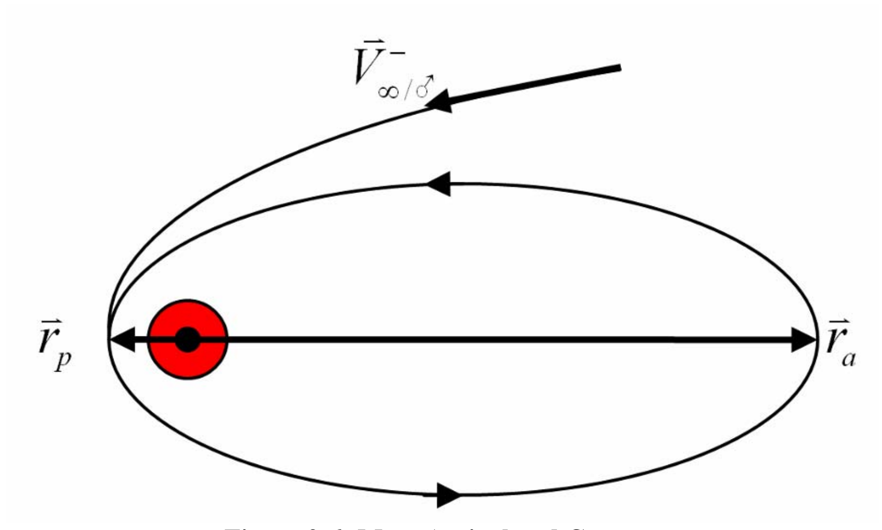


Figure 2-6: Mars Arrival and Capture

Equation (34-3) depends heavily on the hyperbolic excess velocity relative to Mars ($\vec{V}^{-}_{\infty/♂}$) and thus varies with each mission opportunity.

Additionally, the periapsis velocity desired after the capture maneuver depends on the period and periapsis altitude of the parking orbit. We choose a periapsis altitude of 350 kilometers to place the CTV well above the Martian atmosphere (which effectively vanishes above 300 km altitude) to eliminate orbit decay. With the periapsis radius now constrained, we have orbit period as the only remaining free variable affecting propellant cost. We now find the periapsis velocity on the elliptical parking orbit as a function of orbit period. Rearranging the equation for orbital period gives the semi-major axis in terms of period (P) in seconds,

$$a = \sqrt[3]{\left(\frac{\text{P}}{2\pi}\right)^2 \mu_{♂}}$$

(34-4).

The expression for periapsis velocity is,

$$v_p = \sqrt{\frac{2\mu_{♂}}{r_p} - \frac{\mu_{♂}}{a}}$$

(34-5).

Combining these two equations yields the periapsis velocity as a function of orbit period,

$$v_p = \sqrt{\frac{2\mu_{♂}}{r_p} - \left(\frac{2\pi\mu_{♂}}{\mathrm{P}}\right)^{2/3}}$$

(34-6).

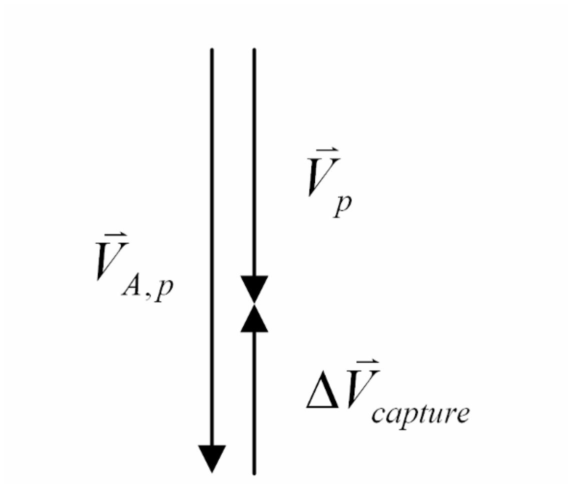


Figure 2-7: Vector diagram for Mars capture maneuver

It follows that the capture velocity increment (ΔV) is given by

$$\Delta V_{capture} = \left| V_p - V_{A,p} \right|$$

(34-7)

or in expanded form

$$\Delta V_{capture} = \left| \sqrt{\frac{2\mu_{♂}}{r_p} - \left(\frac{2\pi\mu_{♂}}{\mathrm{P}}\right)^{2/3}} - \sqrt{\left(V^{-}_{\infty/♂}\right)^2 + 22.86\frac{\mathrm{km}^2}{\mathrm{s}^2}} \right|$$

(34-8)

The capture velocity increment for a given mission solely depends upon the period of the parking orbit. Propellant cost for the capture maneuver increases with decreasing orbit period. This propellant cost will be the focus of our trade study to determine the Mars parking orbit period.

34.3.3.2.2.1.2 Mars Departure Maneuver

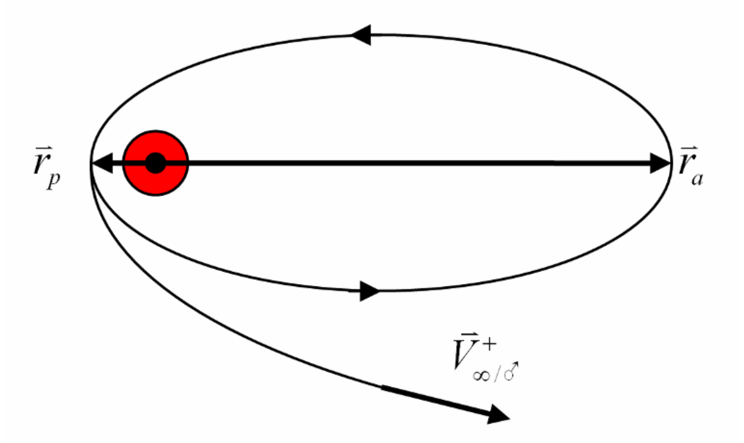


Figure 2-8: Departure from Mars Parking Orbit

Similar to the development for Mars capture, the velocity increment required for the departure burn (Figure 2-8) can be determined as a function of period

$$\Delta V_{capture} = \left| \sqrt{\left(V^{-}_{\infty/♂}\right)^2 + 22.86\frac{\text{km}^2}{\text{s}^2}} - \sqrt{\frac{2\mu_♂}{r_p} - \left(\frac{2\pi\mu_♂}{\text{P}}\right)^{2/3}} \right|$$

(34-9).

Note that this treatment assumes the desired hyperbolic departure asymptote is accessible from the CTV's orbit about Mars via a tangential maneuver. Two apo-twist maneuvers set up the required geometry for the MLV mission and CTV departure from Mars. (The apo-twist is a special case of plane change maneuver and is discussed in the section titled "Apo-Twist Maneuver.") For the trade study described in this section, an apo-twist ΔV budget for orbital repositioning is allotted for two plane changes of 65 degrees magnitude.

34.3.3.2.2.2 Mars Lander Vehicle Maneuvers

As a substantial payload during the CTV's first three major burns (Earth departure, Mars capture, and first apo-twist), the MLV must also be considered in this analysis. In general, a higher-period CTV orbit increases the MLV's propellant mass (thus increasing the CTV's payload mass over the first three burns).

34.3.3.2.2.2.1 Deorbit Maneuver

The deorbit burn is a relatively small maneuver ($\Delta V \approx 20$ m/s) performed at apoapsis of the CTV's parking orbit. We use an iterative process to determine the trajectory to atmospheric interface and compute the magnitude of the deorbit velocity increment (refer to appendix for code).

34.3.3.2.2.2.2 Powered Descent

Propulsive ΔV requirements for the powered portion of landing are 840 m/s (from J. Stalbaum's trajectory propagation). This requirement is assumed to be unaffected by the differing MLV entry velocities produced by the elliptical orbits considered here.

34.3.3.2.2.2.3 Powered Ascent

The ascent takes the MLV from the Martian surface to circular velocity at an altitude of 120 kilometers (ΔV_1 = 4126 m/s) then employs a Hohmann transfer to a 350 kilometer altitude circular orbit (ΔV_2 = 55 m/s, ΔV_3 = 54 m/s). Thus, after a velocity increment of 4235 m/s, the MLV is in a circular orbit which intersects the CTV's parking orbit at MPO periapsis (Figure 2-9). Varying the MPO period does not alter this standard ascent.

34.3.3.2.2.2.4 Rendezvous

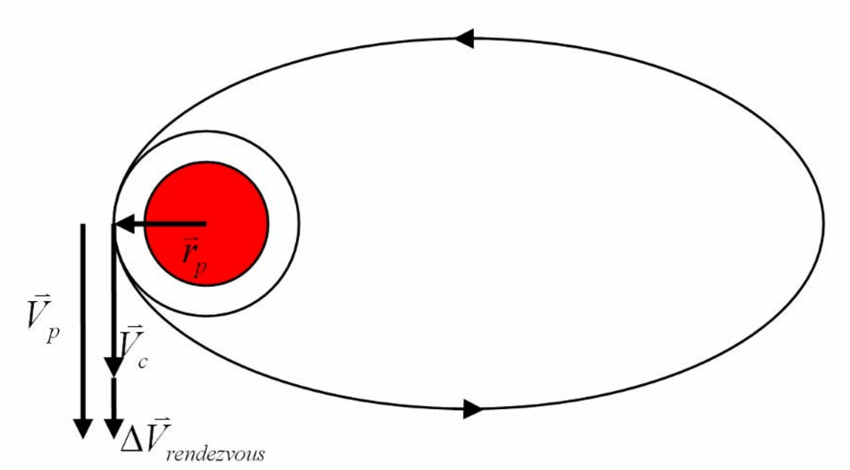


Figure 2-9: MLV rendezvous maneuver

The final step of the MLV's return to the CTV is the rendezvous maneuver (depicted in Figure 2-9). This burn's cost depends upon the velocity of the CTV at periapsis, and is given by

$$\Delta V_{rendezvous} = \sqrt{\frac{2\mu_{♂}}{r_p} - \frac{\mu_{♂}}{a}} - V_C$$

(34-10)

34.3.3.2.2.3 Propellant Cost

Propellant cost for each of the MLV's maneuvers is determined with the ideal rocket equation

$$m_0 = m_f e^{\left(\frac{\Delta V}{g_0 I_{sp}}\right)}$$

(34-11).

This equation permits the estimate of the total MLV mass based upon a specified crew module mass at the conclusion of its mission (and accounting for the aeroshell thermal protection system, parachutes, landing struts, and additional structure jettisoned during the mission). Once the MLV gross mass is obtained, the ideal rocket equation is used to compute the CTV's propellant mass (working backward through all six major burns of the mission).

34.3.3.2.2.4 Assumptions

"Worst-case" (largest ΔV) mission opportunity for CTV:

- Earth departure on 25 September 2028, ΔV = 4.59 km/s
- Mars arrival on 16 August 2029
- Mars departure on 1 August 2030
- Earth arrival on 7 August 2031, ΔV = 1.11 km/s
- MLV crew module mass = 3,953 kg
- MLV descent: Isp = 440 sec; propellant mass fraction, λ = 0.80
- MLV ascent Isp = 465.5 sec; propellant mass fraction, λ = 0.85
- CTV mass following Earth capture = 134,115 kg
- CTV Isp = 1,000 sec
- Budget for 2 x 65 degree apo-twist maneuvers
- Impulsive maneuvers
- Effect of MLV mission duration on MLV crew module mass neglected
- ARV mass = 5,000 kg
- MLV Thermal Protection System mass = 15.13% of MLV gross mass
- MLV parachute mass = 3.1% of MLV gross mass
- MLV descent stage structural mass (landing struts, cryogenic fuel storage system, etc) = 518 kg

34.3.3.2.3 Analysis

34.3.3.2.3.1 Trade Study

Results from the trade study of MLV and CTV masses based on Mars Parking Orbit period are presented in Figure 2-10 and Figure 2-11. The effects are particularly dramatic for periods less than one Martian day.

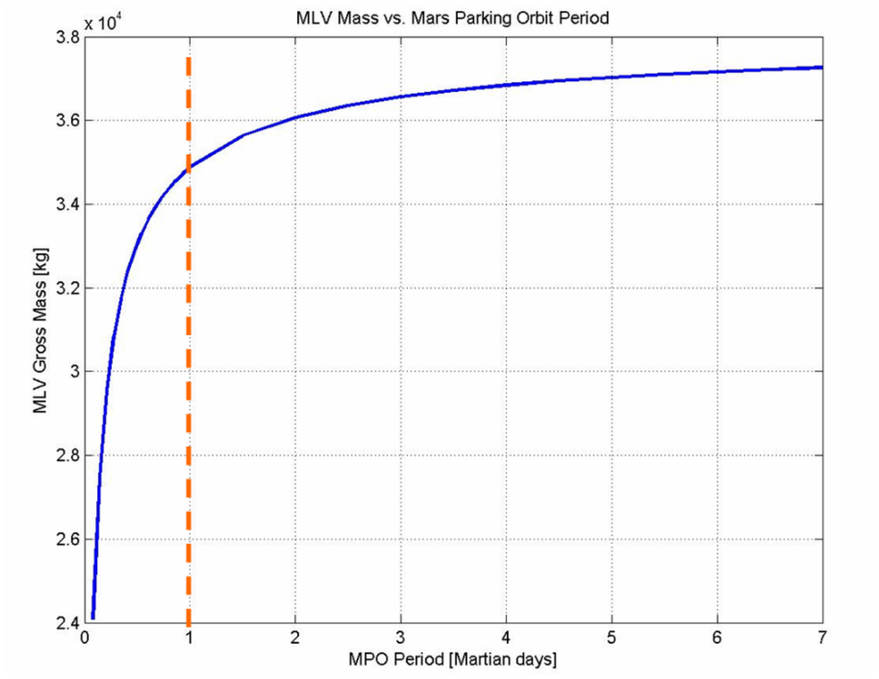


Figure 2-10: MLV gross mass vs. MPO period

For instance, the MLV mass drops more than 10,000 kilograms in moving from one Martian day to a circular orbit (Figure 2-10). Conversely, the CTV gross mass increases by over 450,000 kilograms for the same change in period (Figure 2-11). Figure 2-12 provides the CTV's ΔV and propellant mass as a function of MPO period.

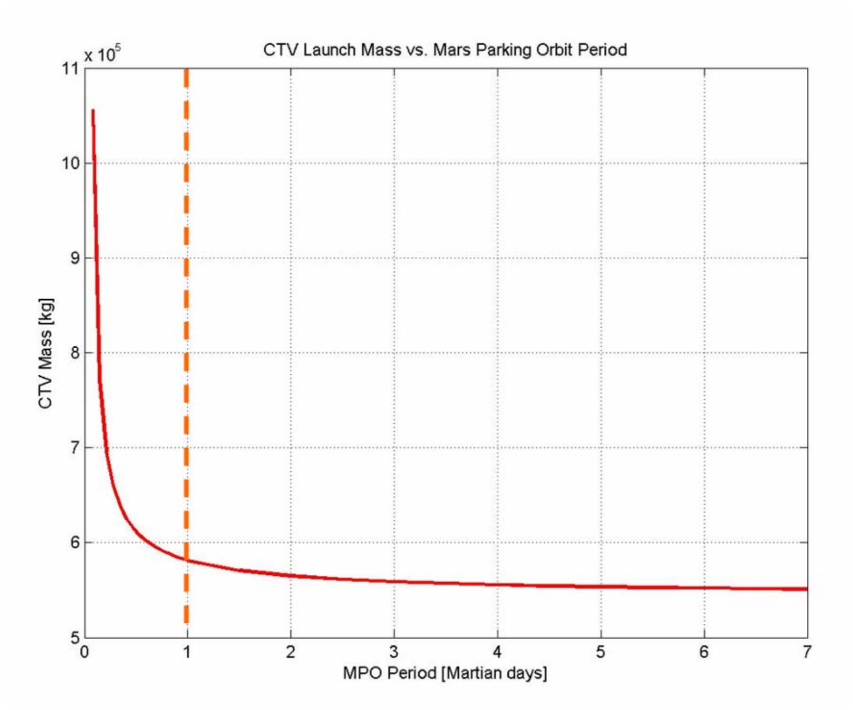


Figure 2-11: CTV gross mass vs. MPO period

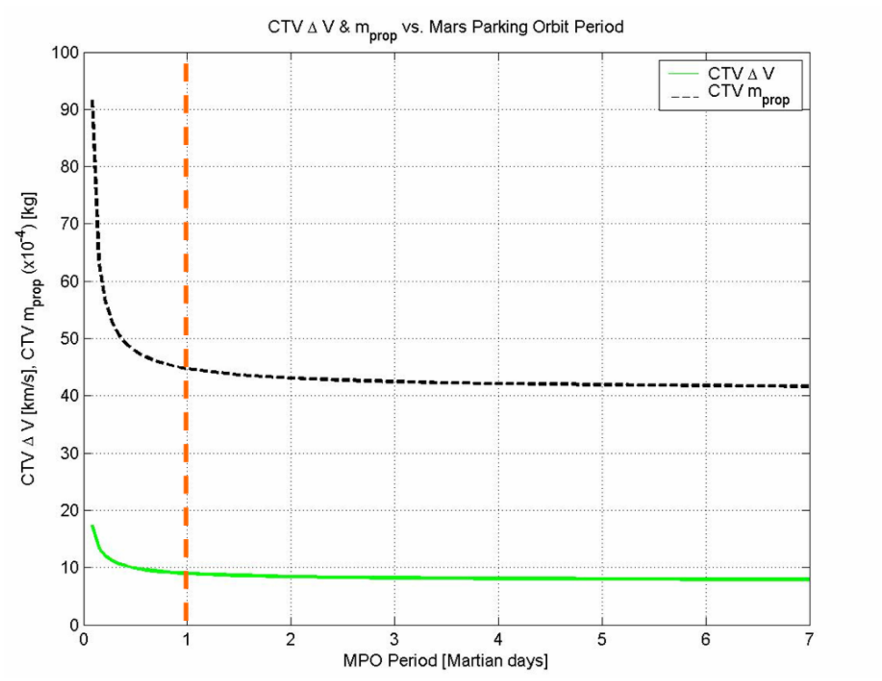


Figure 2-12: CTV Δ*V* and propellant mass vs. MPO period

We select the MPO period based not only on cost but also on operational considerations. From a safety and abort scenario perspective, we seek to maximize the frequency of rendezvous opportunities for the MLV and CTV. Since the rendezvous occurs at periapsis of the CTV's orbit, the frequency of rendezvous opportunities is the inverse of the orbit period. Additionally, the MLV crew compartment mass is minimized when the operational lifetime is a minimum (this occurs for a short surface stay and for shorter MPO periods). Clearly, the optimal solution (of those considered here) is the 7 Martian-day period, as it is the least expensive. Table 34-1 compares the cost of the near optimal 1 Martian-day MPO with that of the 7 Martian-day orbit. We select the 1 Martian-day orbit as a compromise solution which addresses safety, abort scenario, and MLV operational lifetime with a relatively small penalty in mass cost.

Table 34-1

	7 Martian-Day	1 Martian-Day	Difference
CTV ΔV [km/s]	7.93	8.99	1.06
CTV m_{prop} [kg]	416,800	447,100	30,300
CTV Earth departure mass [kg]	550,900	581,200	30,300
MLV ΔV [km/s]	5.58	5.43	-0.15
MLV m_{prop} [kg]	26,500	24,600	-1900
MLV gross mass [kg]	37,300	34,900	-2400

34.3.3.2.3.2 Mars Parking Orbit Elements

The CTV's Mars Parking Orbit will have characteristics listed in Table 34-2.

Table 34-2

Description	Symbol	Value
Periapsis radius [km]	r_p	3750
Apoapsis radius [km]	r_a	37,100
Semi-major axis [km]	a	20,400
Eccentricity	e	0.817
Period [hours]	P	24.62

Bibliography

Howell, K. C., "AAE 532: Orbit Mechanics Course Notes." Purdue University, Fall Semester 2004.

35 Shaw, Zade

35.1.1.1 ARV Link Budget Code

Author: Zade Shaw

Contributors: Brendan Eash, Jeri Metzger

35.1.1.1.1 Code Description

We employ this code to evaluate the communications link budget for the Ascent and Recovery Vehicle (ARV). The code has inputs of data rate, transmission frequency, dish diameters, noise temperature, antenna efficiencies, and link distance. The outputs take the form of plots that display the Signal-to-Noise (SNR) ratio in dB to the transmitting power required.

35.1.1.1.2 Code

```
% Brendan Eash
% Zade Shaw
% AAE 450
% Link Budget Analysis

% Code developed by Brendan Eash on Sunday January 23rd, 2005
% Code modified by Zade Shaw and Brendan Eash on Monday January 24th, 2005
% Code modified by Zade Shaw on Saturday February 5th, 2005

clear all;
close all;
clc;
format long;
format compact;
why(29);

% Data Rate [bps]
Rd = 10*log10(100e6);

% Transmission frequency [GHz]
f = 31.8; % Ka band, GHz

% Earth-to-Mars distance [km]
am = 2.27936636e8; %semi-major axis Mars, km
ae = 1.49597807e8; %semi-major axis Earth, km
S = am + ae; %Earth-Mars Distance, km

% Transmitting Power [W]
POWER = linspace(1e3,90e3,100); % Power, W (100e3)

% Antenna efficiencies
etaR = .65; % receiving antenna efficency
etaT = .70; % transmitting antenna efficency

% Antenna diameters [m]
Dr = 20; % diameter of recieving antenna, m
Dt = 6; % diameter of transmitting antenna, m (9.5 max without array)

% Noise temperature [K]
T = (400+150)/2; % noise equivalent temperature of system (K)

% Space loss [dB]
L_space = 92.45 + 20*log10(f) + 20*log10(S);

% Loss due to inefficiencies [dB]
L_other = 3;

% Atmospheric loss [dB]
L_atm = 6;
```

```
% Boltzmann's constant [dB]
k = -228.6;

% Gain of receiving antenna
GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);

% Noise temperature in dB [dB]
T_dB = 10*log10(T);

% Figure of merit
G_T = GR - T_dB;

% Run for various cases
for i = 1:length(Dt);

    for j = 1:100;

        % Gain of transmission antenna
        GT = 20.4 + 20*log10(Dt(i)) + 20*log10(f) + 10*log10(etaT);

        % Effective Isontropic Radiated Power
        EIRP = 10*log10(POWER(j)) + GT;

        % Signal to noise ratio available
        SNR_avail(i,j) = EIRP + G_T - L_space - L_other - L_atm - k - Rd;

    end
end

% Create Plot

% Convert power values from Watts to kilo-Watts
for i = 1:length(Dt)
    POWER_p(i,:) = POWER/1e3;
end

% Plot
figure(1);
grid on;
hold on;
plot(POWER_p',SNR_avail');
x = linspace(min(POWER)/1e3,max(POWER)/1e3,length(POWER));
ideal_s_n = linspace(3,3,length(POWER));
plot(x,ideal_s_n,'g*');
xlabel('POWER [kW]');
ylabel('S/N RATIO [dB]');
legend('6m Dia. Trans.','Ideal S/N',4);
t = sprintf('POWER  AND  DIAMETER  FOR  CTV  FOR  %iM  RECEIVING  DISH -- Zade Shaw',Dr);
title(t);
```

35.1.1.2 CTV Emergency Power Link Budget Code

Author: Zade Shaw

Contributors: Brendan Eash, Jeri Metzger

35.1.1.2.1 Code Description

We employ this code to evaluate the communications link budget for the Crew Transport Vehicle (CTV) emergency power. The code has inputs of data rate, transmission frequency, dish diameters, noise temperature, antenna efficiencies, and link distance. The outputs take the form of plots that display the Signal-to-Noise (SNR) ratio in dB to the transmitting power required.

35.1.1.2.2 Code

```
% Brendan Eash
% Zade Shaw
% AAE 450
% Link Budget Analysis

% Code developed by Brendan Eash on Sunday January 23rd, 2005
% Code modified by Zade Shaw and Brendan Eash on Monday January 24th, 2005
% Code modified by Zade Shaw on Saturday February 5th, 2005
% Code modified by Zade Shaw on Saturday February 26th, 2005 for CTV
% emergency status

clear all;
close all;
clc;
format long;
format compact;

% Data Rate [bps]
Rd = 10*log10(44e6);  % 4 regular TV signals (2 each way) at 6Mbps each and 10 Mbps for telemetry each
way

% Transmission frequency [GHz]
f = 31.8; % Ka band, GHz

% Earth-to-Mars distance [km]
S = 3.985538969e8; % Earth-Mars Distance, km

% Transmitting Power [W]
POWER = linspace(1e3,90e3,100); % Power, W (100e3)

% Antenna efficiencies
etaR = .60; % receiving antenna efficency
etaT = .55; % transmitting antenna efficency

% Antenna diameters [m]
Dr = 20; % diameter of recieving antenna, m
Dt = 5:2:12; % diameter of transmitting antenna, m (9.5 max without array)

% Noise temperature [K]
T = 400; % noise equivalent temperature of system (K)

% Space loss [dB]
L_space = 92.45 + 20*log10(f) + 20*log10(S);

% Loss due to inefficiencies [dB]
L_other = 3;

% Atmospheric loss [dB]
L_atm = 6;
```

```
% Boltzmann's constant [dB]
k = -228.6;

% Gain of receiving antenna
GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);

% Noise temperature in dB [dB]
T_dB = 10*log10(T);

% Figure of merit
G_T = GR - T_dB;

% Run for various cases
for i = 1:length(Dt);

%     Dt(1) = 8; % diameter of transmitting antenna must be some a value greater than zero

    for j = 1:100;

        % Gain of transmission antenna
        GT = 20.4 + 20*log10(Dt(i)) + 20*log10(f) + 10*log10(etaT);

        % Effective Isontropic Radiated Power
        EIRP = 10*log10(POWER(j)) + GT;

        % Signal to noise ratio available
        SNR_avail(i,j) = EIRP + G_T - L_space - L_other - L_atm - k - Rd;

    end
end

% Create Plot

% Convert power values from Watts to kilo-Watts
for i = 1:length(Dt)
    POWER_p(i,:) = POWER/1e3;
end

% Plot
figure(1);
grid on;
hold on;
plot(POWER_p',SNR_avail');
x = linspace(min(POWER)/1e3,max(POWER)/1e3,length(POWER));
ideal_s_n = linspace(3,3,length(POWER));
plot(x,ideal_s_n,'g*');
xlabel('POWER [kW]');
ylabel('S/N RATIO [dB]');
legend('5m Dia. Trans.','7m Dia. Trans.','9m Dia. Trans.','12m Dia. Trans.','Ideal S/N',4);
t = sprintf('POWER  AND  DIAMETER  FOR  CTV Emergency Status FOR  %iM  RECEIVING  DISH -- Zade
Shaw',Dr);
title(t);
```

35.1.1.3 Stability Analysis Code

Author: Zade Shaw

Contributors: Brendan Eash, Jackie Jaron

35.1.1.3.1 Code Inputs and Outputs

This code has inputs of moments of inertia, CTV mass, initial conditions, and orbit properties. We produce outputs that are in the form of plots, showing the nutation angle verses time or number of revolutions.

35.1.1.3.2 Code

```
% Zade Shaw
% AAE 450
% Rigid Body Dynamics -- CTV Construction

%--------------- Part D2-----------------------------------------

clear all;
close all;
clc;
why(29);

global s K1 K2 K3 mub e a E p

% Given information and Initial Conditions
%-----------------Region 6---------------------------------------

mub = 1.323e11;   % km^3/sec^2
a = 5e8;          % km
e = 0.25;
p = a*(1-e^2);

M = 270000;

Ia = 5.4894*10^8;
Ib = 5.0048*10^8;
Ic = 0.6697*10^8;

I1 = Ic;
I2 = Ia;
I3 = Ib;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;

x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.001;
x6_0 = 0.001;
x7_0 = 0.6;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];
```

```
% Integrate over time interval
dv = [0 24*60*60];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('dimensionalized_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(1);
plot(v,nutation*180/pi);
hold on;
title('Zade Shaw -- Nutation (\phi) for 10 revs w1_0 = w2_0 = 0.1, w3_0 = 1.0 -- Region 6');
xlabel('revolution, \nu [unitless]');
ylabel('nutation, \phi [degrees]');
grid on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Repeat the process without the gravity torque

% Calculate phi (nutation)
[v,x] = ode45('dimensionalized_tf_eom',dv,in_conds);
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot phi (nutation)
figure(1);
plot(v,nutation*180/pi,'r:');
legend('W/ Torque','W/O Torque',0);
hold off;


%---------------------Region 2---------------------------------------------

clear all;
clc;
why(29);

global s K1 K2 K3 mub e a E p

% Given information and Initial Conditions

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

M = 270000;

Ia = 5.4894*10^8;
Ib = 5.0048*10^8;
Ic = 0.6697*10^8;

I1 = Ib;
I2 = Ic;
I3 = Ia;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;

x1_0 = 0;
```

```
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.001;
x6_0 = 0.001;
x7_0 = 0.6;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate over time interval
dv = [0 24*60*60];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('dimensionalized_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(2);
plot(v,nutation*180/pi);
hold on;
title('Zade Shaw -- Nutation (\phi) for 10 revs w1_0 = w2_0 = 0.1, w3_0 = 1.0 -- Region 2');
xlabel('revolution, \nu [unitless]');
ylabel('nutation, \phi [degrees]');
grid on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Repeat the process without the gravity torque

% Calculate phi (nutation)
[v,x] = ode45('dimensionalized_tf_eom',dv,in_conds);
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot phi (nutation)
figure(2);
plot(v,nutation*180/pi,'r:');
legend('W/ Torque','W/O Torque',0);
hold off;


%%%%%------------------------PART D3--------------------------------------
%%%%%------------------Region 6-------------------------------------------

clear all;
clc;
why(29);

global s K1 K2 K3 mub e a E p

% Given information and Initial Conditions

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

M = 270000;

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ic;
```

```
I2 = Ia;
I3 = Ib;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;

x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.1;
x6_0 = 0.1;
x7_0 = 1.0;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for # revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('dimensionalized_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(3);
plot(v,nutation*180/pi);
hold on;
title('Zade Shaw -- Nutation (\phi) for 10 revs -- Region 6 and 2');
xlabel('revolution, \nu [unitless]');
ylabel('nutation, \phi [degrees]');
grid on;

%--------------REPEAT FOR OTHER I.C.'S----------------------------------

clear all;
clc;
why(29);

global s K1 K2 K3 mub e a E p

% Given information and Initial Conditions
% Still in Region 6

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

M = 270000;

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ic;
I2 = Ia;
I3 = Ib;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
```

```
K3 = (I1 - I2)/I3;

s = 0;

x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.05;
x6_0 = 0.05;
x7_0 = 1.0;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for # revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('dimensionalized_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(3);
plot(v,nutation*180/pi,'r');
hold off;
legend('w1_0=w2_0=0.1, w3_0=1.0','w1_0 = w2_0 = 0.05, w3_0 = 1.0',0);


%%%%------------------Do the same for Region 2------------------------------
%%%%---------------REGION 2----------------------------------------------------

clear all;
clc;
why(29);

global s K1 K2 K3 mub e a E p

% Given information and Initial Conditions
% Region 2

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

M = 270000;

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ib;
I2 = Ic;
I3 = Ia;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;

x1_0 = 0;
x2_0 = 0;
```

```
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.1;
x6_0 = 0.1;
x7_0 = 1.0;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for # revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('dimensionalized_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(4);
plot(v,nutation*180/pi);
hold on;
title('Zade Shaw -- Nutation (\phi) for 10 revs -- Region 2');
xlabel('revolution, \nu [unitless]');
ylabel('nutation, \phi [degrees]');
grid on;

%%%%%%----------------------------REPEAT FOR THE OTHER I.C.'S-------

clear all;
clc;
why(29);

global s K1 K2 K3 mub e a E p

% Given information and Initial Conditions
% Still in Region 2

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

M = 270000;

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ib;
I2 = Ic;
I3 = Ia;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;

x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.05;
x6_0 = 0.05;
x7_0 = 1.0;
```

```
% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for # revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('dimensionalized_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(4);
plot(v,nutation*180/pi,'r');
hold off;
% legend('w1_0=w2_0=0.1, w3_0=1.0','w1_0 = w2_0 = 0.05, w3_0 = 1.0',0);
legend('Region 6','Region 2',0);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Part E -- Gyrostat!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
clc;
why(29);

% Region 6

% Assume a density to produce mass of s/c and rotor
den = 10;
M_sc = 270000;
V_r = 2*pi*1^2;
M_r = V_r*den;

global s K1 K2 K3 I1 I2 SR J mub e a E p

% Given information and Initial Conditions

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ic;
I2 = Ia;
I3 = Ib;

J = 0.5*M_r*2^2;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;
SR = 90;

x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
```

```
x5_0 = 0.1;
x6_0 = 0.1;
x7_0 = 1.0;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for 2 revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('construction_gs_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(5);
plot(v,nutation*180/pi);
hold on;
title('Zade Shaw -- Nutation (\phi) for 10 revs w/ rotor and spin = 90 -- Region 6');
xlabel('revolution, \nu [unitless]');
ylabel('nutation, \phi [degrees]');
grid on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Region 6, Set2 I.C.'s

clear all;
clc;
why(29);

% Region 6

% Assume a density to produce mass of s/c and rotor
den = 10;
M_sc = 270000;
V_r = 2*pi*1^2;
M_r = V_r*den;

global s K1 K2 K3 I1 I2 SR J mub e a E p

% Given information and Initial Conditions

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ic;
I2 = Ia;
I3 = Ib;

J = 0.5*M_r*2^2;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;
SR = 90;
```

```
x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.05;
x6_0 = 0.05;
x7_0 = 1.0;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for 2 revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('construction_gs_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(5);
plot(v,nutation*180/pi,'r:');
hold off;
legend('w1_0=w2_0=0.1, w3_0=1.0','w1_0 = w2_0 = 0.05, w3_0 = 1.0',0);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Region 2, Set1 I.C.'s

clear all;
clc;
why(29);

% Region 2

% Assume a density to produce mass of s/c and rotor
den = 10;
M_sc = 270000;
V_r = 2*pi*1^2;
M_r = V_r*den;

global s K1 K2 K3 I1 I2 SR J mub e a E p

% Given information and Initial Conditions

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ib;
I2 = Ic;
I3 = Ia;

J = 0.5*M_r*2^2;

K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;
```

```
s = 0;
SR = 40;

x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.1;
x6_0 = 0.1;
x7_0 = 1.0;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for 2 revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('construction_gs_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(6);
plot(v,nutation*180/pi);
hold on;
title('Zade Shaw -- Nutation (\phi) for 10 revs w/ rotor and spin = 40 -- Region 2');
xlabel('revolution, \nu [unitless]');
ylabel('nutation, \phi [degrees]');
grid on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Region 2, Set2 I.C.'s

clear all;
clc;
why(29);

% Region 2

% Assume a density to produce mass of s/c and rotor
den = 10;
M_sc = 270000;
V_r = 2*pi*1^2;
M_r = V_r*den;

global s K1 K2 K3 I1 I2 SR J mub e a E p

% Given information and Initial Conditions

mub = 1.323e11;   % km^3/sec^2
a = 5e8;           % km % fix this!!!
e = 0.25;
p = a*(1-e^2);

Ia = 5.4894*10^7;
Ib = 5.0048*10^7;
Ic = 0.6697*10^7;

I1 = Ib;
I2 = Ic;
I3 = Ia;

J = 0.5*M_r*2^2;
```

```
K1 = (I2 - I3)/I1;
K2 = (I3 - I1)/I2;
K3 = (I1 - I2)/I3;

s = 0;
SR = 40;

x1_0 = 0;
x2_0 = 0;
x3_0 = 0;
x4_0 = 1;
x5_0 = 0.05;
x6_0 = 0.05;
x7_0 = 1.0;

% Initial conditions vector created
in_conds = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0 x7_0];

% Integrate for 2 revolutions
dv = [0 10];

% Change the tolerances
options = odeset('RelTol',1e-7,'AbsTol',[1e-7, 1e-7]);
warning off MATLAB:odearguments:RelTolIncrease

% Numerical integration using ode45
[v,x] = ode45('construction_gs_eom',dv,in_conds);

% Calculate the nutation angle history
C33 = 1 - 2*x(:,1).^2 - 2*x(:,2).^2;
nutation = acos(C33);

% Plot Nutation
figure(6);
plot(v,nutation*180/pi,'r:');
hold off;
legend('w1_0=w2_0=0.1, w3_0=1.0','w1_0 = w2_0 = 0.05, w3_0 = 1.0',0);
```

35.1.1.4 Stability Analysis EOM

Author: Zade Shaw

Contributors: Brendan Eash, Jackie Jaron

35.1.1.4.1 Code Description

This code is the Equation of Motion (EOM) function used in the stability analysis for the CTV.

35.1.1.4.2 Code

```
% Zade Shaw
% AAE 450
% Dimensionalized EOM

% With Gravity Torque

% Function to define the eom in terms of state variables defined below
function xdot = dimensionalized_eom(v,x)

% Global
global s K1 K2 K3 mub e a E p

M = v*2*pi;
E =  fzero(inline('E - e * sin(E) - M','E','e','M'),M,[],e,M);
r = a*(1-e*cos(E));
omega = sqrt(mub/r^3);

% Defining the state variables
xdot(1) = 0.5*(x(2)*(x(7) - omega) - x(3)*x(6) + x(4)*x(5));
xdot(2) = 0.5*(x(3)*x(5) + x(4)*x(6) - x(1)*(x(7) - omega));
xdot(3) = 0.5*(x(4)*(x(7) - omega) + x(1)*x(6) - x(2)*x(5));
xdot(4) = 0.5*(-x(1)*x(5) - x(2)*x(6) - x(3)*(x(7) + omega));
xdot(5) = (K1*(x(6)*x(7) - 12*omega^2*(x(1)*x(2) - x(3)*x(4))*(x(3)*x(1) + x(2)*x(4))));
xdot(6) = (K2*(x(5)*x(7) -  6*omega^2*(x(3)*x(1) + x(2)*x(4))*(1 - 2*x(2)^2 - 2*x(3)^2)));
xdot(7) = (K3*(x(5)*x(6) -  6*omega^2*(x(1)*x(2) - x(3)*x(4))*(1 - 2*x(2)^2 - 2*x(3)^2)));

% Organizing the xdot vector into a column of values
xdot = xdot';

% Return to the function
return
```

35.1.1.5 Stability Analysis EOM (Dimensionalized Torque Free Case)

Author: Zade Shaw

Contributors: Brendan Eash, Jackie Jaron

35.1.1.5.1 Code Description

This code is the Equation of Motion (EOM) function used in the stability analysis for the CTV for the dimensionalized torque free case.

35.1.1.5.2 Code

```
% Zade Shaw
% AAE 450
% Dimensionalized Torque Free EOM

% Without Gravity Torque
% Function to define the eom in terms of state variables defined below
function xdot = dimensionalized_tf_eom(v,x)

% Given values for zeta and omega_n
global s K1 K2 K3 mub e a E p

M = v*2*pi;
E =  fzero(inline('E - e * sin(E) - M','E','e','M'),M,[],e,M);
r = a*(1-e*cos(E));
omega = sqrt(mub/r^3);

% Defining the state variables
xdot(1) = 0.5*(x(2)*(x(7) - omega) - x(3)*x(6) + x(4)*x(5));
xdot(2) = 0.5*(x(3)*x(5) + x(4)*x(6) - x(1)*(x(7) - omega));
xdot(3) = 0.5*(x(4)*(x(7) - omega) + x(1)*x(6) - x(2)*x(5));
xdot(4) = 0.5*(-x(1)*x(5) - x(2)*x(6) - x(3)*(x(7) + omega));
xdot(5) = (K1*(x(6)*x(7)));
xdot(6) = (K2*(x(5)*x(7)));
xdot(7) = (K3*(x(5)*x(6)));

% Organizing the xdot vector into a column of values
xdot = xdot';

% Return to the function
return
```

35.1.1.6 Stability Analysis EOM (Torque Free Case)

Author: Zade Shaw

Contributors: Brendan Eash, Jackie Jaron

35.1.1.6.1 Code Description

This code is the Equation of Motion (EOM) function used in the stability analysis for the CTV for the torque free case.

35.1.1.6.2 Code

```
% Zade Shaw
% AAE 450

% Without Gravity Torque

% Function to define the eom in terms of state variables defined below
function xdot = construction_tf_eom(v,x)

% Given values for zeta and omega_n
global s K1 K2 K3

% Defining the state variables
xdot(1) = pi*(x(2)*(x(7) - s + 1) - x(3)*x(6) + x(4)*x(5));
xdot(2) = pi*(x(3)*x(5) + x(4)*x(6) - x(1)*(x(7) - s + 1));
xdot(3) = pi*(x(4)*(x(7) - s - 1) + x(1)*x(6) - x(2)*x(5));
xdot(4) = pi*(-x(1)*x(5) - x(2)*x(6) - x(3)*(x(7) - s - 1));
xdot(5) = 2*pi*(-s*x(6) + K1*(x(6)*x(7)));
xdot(6) = 2*pi*(s*x(5) + K2*(x(5)*x(7)));
xdot(7) = 2*pi*(s*x(5) + K3*(x(5)*x(6)));

% Organizing the xdot vector into a column of values
xdot = xdot';

% Return to the function
return
```

35.1.1.7 Stability Analysis EOM (Torque Free Case)

Author: Zade Shaw

Contributors: Brendan Eash, Jackie Jaron

35.1.1.7.1 Code Description

This code is the Equation of Motion (EOM) function used in the stability analysis for the CTV for the torque free case.

35.1.1.7.2 Code

```
% Zade Shaw
% AAE 450

% Without Gravity Torque

% Function to define the eom in terms of state variables defined below
function xdot = construction_tf_eom(v,x)

% Given values for zeta and omega_n
global s K1 K2 K3

% Defining the state variables
xdot(1) = pi*(x(2)*(x(7) - s + 1) - x(3)*x(6) + x(4)*x(5));
xdot(2) = pi*(x(3)*x(5) + x(4)*x(6) - x(1)*(x(7) - s + 1));
xdot(3) = pi*(x(4)*(x(7) - s - 1) + x(1)*x(6) - x(2)*x(5));
xdot(4) = pi*(-x(1)*x(5) - x(2)*x(6) - x(3)*(x(7) - s - 1));
xdot(5) = 2*pi*(-s*x(6) + K1*(x(6)*x(7)));
xdot(6) = 2*pi*(s*x(5) + K2*(x(5)*x(7)));
xdot(7) = 2*pi*(s*x(5) + K3*(x(5)*x(6)));

% Organizing the xdot vector into a column of values
xdot = xdot';

% Return to the function
return
```

35.1.1.8 ELV Link Budget Code

Author: Zade Shaw

Contributors: Brendan Eash, Jeri Metzger

35.1.1.8.1 Code Description

We employ this code to evaluate the communications link budget for the Earth Launch Vehicle (ELV). The code has inputs of data rate, transmission frequency, dish diameters, noise temperature, antenna efficiencies, and link distance. The outputs take the form of plots that display the Signal-to-Noise (SNR) ratio in dB to the transmitting power required.

35.1.1.8.2 Code

```
% Brendan Eash
% Zade Shaw
% AAE 450
% Link Budget Analysis
% ***** ELV *****
% Code developed by Brendan Eash on Sunday January 23rd, 2005
% Code modified by Zade Shaw and Brendan Eash on Monday January 24th, 2005
% Code modified by Zade Shaw on Saturday February 5th, 2005
% Code modified by Zade Shaw on Saturday February 7th, 2005

clear all;
close all;
clc;
format long;
format compact;

% Data Rate [bps]
Rd = 10*log10(10e6);

% Transmission frequency [GHz]
f = 2; % S-band, GHz (Orbiter Communications reference -- Shuttle uses this band)

% Earth-to-ELV distance [km]
S = 200; % Earth-ELV Distance, km

% Transmitting Power [W]
POWER = linspace(0.01e3,1e3,100); % Power, W (100e3)

% Antenna efficiencies
etaR = .50; % receiving antenna efficency
etaT = .50; % transmitting antenna efficency

% Antenna diameters [m]
Dr = 0.5; % diameter of recieving antenna, m
Dt = 1:1:3; % diameter of transmitting antenna, m (9.5)

% Noise temperature [K]
T = 400; % noise equivalent temperature of system (K)

% Space loss [dB]
L_space = 92.45 + 20*log10(f) + 20*log10(S);

% Loss due to inefficiencies [dB]
L_other = 6;

% Atmospheric loss [dB]
L_atm = 6;

% Boltzmann's constant [dB]
k = -228.6;

% Gain of receiving antenna
GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);

% Noise temperature in dB [dB]
T_dB = 10*log10(T);

% Figure of merit
G_T = GR - T_dB;

% Run for various cases
for i = 1:length(Dt);

    Dt(1) = 1; % diameter of transmitting antenna must be some a value greater than zero

    for j = 1:100;

        % Gain of transmission antenna
        GT = 20.4 + 20*log10(Dt(i)) + 20*log10(f) + 10*log10(etaT);

        % Effective Isontropic Radiated Power
        EIRP = 10*log10(POWER(j)) + GT;

        % Signal to noise ratio available
        SNR_avail(i,j) = EIRP + G_T - L_space - L_other - L_atm - k - Rd;
    end
end

% Create Plot

% Convert power values from Watts to kilo-Watts
for i = 1:length(Dt)
    POWER_p(i,:) = POWER/1e3;
end

% Plot
figure(1);
grid on;
hold on;
plot(POWER_p',SNR_avail');
x = linspace(min(POWER)/1e3,max(POWER)/1e3,length(POWER));
ideal_s_n = linspace(3,3,length(POWER));
plot(x,ideal_s_n,'g*');
xlabel('POWER [kW]');
ylabel('S/N RATIO [dB]');
legend('1m Dia. Trans.','2m Dia. Trans.','3m Dia. Trans.','Ideal S/N',0);
t = sprintf('POWER  AND  DIAMETER  FOR  ELV  FOR  0.5M  RECEIVING  DISH -- Zade Shaw',Dr);
title(t);
```

35.1.1.9 MHV in Transit Link Budget Code

Author: Zade Shaw

Contributors: Brendan Eash, Jeri Metzger

35.1.1.9.1 Code Description

We employ this code to evaluate the communications link budget for the Mars Habitat Vehicle (MHV) while in transit from Earth to Mars. The code has inputs of data rate, transmission frequency, dish diameters, noise temperature, antenna efficiencies, and link distance. The outputs take the form of plots that display the Signal-to-Noise (SNR) ratio in dB to the transmitting power required.

35.1.1.9.2 Code

```
% Brendan Eash
% Zade Shaw
% AAE 450
% Link Budget Analysis
% MHV Transit Analysis

% Code developed by Brendan Eash on Sunday January 23rd, 2005
% Code modified by Zade Shaw and Brendan Eash on Monday January 24th, 2005
% Code modified by Zade Shaw on Saturday February 5th, 2005
% Code modified by Zade Shaw on Saturday February 26th, 2005 for the CTV eme
% while in transit

clear all;
close all;
clc;
format long;
format compact;
why(29);

% Data Rate [bps]
Rd = 10*log10(10e6);

% Transmission frequency [GHz]
f = 31.8; % Ka band, GHz
```

```
% Earth-to-Mars distance [km]
am = 2.27936636e8; %semi-major axis Mars, km
ae = 1.49597807e8; %semi-major axis Earth, km
S = am + ae; %Earth-Mars Distance, km

% Transmitting Power [W]
POWER = linspace(1e3,90e3,100); % Power, W (100e3)

% Antenna efficiencies
etaR = .65; % receiving antenna efficency
etaT = .70; % transmitting antenna efficency

% Antenna diameters [m]
Dr = 20; % diameter of recieving antenna, m
Dt = 2; % diameter of transmitting antenna, m (9.5 max without array)

% Noise temperature [K]
T = (150+400)/2; % noise equivalent temperature of system (K)

% Space loss [dB]
L_space = 92.45 + 20*log10(f) + 20*log10(S);

% Loss due to inefficiencies [dB]
L_other = 3;

% Atmospheric loss [dB]
L_atm = 6;

% Boltzmann's constant [dB]
k = -228.6;

% Gain of receiving antenna
GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);

% Noise temperature in dB [dB]
T_dB = 10*log10(T);

% Figure of merit
G_T = GR - T_dB;

% Run for various cases
for i = 1:length(Dt);

    for j = 1:100;

        % Gain of transmission antenna
        GT = 20.4 + 20*log10(Dt(i)) + 20*log10(f) + 10*log10(etaT);

        % Effective Isontropic Radiated Power
        EIRP = 10*log10(POWER(j)) + GT;

        % Signal to noise ratio available
        SNR_avail(i,j) = EIRP + G_T - L_space - L_other - L_atm - k - Rd;

    end
end

% Create Plot

% Convert power values from Watts to kilo-Watts
for i = 1:length(Dt)
    POWER_p(i,:) = POWER/1e3;
end

% Plot
figure(1);
grid on;
hold on;
plot(POWER_p',SNR_avail');
x = linspace(min(POWER)/1e3,max(POWER)/1e3,length(POWER));
```

```
ideal_s_n = linspace(3,3,length(POWER));
plot(x,ideal_s_n,'g*');
xlabel('POWER [kW]');
ylabel('S/N RATIO [dB]');
legend('2m Dia. Trans.','Ideal S/N = 3 dB',4);
t = sprintf('POWER FOR MHV IN TRANSIT FOR %iM RECEIVING DISH -- Zade Shaw',Dr);
title(t);
```

35.1.1.10 MLV Communications System Trade Study

Author: Zade Shaw

Contributors: Jeri Metzger, Brendan Eash

35.1.1.10.1 Trade Study Description

The crucial trade study that led us to the design of the communications system of the Mars Lander Vehicle (MLV) is detailed below.

35.1.1.10.2 Theory and Assumptions

Our goal in designing the MLV's communications system is to minimize mass while maintaining the goal of 40 Mbps. A key assumption in our design process is that pointing a dish system during the descent phase of our mission is much more difficult to achieve than broadcasting with an omni-directional system.

35.1.1.10.3 Analysis

With a goal of minimizing the total mass of the MLV we look at communications systems implementing dish and omni-directional antennas. The dish system's hardware is about 105 kilograms as opposed to only 26 kilograms for omni-directional. However, when using omni-directional antennas more power is required to do the same job because we are broadcasting our signal in all directions. The additional power requires additional cooling systems mass with the end result being that the omni-directional system is actually about 8 kilograms heavier than that of the dish system. Nevertheless, we choose the omni-directional system because of the added complexities involved in having to point the dish system in order to transmit. Not having to point is particularly helpful during the MLV's descent to the Mars surface. This information is summarized in .

Table 3-3.

Table 3-3 MLV Communications System Trade Study

	Omnidirectional	Dish
Size (diameter/volume)	Diameter - 2.24 m Vol. - 0.003 m3	Dish dia. - 0.50 m Vol. - 0.003 m3
Power	0.60 kW	0.09 kW
Hardware Mass	26 kg	105 kg

Thermal Control System Mass	564.17 kg	477.57 kg
Comm. + Thermal Mass	590.17 kg	582.57 kg

References

[1] Larson, Wiley J., & Pranke, Linda K. *Human Spaceflight: Mission Analysis and Design,* (pgs. 869-906). New York, US: McGraw Hill.

[2] Larson, Wiley J., & Wertz, James R. (1999). *Space Mission Analysis and Design,* (pgs. 533-586). El Segundo, US: Microcosm Press.

[3] http://deepspace.jpl.nasa.gov/dsn/index.html

[4] http://eis.jpl.nasa.gov/deepspace/dsndocs/810-005/

[5] http://www.fiber-optics.info/articles/dtv-hdtv.htm

[6] http://www.triadtwcable.com/cableserv/images/SAHDTVFAQs.pdf

[7] http://www.tmeg.com/tutorials/antennas/antennas.htm

[8] http://www.rfsworld.com/index.php?p=582

[9] http://www.intel.com/design/network/products/optical/lc_transceivers.htm

[10] http://www.terrawaveonline.com/Dynamic/

35.1.1.11 MLV Link Budget Code

Author: Zade Shaw

Contributors: Brendan Eash, Jeri Metzger

35.1.1.11.1 Code Description

We employ this code to evaluate the communications link budget for the Mars Lander Vehicle (MLV). The code has inputs of data rate, transmission frequency, dish diameters, noise temperature, antenna efficiencies, and link distance. The outputs take the form of plots that display the Signal-to-Noise (SNR) ratio in dB to the transmitting power required.

35.1.1.11.2 Code

```
% Brendan Eash
% Zade Shaw
% AAE 450
% Link Budget Analysis

% Code developed by Brendan Eash on Sunday January 23rd, 2005
% Code modified by Zade Shaw and Brendan Eash on Monday January 24th, 2005
% Code modified by Zade Shaw on Saturday February 5th, 2005

clear all;
close all;
clc;
format long;
format compact;
why(29);

% Data Rate [bps]
Rd = 10*log10(40e6); % HDTV signal and telemetry

% Transmission frequency [GHz]
f = 2.5; % S band, GHz

% Mars surface-to-Mars orbit distance [km]
a = 20081; % circular orbit of MORS
S = a; % MLV to Mars orbit max Distance, km

% Transmitting Power [W]
POWER = linspace(0.01e3,2e3,100); % Power, W (100e3)

% Antenna efficiencies
etaR = .65; % receiving antenna efficency
etaT = .65; % transmitting antenna efficency

% Antenna diameters [m]
Dr = 1; % diameter of recieving antenna, m

% Noise temperature [K]
T = (150+400)/2; % noise equivalent temperature of system (K)

% Space loss [dB]
L_space = 92.45 + 20*log10(f) + 20*log10(S);

% Loss due to inefficiencies [dB]
L_other = 2;

% Atmospheric loss [dB]
L_atm = 3;

% Boltzmann's constant [dB]
```

```
k = -228.6;

% Gain of receiving antenna
GR = 20.4 + 10*log10(etaR) + 20*log10(Dr) + 20*log10(f);

% Noise temperature in dB [dB]
T_dB = 10*log10(T);

% Figure of merit
G_T = GR - T_dB;

% Run for various cases
for i = 1;

    for j = 1:100;

        % Gain of transmission antenna
        GT = 12; % omni-directional antennas have no gain but by using load coils can get an effective
gain (Thanks Prof. Filmer)

        % Effective Isontropic Radiated Power
        EIRP = 10*log10(POWER(j)) + GT;

        % Signal to noise ratio available
        SNR_avail(i,j) = EIRP + G_T - L_space - L_other - L_atm - k - Rd;

    end
end

% Create Plot

% Convert power values from Watts to kilo-Watts
for i = 1
    POWER_p(i,:) = POWER/1e3;
end

% Plot
figure(1);
grid on;
hold on;
plot(POWER_p',SNR_avail');
x = linspace(min(POWER)/1e3,max(POWER)/1e3,length(POWER));
ideal_s_n = linspace(3,3,length(POWER));
plot(x,ideal_s_n,'g*');
xlabel('POWER [kW]');
ylabel('S/N RATIO [dB]');
t = sprintf('OMNIDIRECTIONAL  MLV  POWER  FOR  %iM  CTV  RECEIVING  DISH -- Zade Shaw',Dr);
title(t);
```

35.1.1.12 Additional MORS Pictures

Authors: Zade Shaw and Justin McCurdy

35.1.1.12.1 Model Description

We show two additional MORS models below in Figure 3-13 and Figure 3-14.

35.1.1.12.2 Models

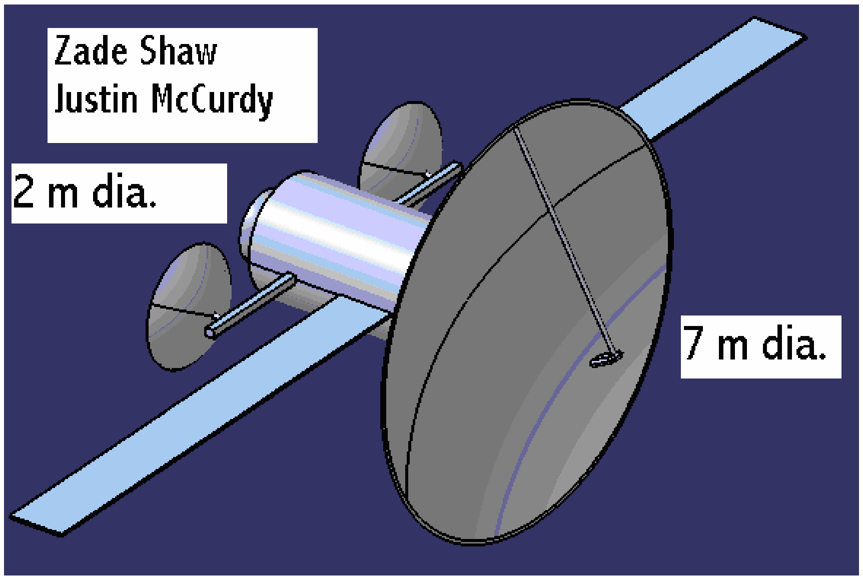


Figure 3-13 MORS Main Dish View

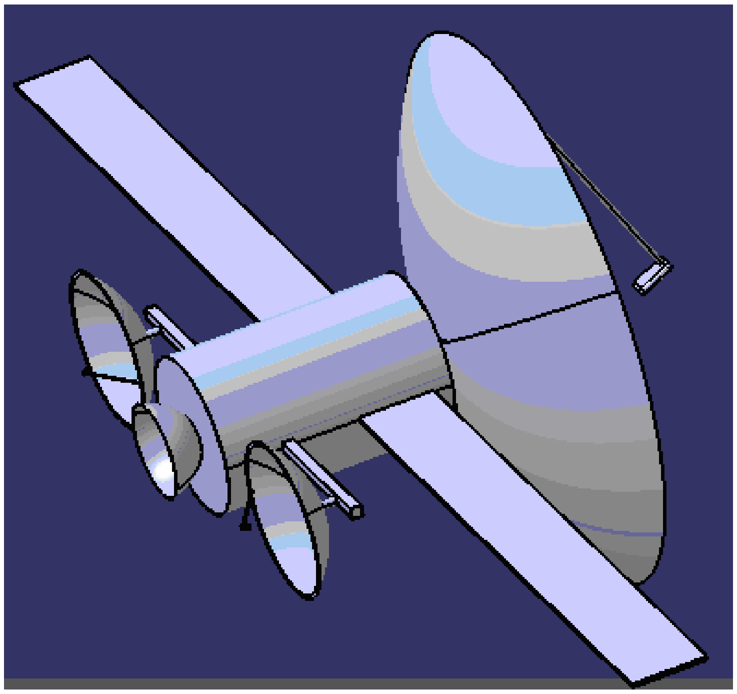


Figure 3-14 MORS 1-Meter Diameter Dish View

36 Sippel, Aaron D.

36.1 Appendix on Docking, CTV Sizing, and Crew Quarters

Author(s): Aaron Sippel

Contributors: Matt Harrigan

36.1.1 Docking Systems

For the preliminary study on docking, we choose to size our systems based on the International Space Station (ISS) and the Pirs docking compartment. The objective is to design a system that is similar on any of the vehicles that would be in space. The common docking system ensures compatibility and safety. The system on the ISS is the "Quest" Joint Airlock Module [1]. We considered looking at an airlock to do space walks early on in the mission design, but this solution was ruled out later due to its inefficiencies and an overall increase in the probability of a mission failure. It was simply unnecessary. The Pirs docking compartment [2] offers much more flexibility in the design, but is to complex for the simple functions we needed on the Crew Transport Vehicle (CTV). A table of dimension and design information can be seen in Table 36-1. The length parameter is the length of the cylinder. The docking structures are thought of as cylinders and the axis of symmetry is aligned horizontally.

Table 36-1 Design parameters of current docking systems.

Design Parameters	Quest Joint Airlock (ISS) [3]	Pirs Docking Compartment [2]
Length (m)	5.5	5
Diameter (m)	4	2.2
Mass (kg)	6,064	3,630
Volume (m^3)	34	6

At the time, we as a team had decided that the docking compartment would have the capability to use one or multiple types of space suits. Both of the above mentioned docking structures have these capabilities. Another consideration we worked with was the actual shape of the overall structure. Since the compartment is pressurized it is best to keep the compartment as round as possible since rounded structures are the most efficient in pressure loading. Adding corners would require additional mass in order to take the increased stresses at the corner. A number of shapes were considered, from a round circular shape to regular polygons of up to eight sides. This is done similarly in a study by the University of Missouri [4]. At the end of the study, we decided that an eight sided polygon would be best for a number of reasons. The corners are not very sharp, so the increase in structural weight is minimal. Also, the flat, outer sides provide a flat surface to put solar panels. We chose to accommodate as many of the groups in the team as possible. The final design is summed up in Table 36-2 below.

Table 36-2 Preliminary Docking Study Results.

Docking Structure Dimensions	Values
Length (m)	5
Diameter (m)	3.3
Mass (kg)	4,000
Volume (m^3)	15
Number of Sides	8

The next consideration is the method in which the vehicles dock. This is a key issue from the stand point of communications because there can be up to a 20 minute communication delay when the CTV is in Martian orbit. The docking group decided that the use of autonomous docking should be explored. We found sufficient information proving that autonomous docking methods succeeded in the past and could be applied in our mission. This relinquishes any problems with communication and the system docking in Martian orbit. Manned controls are also not needed as a result of this. A design and patent owned by NASA is below in Figure 36-1. The patent [5] describes how three optic devices gently glide two vehicles together in a safe manor.

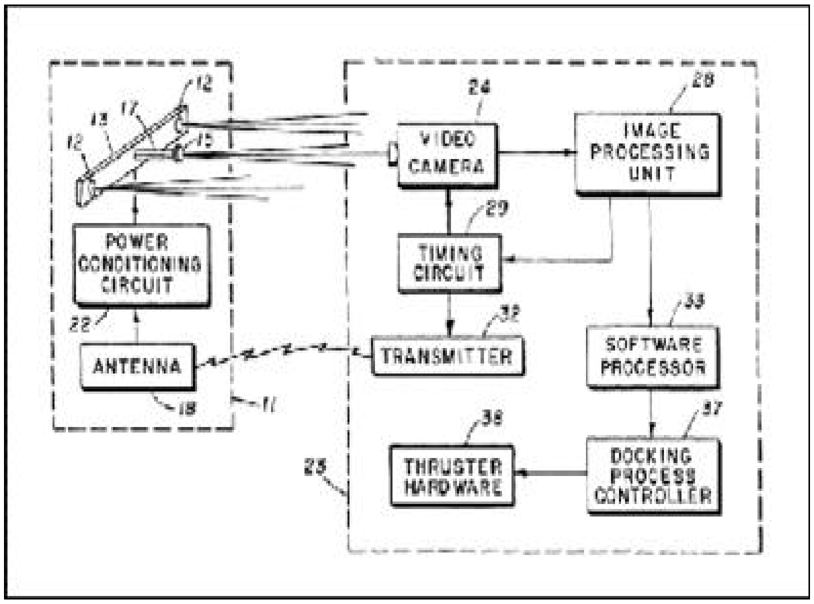


Figure 36-1 Diagram of the NASA autonomous docking system.[5]

36.1.2 Initial Sizing of the CTV

After some major assumptions were made, we could finally make an initial mass of the CTV. It is important to note that we are considering the configuration in which the crew quarters are at the end of a 25 m truss and there is a counterweight placed on a truss directly opposite of the crew quarters. The configuration just described is not the final configuration chosen for the project. This first analysis is done in MATLAB with ctv_analysis.m seen below.

In summary, the code estimates the loading that the crew quarters must support during launch, i.e. supporting an acceleration of 30 G's. To account for the compressive loads, the crew compartment walls are modeled as a monocoque cylinder. We calculate the actual stress with Equation **36-1**. Equations 36-2 through 36-4 taken from Sarafin [6].

$$\sigma = \frac{P}{2\pi r t}$$

36-1

$$\sigma_{critical} = 0.6\eta\gamma\frac{Et}{r}$$

36-2

$$\gamma = 1 - 0.901\left(1 - e^{-\phi}\right)$$

36-3

$$\phi = \frac{1}{16}\sqrt{\frac{r}{t}}$$

36-4 (for r/t < 1500)

η is the plasticity correction factor and is set equal to one for our purposes. γ is the correlation factor and is only valid when the radius to thickness ratio is less than 1,500. r is the radius and t is the thickness of the wall. The calculation of the critical stress is in question because it does not compare well to the data from Professor Sun's book [7]. We find the thickness by noticing where the actual stresses are less then the critical stress. An example plot is done with aluminum in Figure 36-2. This figure is an output of the ctv_analysis.m file. An extra safety factor of a half centimeter is added to the necessary wall thickness.

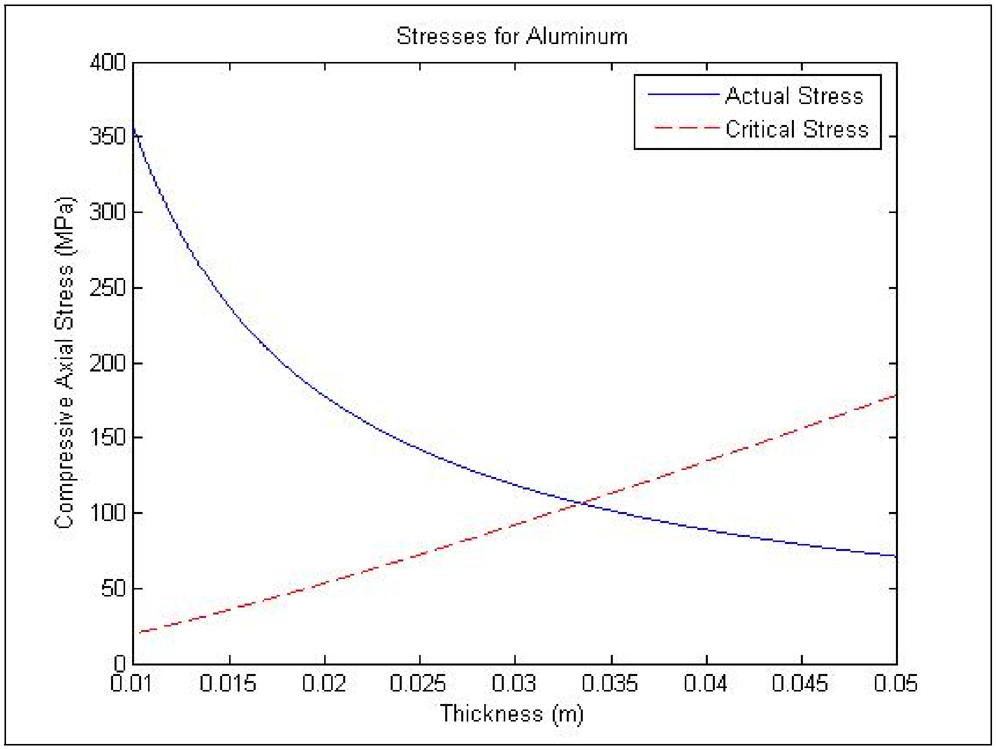


Figure 36-2 Stress Analysis from ctv_analysis.m.

The crew quarters is modeled as a cylinder with an internal volume of 120 m^3. The quarters are surrounded by the appropriate thickness determined earlier. We then calculate the total mass of the cylinder assuming a number of materials. The total mass of the CTV is then estimated using the rule of structural mass to be about 15% of the total mass as described by Osgood [8]. The results can be seen in Table 36-3. The materials chosen were taken from Sarafin's book [6].

Table 36-3 Summary of Material Trade Study.

	Cylindrical Configuration	
Material	Structural Mass (kg)	Total CTV Mass (kg)
Graphite/Epoxy Composite	29,050	206,000
Aluminum Alloy 2219-T8511	50,880	351,200
Titanium Ti-6Al-4V	67,400	461,000
Magnesium Alloy Mg-Az31B-H24	42,240	294,000
Beryllium Alloy	26,500	189,000
Combination of all Materials	52,002	358,650

The bottom line of Table 36-3 describes the masses as if all materials were combined while the other lines assume only one material. The combination of all materials considers a different ratio of each material. 50% of the structure is assumed to be aluminum alloy, 10% to be graphite/epoxy composite, 30% to be titanium, 5% to be magnesium, and 5% to be beryllium. We choose these ratios based on their common uses in spacecraft structures. In retrospect, we can see that the estimated overall value is extremely close to the final mass of the CTV completely fueled. However, the final mass was calculated for a completely different configuration and for a different set of assumptions.

```
%%%%%%%%%%%%%%%%%%%
ctv_analysis.m      (MATLAB Code)
%Aaron Sippel
%AAE 450
%Structures
%CTV Structural Analysis
%1/31/04

%This code compares the wall thicknesses needed to support the loads at
%launch as well as the pressure loads.  From this and an assumption of
%material, a rough estimate of the total structural weight of the vehicle
%can be determined.
clear
clc
%Assumptions and Constants
mest=380000;                % Pre-analysis Estimated mass
pint=101325;                % Pressure (Pa = N/m^2)
gforce=30*9.81;             % Effective Acceleration (a = m/s^2)
P=mest*gforce;              % Max Effective liftoff force (P = m*a)
r=5;                        % Radius (r = m)
```

```
t=linspace(.01,.05,2^12);       % Thickness (t = m)

%Material Properties (Aluminum Alloy 2014-T6)
E=73.1*10^3;                    % Input Youngs Modulus (MPa)
rho=2800;                       % Input density (kg/m^3)
%Graphite Composite


eta=1;                          % Plasticity correction when stress
                                % is below the material's
                                % proportional limit

gamma=1-0.901.*(1-exp(-(1/16).*sqrt(r./t)));
                                % Equation for correlation factor.
                                % valid when r/t <1500. Axial
                                % loads only

crst=(0.7)*0.6.*eta.*gamma.*E.*t./r;
                                % Critical Stress (MPa)
                                % Multiplying by 0.7 is a factor of safety



compstress=(P./(2*pi.*r.*t))./(10^6);  % Actual Compressive Stress (MPa)

index = find(crst./compstress>=1);
t1=index(1);                           % Extracts the wall thickness needed
t1=t(t1)+.005;

%Determine the wall thickness from graph
figure(1)
plot(t,compstress)
hold on
plot(t,crst,'r--')
xlabel('Thickness (m)')
ylabel('Compressive Axial Stress (MPa)')
legend('Actual Stress','Critical Stress')
title('Stresses for Aluminum')
hold off

%%%%%%%%%%%%%%%%%%Calculations of Mass
%Crew Area Cylinder
vol=120;                                % Volume of Living Space (m^3)
length=2*(vol/(pi*r^2));                % Length of cylinder (m)
matvol=pi*(t1^2)*length+t1*pi*5^2;      % Material Volume (m^3)
crewmass=matvol*rho;                    % Mass of crew area (kg)


%Counter weight
counter=crewmass                        % Mass of counterweight (kg)

%Truss weight
mtruss=1500                             % Mass of Trusses (kg)

%Central Body Approximation
mcenter=2*(mtruss+crewmass+counter)     % Central Body mass (kg)

%Structural Mass Approximation
mstruct=crewmass+counter+mtruss+mcenter % Structural mass (kg)

%TOTAL CTV MASS ESTIMATION
massctv=(mstruct/.15)+12000             % Total mass ctv (kg)
                                        % =Scaled Struct. mass + 2 airlocks
%%%%%% End of ctv_analysis.m    %%%%%%%%%%%%%%%%%%
```

36.1.3 CTV Crew Quarter Trade Study

Much of the design issues related to the crew quarters have been discussed in section 2.3.6.1 of the report. The code used to analyze the structural components appears below in the MATLAB file, composite_pressure_vessel_with_shielding.m. This code assumes a carbon fiber composite material and calculates wall thicknesses for a pressure vessel as we discussed earlier. With this code, we calculate the stringer and stiffener masses, the floor mass, and the radiation shielding mass to get a total mass of the crew compartment wall and internal structures. This code has been reiterated a number of times. We have corrected equations and rearranged the code to sustain radiation shielding on a number of equations.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
composite_pressure_vessel_with_shielding.m   (MATLAB Code)
%Aaron Sippel
%Matt Harrigan
%Structures Group
%Crew Quarters Structural Mass Calculations
%Includes use of composites

clc
clear all

%Assumptions
%Cylinder, ends have 2 times the radius
%

n_factor = .1;
non_struc_mass = 30000;
yield_stress = 1.56e9/4; %pa
E = 140e9/4; %pa
pressure = 101000; %pa
radius = 3.6; %m
radiusend=2*radius; %m
length = 5.1; %m
safety_factor = 2;
shell_safety_factor = 3
rho = 1550 %kg/m3
total_mass = 100000; %total mass including stuff on the inside
launch_g = 6;
shielding_density = 6; %g/cm^2, assume carbon is a good shield thing

%Cylinder
thick_hoop = shell_safety_factor*pressure*radius/yield_stress %m

%Cap
thick_cap = shell_safety_factor*pressure*radius/yield_stress; %m

%Mass
shell_mass = (pi*((radius+thick_hoop)^2-radius^2)*length+2*(4/3)*pi*((radiusend+thick_cap)^3-
radiusend^3))*rho %kg

shell_area = 191; %m2    from CATIA
automatic_shielding_density = shell_mass/shell_area*.1; %g/cm2
```

```
additional_shielding_density = shielding_density-automatic_shielding_density;
radiation_mass = additional_shielding_density*10*shell_area


%Stiffeners
%resists axial compression during launch
%assumes worst case load throughout length of cylinder

long_num = 50; %number of axial stiffeners
ring_num = 10; %number of ring stiffeners
stiff_width = .04 %radial dimension of axial stiffeners
stiff_length = length/(ring_num+1) %effective buckling length of axial stiffeners
end_width = .01 %length of end of the C channel

spacing = 2*pi*radius/long_num; %m, distance between stiffeners
stiff_load = total_mass*launch_g*9.8/long_num; %N

%failure due to column buckling, pinned ends
stiff_thick_1 = safety_factor*stiff_load*12*stiff_length^2/(pi^2*E*(stiff_width)^3); %m
%failure due to yielding
stiff_thick_2 = safety_factor*stiff_load/((stiff_width+2*end_width)*yield_stress); %m
%find bigger of thicknesses
stiff_thick = max([stiff_thick_1 stiff_thick_2]); %m

stiff_mass = stiff_thick*(2*end_width+stiff_width)*length*rho*long_num %kg
ring_mass = 2*radius*pi*stiff_width*stiff_thick*rho*ring_num %kg

%Floors
%floor is carbon fiber and fiberglass honeycomb sandwich, transfers load to
%equally spaced joists
%Assumes each joist carries equal load

mass_per_floor = 8000;
floor_pressure = mass_per_floor*launch_g*9.8/(pi*radius^2);
unsupported_dist = 3;
floor_thick = .02;

floor_sheet_thick = safety_factor*unsupported_dist*floor_pressure/(2*floor_thick*yield_stress);

mass_floor = (2*floor_sheet_thick + 1/500*floor_thick)*pi*(radius^2)*rho

%Legs, joints, secondary structures are an additional 100% of the mass
struc_mass = ring_mass + stiff_mass + shell_mass + 2*mass_floor + n_factor*non_struc_mass
total_mass = struc_mass + radiation_mass

%%%%%End of composite_pressure_vessel_with_shielding.m   %%%%%%%%%%%%%%%
```

36.1.4 All Catia Work

With the known dimensions of every vehicle, we design the following components in CATIA.

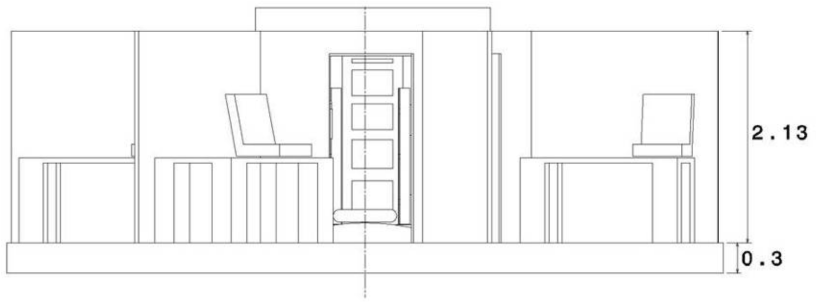


Figure 36-3 Crew Compartment first floor side view.

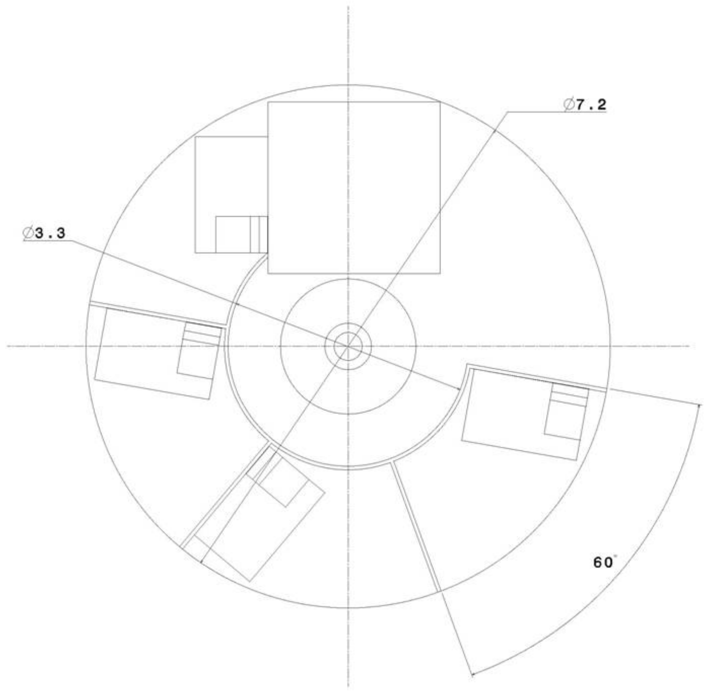


Figure 36-4 Crew Compartment first floor top view.

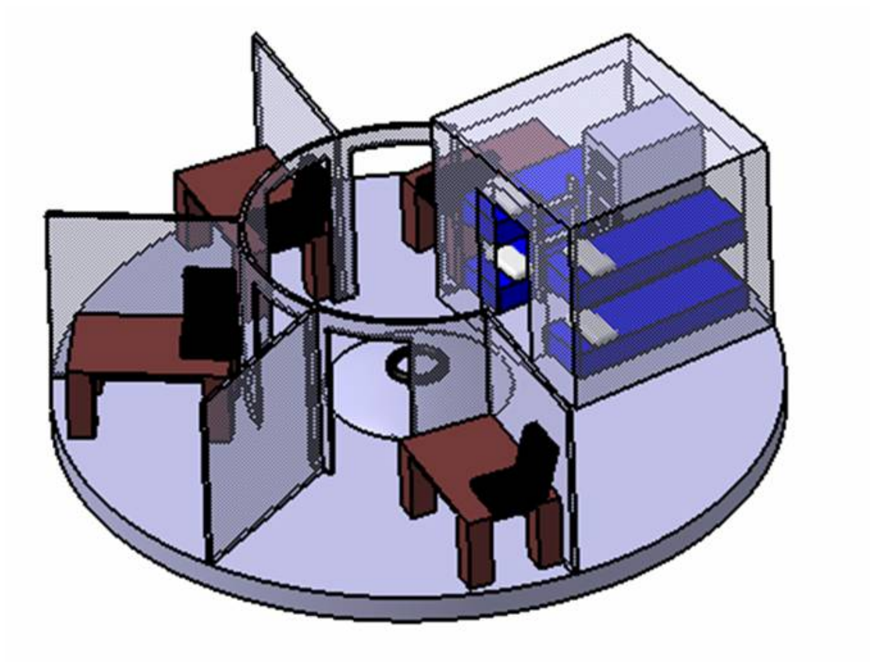


Figure 36-5 Crew quarters first first floor perspective view.

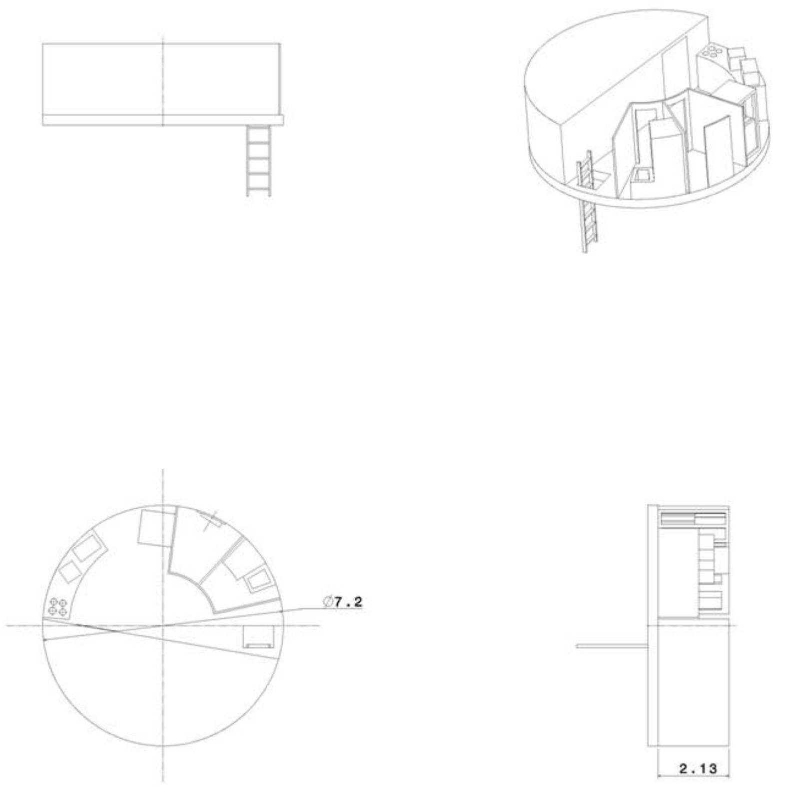


Figure 36-6 Crew quarters second floor 3 view.

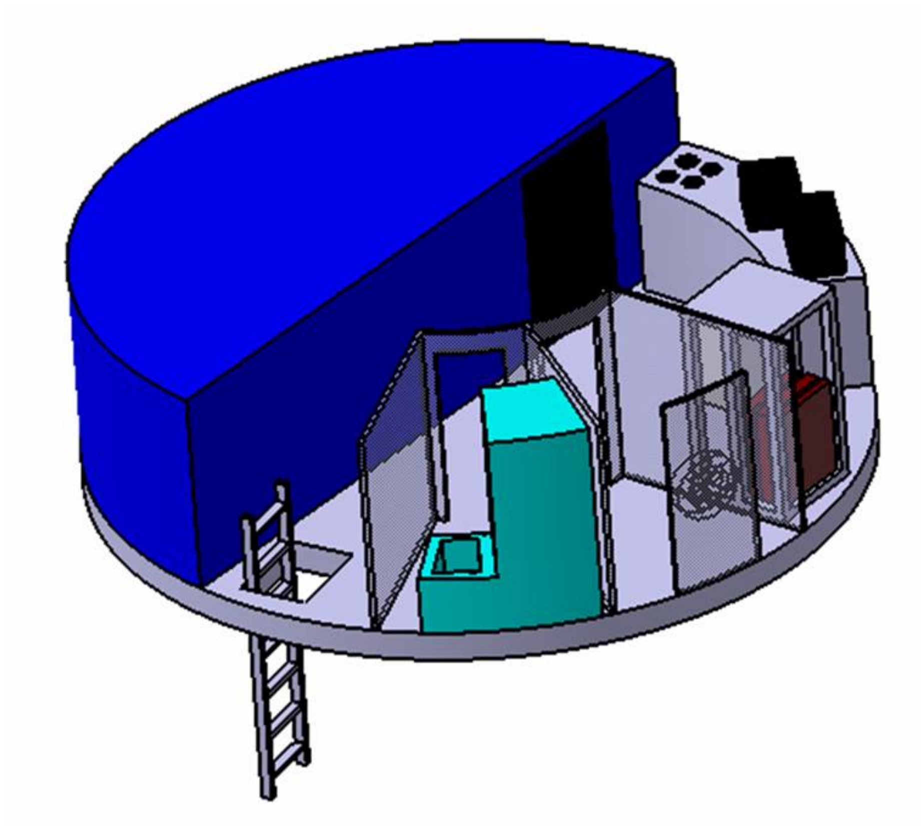


Figure 36-7 Crew quarters second floor perspective.

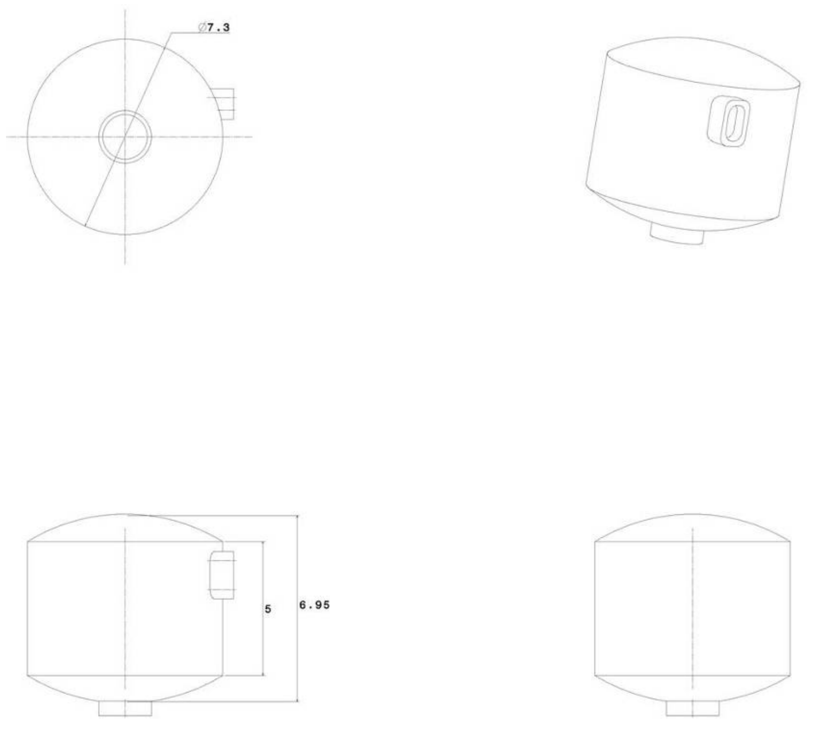


Figure 36-8 Crew capsule 3 view.

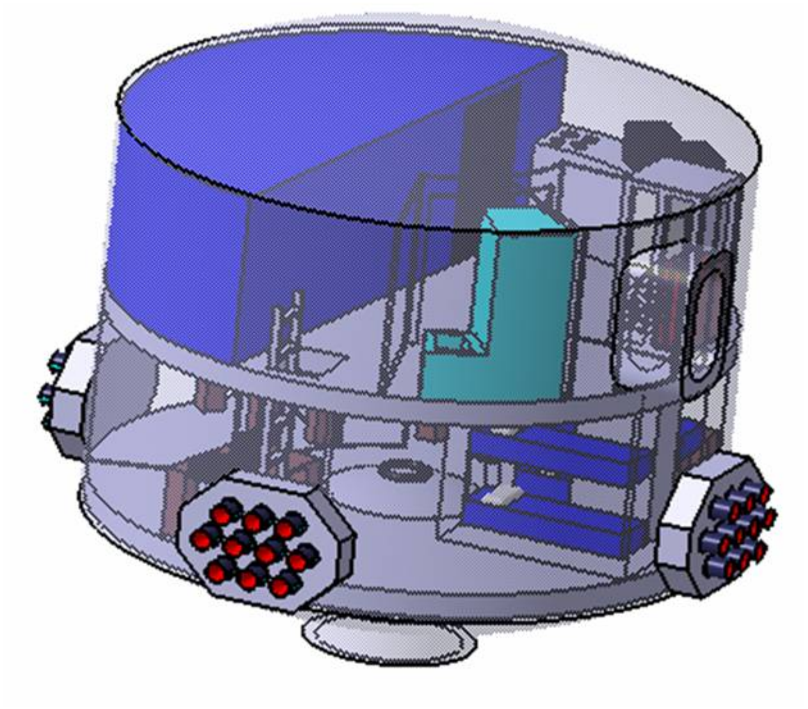


Figure 36-9 Crew capsule perspective including interior.

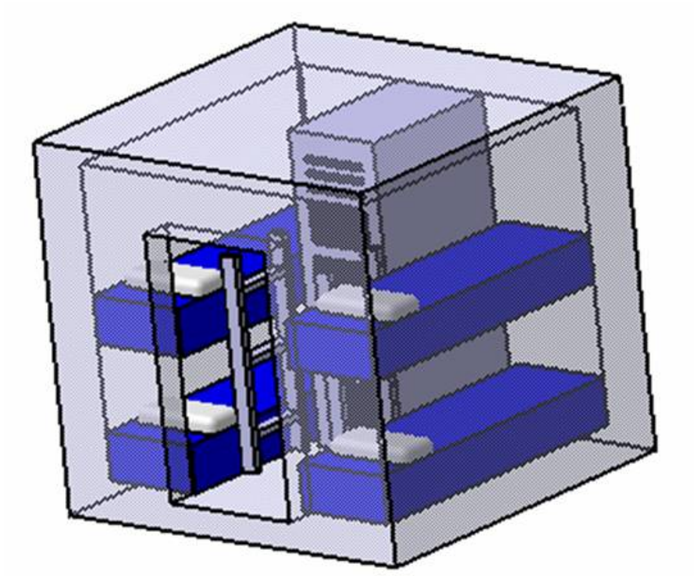


Figure 36-10 Radiation Bunker (Kathryn Bradley and Aaron Sippel).

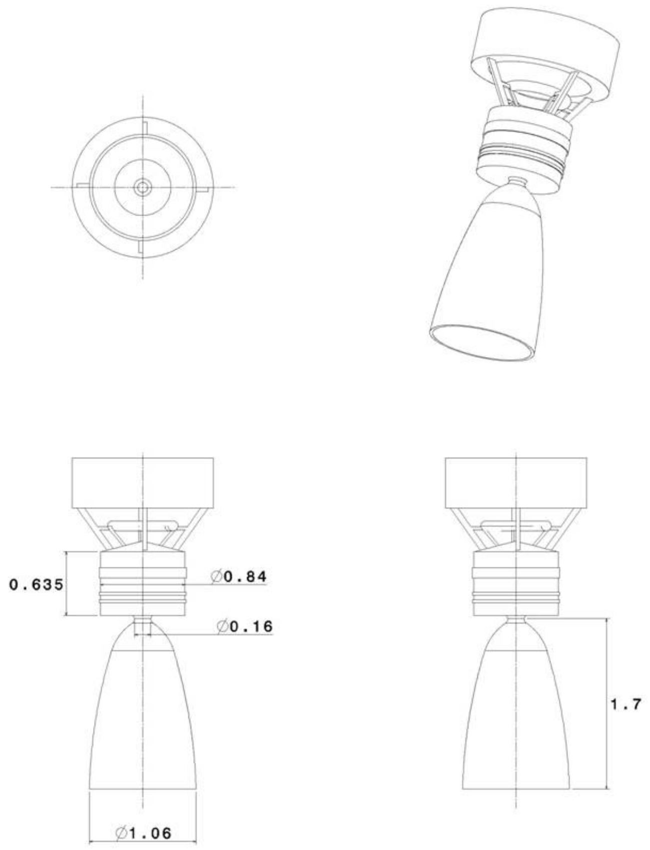


Figure 36-11 CTV nuclear engine 3 view.

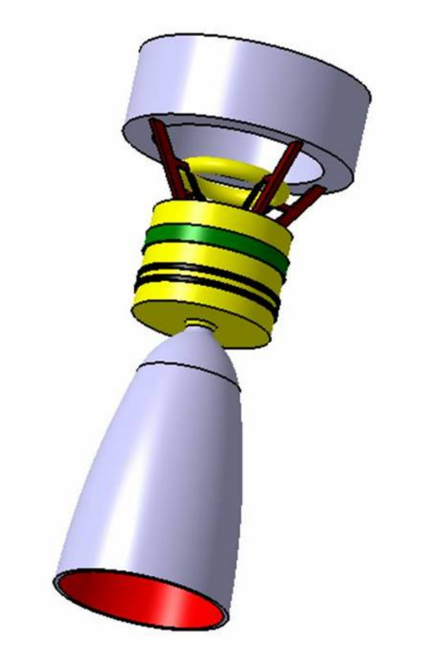


Figure 36-12 CTV nuclear engine perspective.

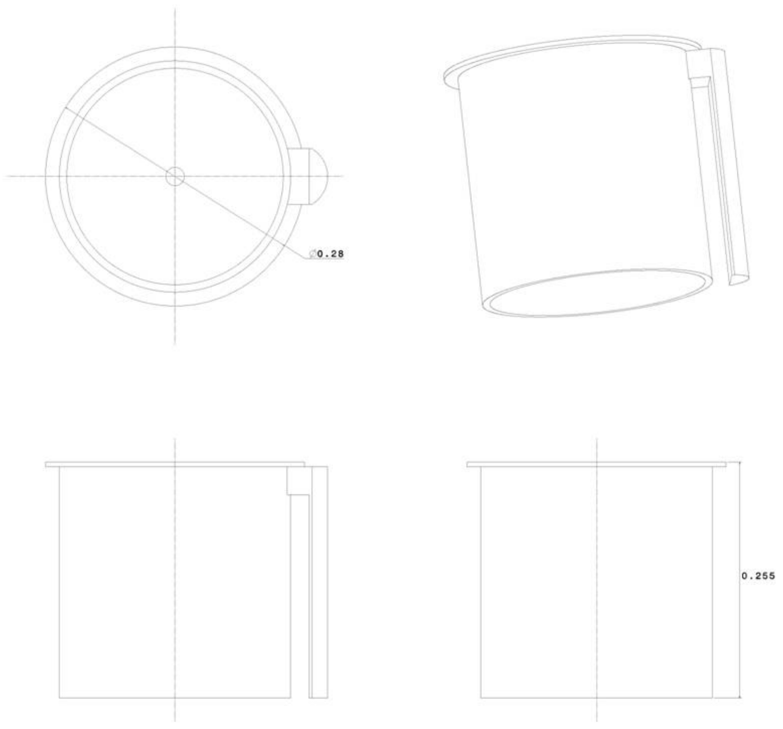


Figure 36-13 Xenon Ion Propuslion System (XIPS) 3 view.

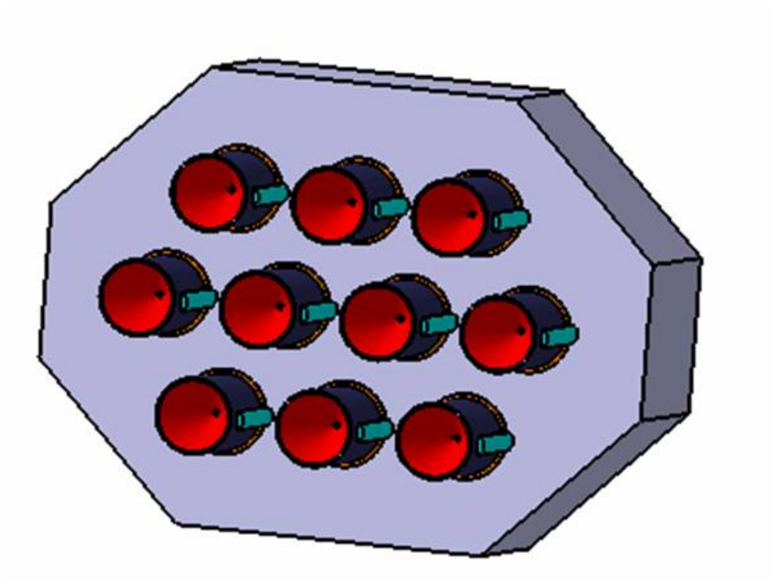


Figure 36-14 XIPS board configuration 3 view.

References

[1] "Universal Docking Module." 13 Oct. 2004. Wikipedia. 17 Jan. 2005 <http://en.wikipedia.org/wiki/Universal_Docking_Module>.

[2] "Docking Compartment." 8 Jan. 2005. Wikipedia. 17 Jan. 2005 <http://en.wikipedia.org/wiki/Docking_Compartment>.

[3] "Joint Airlock." 1 Jan. 2005. Wikipedia. 17 Jan. 2005 <http://en.wikipedia.org/wiki/Joint_Airlock>.

[4] Conceptual Design of the MR SAT Tethered Satellite Project. University of Missouri-Rolla. 17 Jan. 2005 <http://web.umr.edu/~mrsat/Documents/Conceptual%20Design/Structures%20CDD.pdf>.

[5] Synchronized Autonomous Docking Station. Marshall Space Flight Center Technology Transfer Program. 17 Jan. 2005 <http://techtran.msfc.nasa.gov/Patents/(52).html>.

[6] Sarafin, Thomas P., et al. Spacecraft Structures and Mechanisms from Concept to Launch. El Segundo, CA: Microcosm, 2003.

[7] Sun, C.T. (Chin-Teh). Mechanics of Aircraft Structures. New York: John Wiley and Sons, 1998.

[8] Osgood, Carl C. Spacecraft Structures. Englewood Cliffs, NJ: Prentice Hall Inc., 1966. 18-30.

37 Spindler, Phillip

37.1 Virtual Inertially Fixed Communications Dish Array

Author: Phillip Spindler

37.1.1 System Description

This section discusses a method to solve the communication link problem on a rotating spacecraft. The problem arises when a spacecraft spins the communications dish would have to constantly gimbal to maintain communication with Earth. A constantly gimbaling dish would break down quickly due to mechanical failure. We can solve this problem by using an array of dishes.

37.1.2 Theory and Assumptions

Figure 3-1 shows an example communications dish array. In this Figure, the spacecraft rotates about the thick cylinder and the three communications dishes are fixed the spacecraft.

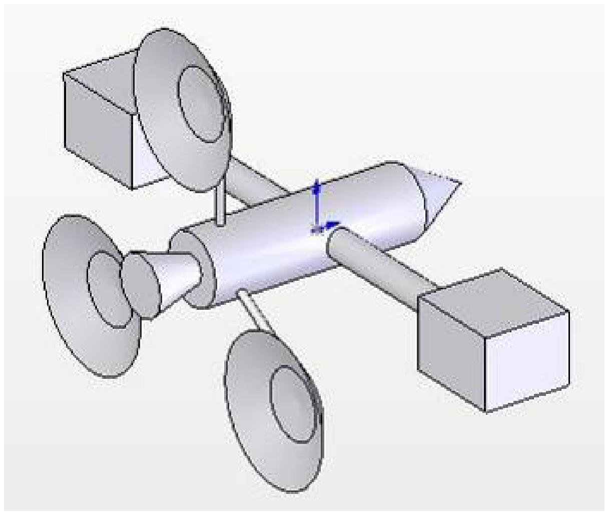


Figure 3-1 Communications Dish Array Example

The number of dishes required is calculated geometrically by finding the maximum angle the spacecraft can rotate without a given dish loosing communication with Earth. The communication signal leaves the dish in a cone shape of a specified angle. The signal then forms a circle in a plane with Earth. The dish looses communication when Earth is not inside that circle. To investigate a worst case scenario, we consider the Crew Transport Vehicle (CTV) when it is at its maximum distance from Earth. The orientation of the CTV with respect to Earth is described by one simple rotation.

37.1.3 Analysis

Figure 3-2 shows the number of dishes required to maintain constant communication with Earth. Since we cannot have half a dish, the CTV must have six communications dishes to allow for constant communication with Earth.

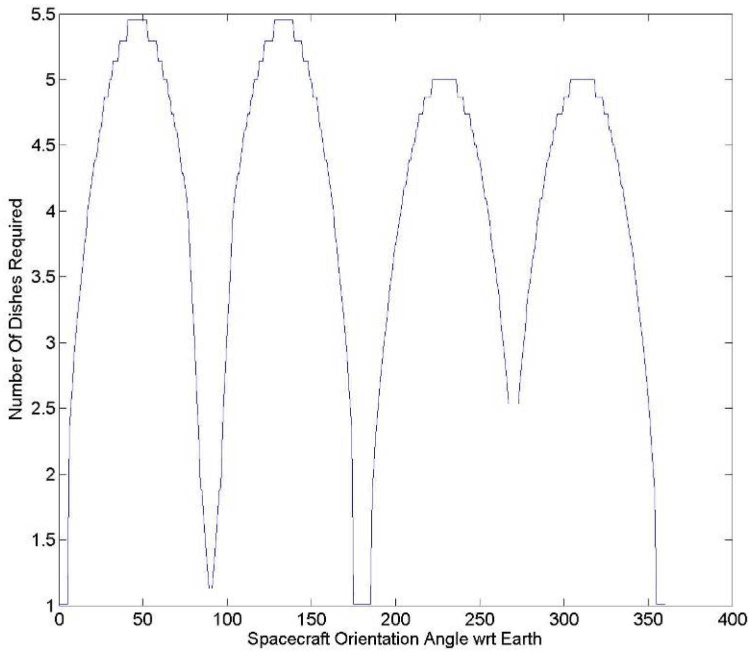


Figure 3-2 Number of Dishes Required

The code below calculates the number of dishes required to maintain constant communication with Earth.

```
%AAE 450
%Phil Spindler
%
%Size Virtual Com Dish Array

clear all
close all
clc

%this file runs the size_virtual_array function

r = 10; %radius from inertially fixed point to dish center (m)
d = 10; %diameter of dish (m)
beta = 1*pi/180; %beam spread half angle - must be > .001934 deg

%euler angles of rotation for the spacecraft.
% ie, euler angles to get from e^ to s^

Ro = 1.50e11+2.28e11; %distance between earth and spacecraft (m)

%run through all possable spacecraft orientations
index = 0;
for phideg = 0:1:360
    index = index + 1;
    phi(index) = phideg*pi/180;

[angle_between_dishes_deg(index),number_dishes(index),BAD(index)]=size_virtual_array(r,d,beta,phi(inde
x),Ro,0);
```

```
end

angle_between_dishes_deg(BAD>0) = NaN;
number_dishes(BAD>0) = NaN;

figure
plot(phi.*180./pi,angle_between_dishes_deg)
xlabel('Spacecraft Orientation Angle wrt Earth')
ylabel('Angle Between Dishes')

figure
plot(phi.*180./pi,number_dishes)
xlabel('Spacecraft Orientation Angle wrt Earth')
ylabel('Number Of Dishes Required')
```

This MATLAB code calculates how many dishes in an array are required.

```
function [angle_between_dishes_deg,number_dishes,BAD]=...
    size_virtual_array(r,d,beta,phi,Ro,displaydata)

%AAE 450 - 01.22.05
%Phil Spindler
% Calculates how many dishes in an array are required to create an
% artificial inertially fixed dish.
%
%Ro = 1.50e11+2.28e11; %distance between earth and spacecraft (m)
%r = 20; %radius from inertially fixed point to dish center (m)
%d = 10; %diameter of dish (m)
%beta = 1*pi/180; %beam spread half angle - must be > .001934 deg
%
%phi = angle of rotation of spacecraft wrt earth about s2^
%
% e1^ e3^ are in orbit plane, e2^ is up
% initially s^ corresponds to e^
% initially d^ corresponds to s^ then rotates about s1^

Re = 6378100; %Radius of earth (m)

Earth_e = [0; 0; 0]; %location of earth in e^
SC_s = [0;0;0]; %location of spacecraft in s^
SC_e = [-Ro;0;0]; %location of spacecraft in e^
D_d = [0;0;0]; %location of dish center in d^
D_s = [0;r;0]; %location of dish center in s^

BAD = 0;

%display input data

if (displaydata)
    fprintf('Radius of the Earth: %e meters \n',Re)
    fprintf('Dist Between Earth and Spacecraft: %e meters \n',Ro)
    fprintf('Dist from inertially fixed axis on SC to dish center: %0.2f meters \n',r)
    fprintf('Diameter of dish: %0.2f meters \n',d)
    fprintf('Beam Spread Half Angle: %f radians = %0.4f degrees \n\n',[beta beta*180/pi])
    fprintf('Spacecraft Angle - phi: %f radians = %0.2f degrees\n',[phi phi*180/pi])
end

%find direction cosine matrix for s^ in terms of e^ by euler angle rotation
% ie. from s^ to e^
l_se = [cos(phi) 0 sin(phi); 0 1 0; -sin(phi) 0 cos(phi)];

D_e = l_se*D_s + SC_e; %location of dish center in e^

%define d1^ by a vector from dish center to earths center
d1 = -D_e / sqrt(D_e(1)^2+D_e(2)^2+D_e(3)^2);

%project d1^ onto e1^, e2^ plane and make of length 1 (in this plane)
d1_projectedto_e1e2 = [d1(1); d1(2)]./sqrt(d1(1)^2+d1(2)^2);
```

```
%define d2^ by rotating the project above 90deg in the e1^, e2^ plane
d2 = [-d1_projectedto_e1e2(2); d1_projectedto_e1e2(1); 0];

%define d3^ by cross product
d3 = cross(d1,d2);

%define direction cosine matrix for transformation from d^ to e^
l_de = [d1 d2 d3];

%define vector from dish center towards earth based on distance from earth
%to dish center and the beam spread angle - this gives the outer radius of
%the ream (a line along the cone of the beam)
H1_d = [sqrt(D_e(1)^2+D_e(2)^2+D_e(3)^2); tan(beta)*sqrt(D_e(1)^2+D_e(2)^2+D_e(3)^2); 0];

%transform beam vector from d^ to s^
H1_s = l_se'*l_de*H1_d;

%loop through spacecraft rotations about s1^ till the circle of comm beam
%does not fully enclose earth.

IS_OKAY = 1; %test variable
i = 1; %count variable
theta = 0:1:180;
while IS_OKAY & theta(i) < 180
    if theta(i) > 179.8
        disp('WARNING - SOLUTION PROBABLY NOT VALID')
        BAD = 1;
    end

    %define matrix to perform rotation about s1^ by angle theta
    l_rotate_about_s1 = [1 0 0;0 cos(theta(i)*pi/180) sin(theta(i)*pi/180);0 -sin(theta(i)*pi/180)
cos(theta(i)*pi/180)];

    %rotate the dish
    D_rotated_s = l_rotate_about_s1*D_s;

    %define dish location from e^ origin in d^
    D_rotated_d = l_de'*l_se*D_rotated_s - SC_e;

    %calculate the distance that D is from origin of d^ in the
    %d2^,d3^ plane
    D_rotated_2d_d = D_rotated_d;
    D_rotated_2d_d(abs(D_rotated_2d_d)>abs(max(D_rotated_2d_d)/10)) = 0;
    D_rotated_2d_length_d = sqrt(max(abs(D_rotated_2d_d))^2 + median(abs(D_rotated_2d_d))^2);

    %rotate the beam vector about s1^ by theta
    H1_rotated_s = l_rotate_about_s1*H1_s;

    %convert from s^ to e^ then to d^
    H1_rotated_d = l_de'*l_se*H1_rotated_s;

    %find beam radius in the d2^, d3^ plane
    H1_rotated_d(abs(H1_rotated_d)>abs(max(H1_rotated_d)/10)) = 0;
    H1_rotated_d_length = sqrt(max(abs(H1_rotated_d))^2 +median(abs(H1_rotated_d))^2);

    %check to make sure the circle the beam cuts out in the d2^, d3^ plane
    % encloses the earth
    IS_OKAY = D_rotated_2d_length_d < (H1_rotated_d_length-Re)/2;
    i = i + 1; %iterate count variable
end

angle_between_dishes = theta(i-2)*2*pi/180;
angle_between_dishes_deg = angle_between_dishes*180/pi;
number_dishes = 360/angle_between_dishes_deg;

if (displaydata)
    fprintf('The center of the dishes can be %0.0f degrees apart \n',angle_between_dishes_deg)
    fprintf('For constant communication you need %0.3f dishes \n\n',number_dishes)
end

if (2*pi*r/number_dishes < d)
```

```
        disp('---------- WARNING ----------')
        disp('The number of dishes required will not fix based on the ')
        disp('dish diameter and radius to dish center input.  Try ')
        disp('changing the geometry input or increase the beam spread angle.')
        disp('-----------------------------')
        BAD = 1;
    end
```

37.2 Optimization of CTV Date for Mars Departure

Author: Phillip Spindler

37.2.1 Description

This section outlines the methods used to find the optimum date for the Crew Transport Vehicle (CTV) to leave Mars for Earth return.

37.2.2 Theory and Assumptions

We use two independent methods during this analysis. The first method is a simplified orbital analysis which assumes circular orbits. This method finds the number of days prior to opposition the CTV must leave to achieve a Hohmann transfer. See the description in the Analysis section and [1]. The second method employs the program Midas, which is distributed by the Jet Propulsion Laboratory (JPL). Midas calculates the optimum trajectory to minimize the amount of propellant used during the mission.

37.2.3 Analysis

Using the circular orbit method, the semi-major axis of the transfer orbit, shown in Figure 3-3, is calculated using Equation 3-1.

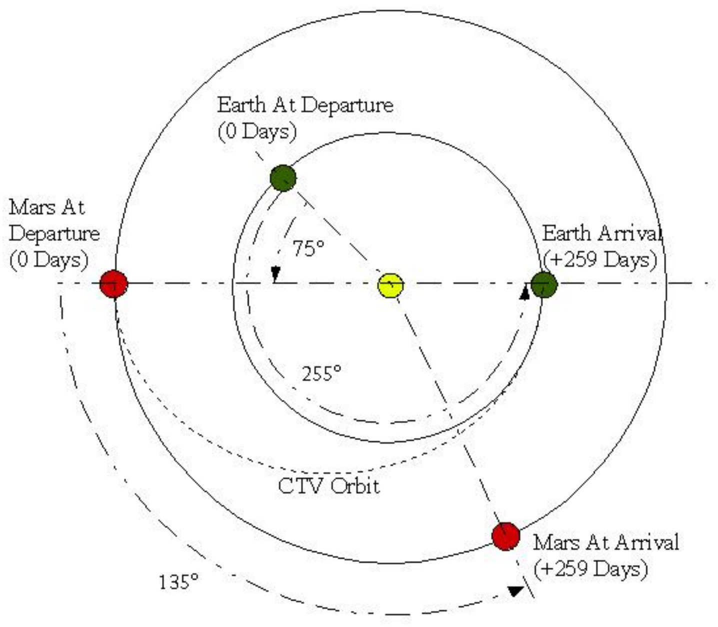


Figure 3-3 CTV Hohmann Transfer Return Trajectory Using Circular Orbits

$$a = (r_{perihelion} + r_{aphelion})/2 = (1 + 1.52)/2 = 1.26AU$$

Equation 3-1 Semi-Major Axis

We find the eccentricity of the CTV return orbit using Equation 3-2 and the orbital period using Equation 3-3.

$$e = a - \frac{r_{aphelion}}{a} = 1 - \frac{1}{1.26} = 0.21$$

Equation 3-2 Eccentricity

$$P = a^{\frac{3}{2}} = 1.26^{\frac{3}{2}} = 518days$$

Equation 3-3 Orbital Period

The transfer from Mars to Earth takes half the orbital period as shown in Equation 3-4.

$$TOF = \frac{P}{2} = 259days$$

Equation 3-4 Time Of Flight

We conclude that the CTV must leave Mars 259 days before Earth is in the position where the CTV will intersect it. Earth's orbital period is 1 year, so it moves 0.99°/day. Mars' orbital period is 1.89 years, so it moves 0.52°/day. In the 259 day transfer Mars will move 136° and Earth will move 255°. When the CTV leaves Mars, Earth must be (255°-180°) = 75° behind Mars. The relative rate that Earth moves with respect to Mars is (0.99°/day -0.52°/day) = 0.47°/day. Therefore the Earth will catch up to Mars (the 75°) in 164 days. This means the CTV should leave Mars 162 days before opposition with Earth. We can now calculate the optimal dates to leave Mars based on a table of the opposition dates.

The second method employs the JPL program named Midas. This method is more accurate than the first method because it does not assume circular orbits. Midas is used to optimize a trajectory with a Mars departure date every 5 days for a 19 year time span starting with the first Earth departure date. This method allows us to see the effect of the departure date instead of simply assuming Midas correctly finds the optimal date. An automated process using several Perl scripts to batch run Midas is shown below. The first script generates Midas input files for each possible leave date. The second script runs Midas for all the input files created. The third script extracts relevant information from the Midas output files and puts it into a text file. The MATLAB script below plots the output data from Midas. Figure 3-4 outlines the trajectories optimized by Midas. Note that different time of flight guesses were input to Midas which led Midas to different optimal trajectories. The idea trajectory is in the lower left of the plot, which gives a low time of flight and a low required ΔV.

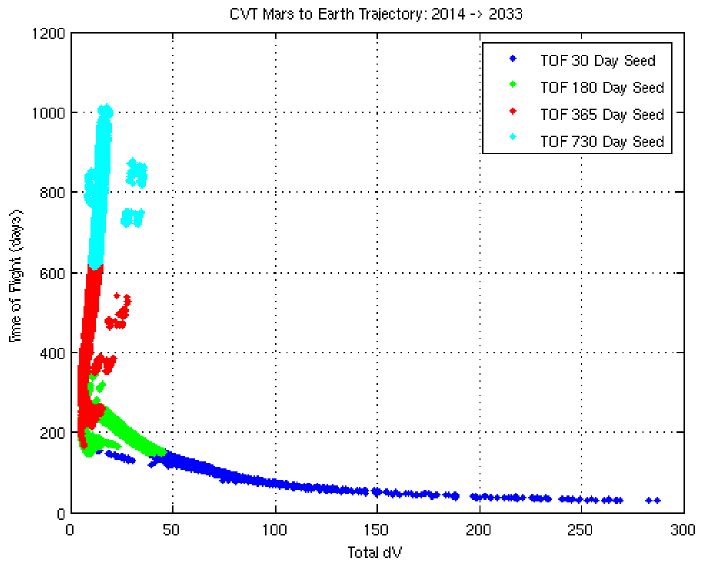


Figure 3-4 Optimized CTV Return Trajectory

From this analysis we also see the leave date sensitivity. Once a fuel use requirement for the CTV return trajectory is set, then any additional fuel available on the CTV can be used to leave at a non-optimal date. An example of this is shown in Figure 3-5, note that this Figure plots ΔV which is directly relatable to fuel usage. Consider the ΔV available is defined by the horizontal line. Any time the shaded areas are below this line the CTV would be able to leave Mars orbit. We note that for a given optimal leave date, the bottom most point of each stalagmite on the plot, there is a window in which the CTV could leave. This window is normally about 70 days.

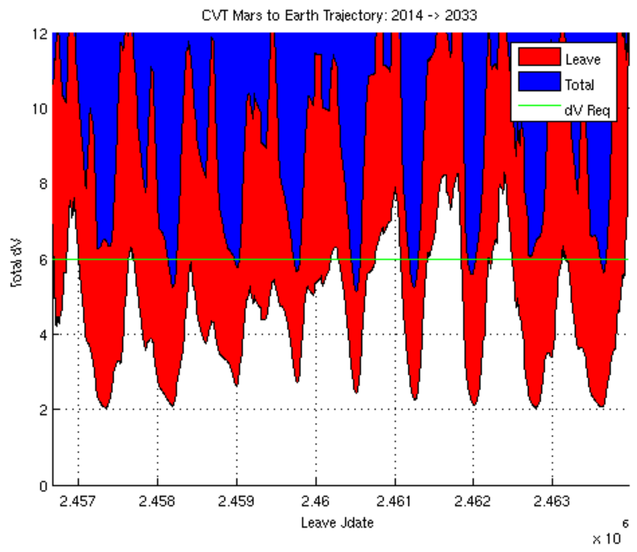


Figure 3-5 Leave Date Sensitivity

1) This Perl script generates Midas input files for every 5 days of every month from 2014 to 2033.

```
#/usr/bin/perl

@days = ("01",”05”,”10”,"15",”20”,”25”);
@months = ("01","02","03","04","05","06","07","08","09","10","11","12");
@years =
("2014","2015","2016","2017","2018","2019","2020","2021","2022","2023","2024","2025","2026","2027","20
28","2029","2030","2031","2032","2033");

$title = "CTV Mars to Earth Trajectory";
$leaving_body ="MARS";
$arrival_body = "EARTH";
$leaving_alt = "350";
$arrival_Ra = "115000";
$arrival_Rp = "200";
$TOF = "260";

foreach $year (@years) {
    foreach $month (@months) {
            foreach $day (@days) {
                    $filename = "CTV_mars2earth_" . $year . "-" . $month . "-" . $day . "_" . $TOF .
"\.inp";
                    open(OUTF,">$filename");
                            print OUTF "HEAD=\'$title\'\n";
                            print OUTF "SHOTA=\'$leaving_body\'\n";
                            print OUTF "BULSI=\'$arrival_body\'\n";
                            print OUTF "JDL=$year,$month,$day\n";
                            print OUTF "nda=2\n";
                            print OUTF "re=$leaving_alt\n";
                            print OUTF "ra=$arrival_Ra\n";
                            print OUTF "rp=$arrival_Rp\n";
                            print OUTF "jdate=0.0\n";
                            print OUTF "adate=$TOF\n";
                            print OUTF "varyi=\'adate\'\n";
                            print OUTF "loops=50, \$\n";
                            print OUTF "\$END\n";
                    close(OUTF);
            }
    }
}
```

2) This Perl script runs Midas for every input file in the current directory.

```
#/usr/bin/perl

$initial_dir = ".";

opendir(INT_DIR,"$initial_dir");
    @files = grep /\.inp/i, readdir INT_DIR;
closedir(INT_DIR);

foreach $file (@files) {
    system("midas","-s",$file);
    system("rm",$file);
}
```

3) This Perl script gets relevant data from all the Midas output files and stores it in a text file.

```
#/usr/bin/perl
#hca - transfer angle
#dvl - delta v leaving
#dvt - total delta v
#vhl - launch v inf
#vhp - arrival v inf

$initial_dir = ".";

opendir(INT_DIR,"$initial_dir");
    @files = grep /\.out/i, readdir INT_DIR;
closedir(INT_DIR);

$_ = @files[1];
/[\w\d]+_(\d)+\.out//gi;

$filename = "output" . $_ . ".txt";
open(OUTF,">$filename");

print OUTF "Filename\tLeave Date\tArrival Date\tLeave JDate\t Arrival JDate\tTime of Flight\tLeaving
dV\tTotal dV\tLeaving Vinf\tArrival Vin\tTransfer Angle  \n";

foreach $file (sort @files) {
    ##### open all the files #####
    open(INF,"$file");
            @content = <INF>;
    close(INF);

    print "$file\n";

    ##### remove all line breaks #####
    foreach $line (@content) {
            chomp($line);
    }

    ##### convert content array to string #####
    $content = join("",@content);

    ##### define local variable as content string #####
    $_ = $content;

    ##### find hca #####
    if (/hca\s+(\d+)\.(\d+)/i) {
            $transfer_angle = $1 . "\." . $2;
    }

    ##### find dvl #####
    if (/dvl\s+(\d+)\.(\d+)/i) {
            $leaving_dv = $1 . "\." . $2;
    }
    ##### find dvt #####
    if (/dvt\s+(\d+)\.(\d+)/i) {
          $total_dv = $1 . "\." . $2;
    }

    ##### find vhl #####
    if (/vhl\s+(\d+)\.(\d+)/i) {
          $leaving_vinf = $1 . "\." . $2;
    }

    ##### find vhp #####
    if (/vhp\s+(\d+)\.(\d+)/i) {
          $arrival_vinf = $1 . "\." . $2;
    }

    ##### find leave date #####
    if (/Departure:[\w\s]+:[\s\d]+\.[\s\d]+\w+\s+(\w+)\s+(\d+),\s+(\d+)\s+\d+:\d+:\d+\s+(\d+)/i) {
          $leaving_date = $1 . "\." . $2 . "\." . $3;
          $leaving_jdate = $4;
    }

    ##### find arrive date #####
    if (/Arrival:[\w\s]+:\s+(\d+)\.(\d+)\s+\w+\s+(\w+)\s+(\d+),\s+(\d+)\s+\d+:\d+:\d+\s+(\d+)/i) {
          $TOF = $1 . "\." . $2;
          $arrival_date = $3 . "\." . $4 . "\." . $5;
          $arrival_jdate = $6;
    }

    print OUTF $file . "\t" . $leaving_date . "\t" . $arrival_date . "\t" . $leaving_jdate . "\t" .
$arrival_jdate . "\t" . $TOF . "\t" . $leaving_dv . "\t" . $total_dv . "\t" . $leaving_vinf . "\t" .
$arrival_vinf . "\t" . $transfer_angle . "\t" . "\n";

    system("rm",$file);

}

close(OUTF);
```

4) This MATLAB script opens the text file created from the above script and plots the data.

```
clear
close all
clc

data = dlmread('output30.txt','\t',2,3);

leavej_1 = data(:,1);
arrivej_1 = data(:,2);
tof_1 = data(:,3);
leavedv_1 = data(:,4);
totaldv_1 = data(:,5);
leavevinf_1 = data(:,6);
arrivevinf_1 = data(:,7);
angle_1 = data(:,8);

figure
plot(totaldv_1,tof_1,'b.')
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 30 Days')
xlabel('Total dV')
ylabel('Time of Flight (days)')
grid

figure
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 30 Days')
subplot(2,1,1)
plot(leavej_1,totaldv_1,'b.')
grid
xlabel('Leave Jdate')
```

```
ylabel('Total dV (km/s)')
subplot(2,1,2)
plot(leavej_1,tof_1,'b.')
grid
xlabel('Leave Jdate')
ylabel('Time of Flight (days)')

%%%%%%%%%%%%%%

data = dlmread('output180.txt','\t',2,3);

leavej_2 = data(:,1);
arrivej_2 = data(:,2);
tof_2 = data(:,3);
leavedv_2 = data(:,4);
totaldv_2 = data(:,5);
leavevinf_2 = data(:,6);
arrivevinf_2 = data(:,7);
angle_2 = data(:,8);

figure
plot(totaldv_2,tof_2,'g.')
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 180 Days')
xlabel('Total dV')
ylabel('Time of Flight (days)')
grid

figure
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 180 Days')
subplot(2,1,1)
plot(leavej_2,totaldv_2,'g.')
grid
xlabel('Leave Jdate')
ylabel('Total dV (km/s)')
subplot(2,1,2)
plot(leavej_2,tof_2,'g.')
grid
xlabel('Leave Jdate')
ylabel('Time of Flight (days)')


%%%%%%%%%%%%%%%%%%%%%%

data = dlmread('output365.txt','\t',2,3);

leavej_3 = data(:,1);
arrivej_3 = data(:,2);
tof_3 = data(:,3);
leavedv_3 = data(:,4);
totaldv_3 = data(:,5);
leavevinf_3 = data(:,6);
arrivevinf_3 = data(:,7);
angle_3 = data(:,8);

figure
plot(totaldv_3,tof_3,'r.')
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 365 Days')
xlabel('Total dV')
ylabel('Time of Flight (days)')
grid

figure
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 365 Days')
subplot(2,1,1)
plot(leavej_3,totaldv_3,'r.')
grid
xlabel('Leave Jdate')
ylabel('Total dV (km/s)')
subplot(2,1,2)
plot(leavej_3,tof_3,'r.')
grid
```

```
xlabel('Leave Jdate')
ylabel('Time of Flight (days)')

%%%%%%%%%%%%%%%%%%%%%%%

data = dlmread('output730.txt','\t',2,3);

leavej_4 = data(:,1);
arrivej_4 = data(:,2);
tof_4 = data(:,3);
leavedv_4 = data(:,4);
totaldv_4 = data(:,5);
leavevinf_4 = data(:,6);
arrivevinf_4 = data(:,7);
angle_4 = data(:,8);

figure
plot(totaldv_4,tof_4,'c.')
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 730 Days')
xlabel('Total dV')
ylabel('Time of Flight (days)')
grid

figure
title('CVT Mars to Earth Trajectory: 2014 -> 2033 - Seed 730 Days')
subplot(2,1,1)
plot(leavej_4,totaldv_4,'c.')
grid
xlabel('Leave Jdate')
ylabel('Total dV (km/s)')
subplot(2,1,2)
plot(leavej_4,tof_4,'c.')
grid
xlabel('Leave Jdate')
ylabel('Time of Flight (days)')

%%%%%%%%%%%%%%

figure
plot(totaldv_1,tof_1,'b.',totaldv_2,tof_2,'g.',totaldv_3,tof_3,'r.',totaldv_4,tof_4,'c.')
title('CVT Mars to Earth Trajectory: 2014 -> 2033')
xlabel('Total dV')
ylabel('Time of Flight (days)')
grid
legend('TOF 30 Day Seed','TOF 180 Day Seed','TOF 365 Day Seed','TOF 730 Day Seed')

% combined_data = [leavej_1 leavedv_1 totaldv_1;leavej_2 leavedv_2 totaldv_2;leavej_3 leavedv_3
totaldv_3;leavej_4 leavedv_4 totaldv_4];
% combined_data = sortrows(combined_data,1);


for i = 1:1:length(leavej_1)
    tdv = [totaldv_1(i) totaldv_2(i) totaldv_3(i) totaldv_4(i)];
    leavej = [leavej_1(i) leavej_2(i) leavej_3(i) leavej_4(i)];
    ldv = [leavedv_1(i) leavedv_2(i) leavedv_3(i) leavedv_4(i)];
    best = find(tdv==min(tdv));
    best = best(1);
    combined_data(i,:) = [leavej(best);ldv(best);tdv(best)];

end

combined_data = [combined_data; combined_data(end,1) max(combined_data(:,3)) max(combined_data(:,3))];
combined_data = [combined_data(1,1) max(combined_data(:,3)) max(combined_data(:,3)); combined_data];
combined_data2 = combined_data;
combined_data2(:,2) = smooth(combined_data2(:,2),20);
combined_data2(:,3) = smooth(combined_data2(:,3),20);

figure
hold on
fill(combined_data(:,1),combined_data(:,2),'r',combined_data(:,1),combined_data(:,3),'b')
plot(combined_data(:,1),combined_data(:,1).*0+6,'g-')
```

```
hold off
title('CVT Mars to Earth Trajectory: 2014 -> 2033')
xlabel('Leave Jdate')
ylabel('Total dV')
grid
legend('Leave','Total','dV Req')
axis([min(leavej_1) max(leavej_1) 0 12])


figure
hold on
fill(combined_data2(:,1),combined_data2(:,2),'r',combined_data2(:,1),combined_data2(:,3),'b')
plot(combined_data2(:,1),combined_data2(:,1).*0+6,'g-')
hold off
title('CVT Mars to Earth Trajectory: 2014 -> 2033')
xlabel('Leave Jdate')
ylabel('Total dV')
grid
legend('Leave','Total','dV Req')
axis([min(leavej_1) max(leavej_1) 0 12])
```

37.2.4 References

[1] Thomas, David. Orbital Transfers Between Planetary Orbits: An Example. Ball State University. 02.14.05 http://www.cs.bsu.edu/homepages/dathomas/SpaceGrant/OrbitalTransfer.htm

38 Stalbaum, John

38.1 CTV Stationkeeping Analysis

Created by: John Stalbaum

This code executes a CTV stationkeeping analysis. Phil Spindler goes into more depth on how this code operates in his chapter 2 section on CTV stationkeeping. The next code is a Hohmann transfer function.

The next codes are a wrapper for a high definition Earth atmosphere empirical correlation, the NRLMSISE-00 model. This model is used for calculating the orbital decay of satellites often. The purpose of the wrapper program is to create a Windows executable file of the C program modeling this correlation. From this, a Matlab function is created which contains the means for calling this executable. The result is a Matlab function that can be given an altitude, and will return the density of the atmosphere to 20 decimal places (and conceivably more, if necessary). The full source code and a detailed description of the atmosphere model used can be found at reference [1].

Code 1 – CTV Stationkeeping Analysis Code ("stationkeeping.m")

```
% AAE 450 CTV Stationkeeping code
% By John Stalbaum

clear all; close all; clc;

% Area Input
% -----------------------------------------
boom_area = 3*(.1*20 + 2.85*.05*5);  % Triangluar booms flying as a rectangle
crew_area = 7.5*5;
power_area = 7.5*5;
MLV_area = 20*10/2;
tanks_area = 20*8;
total_area = boom_area*2 + crew_area + MLV_area + tanks_area + power_area;
% -----------------------------------------

% Mass Input
% -----------------------------------------
m_boom = 8000;
crew_mass = 50000;
power_mass = 10000;
MLV_mass = 25000;
tanks_mass = 130000;
total_mass = m_boom*2 + crew_mass + MLV_mass + tanks_mass + power_mass;
% -----------------------------------------


m_empty = total_mass;

MU = 398600;           % km^3/s^2
Isp = 3800;
re = 6356.9e3;

duration = linspace(1,9,9)*365/12;
r_orbit = [200000 : 1000 : 300000]+re;

%bgrcmyk
```

```
ltype = ['b: '; 'b- '; 'b-.'; 'b--'; 'g: '; 'g- '; 'g-.'; 'g--'; 'r: '; 'r- '; 'r-.'; 'r--'];

hold on;
legendstring = [];
for outer = 1 : length(r_orbit)
    v_orbit = sqrt(MU*1000/r_orbit(outer))*1000;
    rho = rhoatm(r_orbit(outer)-6356.9e3);
    q = 1/2*rho*v_orbit^2;
    D = 1*q*total_area; % Drag force
    nengines(outer) = ceil(D/0.165);
    P(outer) = 4.5*D/0.165; % Engines consume 4.5 KW
    dVday = D/total_mass*60^2*24; % delta-V incurred per day.
    for inner = 1 : length(duration)
        m0 = total_mass*exp(duration(inner)*dVday/(Isp*9.8));
        mp(outer,inner) = m0-total_mass;
    end
    [r, c] = size(legendstring);
    if (r_orbit(outer) - re)/ 10000 == round((r_orbit(outer) - re)/ 10000)
        plot(duration*12/365,mp(outer,:),ltype(r+1,:))
        legendstring = [legendstring; ['Altitude = ' num2str(r_orbit(outer)-6356.9e3)]];
    end
    if outer ~= 1
        mhohmann = exp(hohmann(200,(r_orbit(outer)-re)/10^3)*10^3/(9.8*450))*total_mass-total_mass;
        transfer_totalmass(outer) = mp(outer,length(duration))+ mhohmann;
    else
        transfer_totalmass(outer) = mp(outer,length(duration));
    end
end
title('Propellant Costs for CTV Stationkeeping at Various Durations and Orbits');
xlabel('Stay duration (months)');
ylabel('Propellant Cost (kg)');
legend(legendstring(1,:),legendstring(2,:),legendstring(3,:),legendstring(4,:),legendstring(5,:),legen
dstring(6,:),legendstring(7,:),legendstring(8,:),legendstring(9,:),legendstring(10,:),legendstring(11,
:));
figure(2);
[AX,H1,H2] = plotyy((r_orbit-re)/10^3,nengines,(r_orbit-re)/10^3,P);
set(get(AX(1),'Ylabel'),'String','# of Engines');
set(get(AX(2),'Ylabel'),'String','Power Requirement (kW)');
set(get(AX(1),'Xlabel'),'String','Orbital Altitude (km)');
title('Hardware and Power Requirements of Stationkeeping System');
set(H1,'LineWidth',3)
set(H2,'LineWidth',3)
figure(3);
plot((r_orbit-re)/10^3,transfer_totalmass);
xlabel('Orbital Altitude (km)');
ylabel('Total propellant required for boost + stationkeeping (kg)');
title('Minimum Propellant Usage for Hohmann Orbital Boost and Stationkeeping Combination');

[smallest, ind] = min(transfer_totalmass);
disp(['The minimum propellant usage for construction occurs at an orbital altitude of '
num2str((r_orbit(ind)-re)/10^3) ' kilometers. ' num2str(smallest) ' kg of propellant is used, and '
num2str(nengines(ind)) ' engines are required. Power usage is ' num2str(P(ind)) ' kW.']);

% Force requrement for this is:
% 2.8281 N;
% For Boeing 702:
% "This has an ISP of 3800 seconds, requires 4.5 kw, and provides a thrust
% of 0.165 N. This used to be a Hughes product."
% 50 kW hall thruster. (inferior, mathematically):
% "provided up to 2.9 newtons and ISP from 1750 to 3250 seconds."
% Data from http://www.spacetethers.com/thrust.html
```

Code 2 – Hohmann Transfer Code (“hohmann.m”)

```
function dv = hohmann(ha,hb);
% Calculates the delta-V needed for a Hohmann transfer from orbital
% altitude ha to orbital altitute hb.  By John Stalbaum.  Inputs in
% kilometers, and output in km/s.

re = 6356.9;
ra = re+ha;
```

```
rb = re + hb;
atx = (ra+rb)/2;
dv = 631.3481*(abs((2/ra-1/atx)^(1/2)-(1/ra)^(1/2))+abs((2/rb-1/atx)^(1/2)-(1/rb)^(1/2)));
```

Code 3 – Matlab Wrapper for NRLMSISE-00 Atmospheric Model, Matlab Side ("rhoatm.m")

```
% Wrapper for increased fidelity Earth atmospheric model.
% Written by John Stalbaum for AAE 450

function rho = rhoatm(altitude);

[s, w] = unix(['rhoatm ' num2str(altitude)]);
rho = str2num(w);

% End of Script
```

Code 4 - Matlab Wrapper for NRLMSISE-00 Atmospheric Model, C Side ("rhoatm.c")

```
/* Adapted by John Stalbaum for AAE 450, CTV Stationkeeping Analysis   */

/* This code was adapted from the NRLMSISE-00 test case.  takes in a   */
/* value of altitude, and returns the atmospheric density to 20        */
/* decimal places.                                                     */


/* ------------------------------------------------------------------- */
/* ------------------------------ INCLUDES --------------------------- */
/* ------------------------------------------------------------------- */

#include <stdio.h>
#include <stdlib.h>
#include "nrlmsise-00.h"



/* ------------------------------------------------------------------- */
/* ----------------------------- TEST_GTD7 --------------------------- */
/* ------------------------------------------------------------------- */

void test_gtd7(double altitude) {
    struct nrlmsise_output output;
    struct nrlmsise_input input;
    struct nrlmsise_flags flags;
    struct ap_array aph;
    int i;
    /* input values */
    for (i=0;i<7;i++)
            aph.a[i]=100;
    flags.switches[0]=0;
    for (i=1;i<24;i++)
            flags.switches[i]=1;
    input.doy=172;
    input.year=0; /* without effect */
    input.sec=29000;
    input.alt=altitude/1000; // Altitude in meters.
    input.g_lat=60;
    input.g_long=-70;
    input.lst=16;
    input.f107A=150;
    input.f107=150;
    input.ap=4;
    gtd7d(&input, &flags, &output);
    printf("%1.30f\n",output.d[5]*100*100*100/1000);
}


/* ------------------------------------------------------------------- */
/* ------------------------------- MAIN ------------------------------ */
/* ------------------------------------------------------------------- */

int main(int argc, char **argv) {
    double altitude;
    char *blah;
    if (argc > 2) {
            printf("\nToo many arguments.\n");
            return 0;
    }
    else if (argc < 2) {
            printf("\nToo few arguments.\n");
            return 0;
    }
    altitude = strtod(argv[1],&blah);
    test_gtd7(altitude);
    return 0;
}
```

References

[1] Picone, J.M., Drob, D.P., Meier, R.R., Hedin, A.E. NRLMSISE-00 – A New Empirical Model of the Atmosphere. 4 Mar. 2004. United States Navy Website. 15 Mar. 2005 < http://www.nrl.navy.mil/content.php?P=03REVIEW105>.

38.1.1.1.1 Mars Atmosphere Model, "atmosphere2.m"

Author: John Stalbaum

Contributor: Jackie Jaron

This is a model of Mars' atmosphere; all of my MLV codes draw from this model. The functions which produce this model were acquired from NASA Glen Research Center's website. This data was acquired from the Viking lander. [1]

```
% This function calculates atmospheric properties based upon a given
% height, of Mars.
% Written by John Stalbaum

function [p, T, r, a, mu] = marsatmosphere2(h)
% Convert altitude to meters.
h = h/0.3048;

% Calculate various atmospheric properties in English units.
if h < 22960 % Below a certain altitude
    T = -25.68 - .000548 * h;
    p = 14.62 * exp(-.00003 * h);
    r = p / [1149 * (T + 459.7)];
elseif h >= 22960 % Upper atmosphere
    T = -10.34 - .001217 * h;
    p = 14.62 * exp(-.00003 * h);
    r = p / [1149 * (T + 459.7)];
end


R = 192; % Metric R of Martian atmosphere
% Convert back to metric units.
r = 515.378818*r;
T = (460 + T)*0.5555;
p = p * 47.880259;
mu = real((.03170*T^(1.5))*(734.7/(T + 216))*10^-10);  %mu at altitude
gamma = 1.288;
a = sqrt(gamma*R*T);

% If something is fishy, assign these values as zero, representing no
% atmosphere present.
if h > 104e3/0.3048
    r = 0;
    T = 0;
    p = 0;
    mu = 0;
    a = 0;
end


% End of Function
```

References

[1] Benson, Tom. Mars Atmosphere Model – English Units. 4 Mar. 2004. NASA Glen Research Center. 25 Jan. 2005 <http://www.grc.nasa.gov/WWW/K-12/airplane/atmosmre.html>.

38.2 MLV Ascent Code

By: John Stalbaum

Contributors: Rob Anderson, Prof. Williams

This section gives the code that performed the simulation of MLV descent, described in the chapter 2 section. The steering method was obtained from [1].

Code 5 – MLV Ascent Simulation Code (“ascent.m”)

```
% This function numerically integrates the ascent trajectory of the lander.

% Items of particular interest that can be extracted from this code (set
% forth by sense, and the thermal group):
% 1) Time
% 2) Velocity (m/s)
% 3) Mach number
% 4) Atmospheric density (kg/m^3)
% 5) Atmospheric viscosity
% 6) Atmospheric temperature
% 7) Gamma (I don't know what it's called but it's 1.4 for air or something -
% if it's possible to get it)
% 8) Mass

% By John Stalbaum and Jackie Jaron

%function [t, v, M, rho, mu, T, m, dV, p, h,F] = mlvdescent(m0,t0)
clear all; close all; clc;
% Inputs
Thrust = 2.2e5;
htarget = 120e3;
turnalt = 0;
turnfactor = 1.5;
m0 = 2.2528e+004;
%m0 = 23000;
t0 = 0;

% SECTION WHICH DEFINES TRANSFER ORBIT

rmkm = 3397;
rp = rmkm + htarget/10^3; % Periapsis radius of transfer orbit, km
ra = 36416.3e3; % Apoapsis radius of transfer orbit, km
GM = 4.282828e4; % Gravitational constant of Mars, km^3/s^2
ah = (rp + ra)/2; % km
va = sqrt(2*GM/ra-GM/ah); % in km/s
vp = sqrt(2*GM/rp-GM/ah); % in km/s
deltaVrend = abs(va-0.4684)*10^3; % in m/s
vtarget = sqrt(GM/(rmkm+htarget/10^3))*10^3;

% Note to Jackie: Change this to affect the velocity that this code ends
% in.

% Constants:
rmars = 3397e3; % Radius of mars, meters.
v0 = 0;
gam0 = pi/2;       % defined horizantal
h0 = 0;
dt = 1; % Timestep
vsurface = 2*pi*rmars/(24.6597*60^2); % The relative velocity of Mars' surface to lander.
Cd = 0.8;
gmars = 3.72;
g = 9.8;
Isp = 465;
A = pi*4^2;
```

```
% A few words on frames: the default frame is fixed in inertially

% Initial conditions of integration.
i = 1;
v(1) = v0; % Set the velocity initial condition.
t(1) = t0;
m(1) = m0; % This function estimates the initial mass.
gam(1) = gam0; % The initial, inputted inclination angle.
h(1) = h0; % The initial altitude is 350 km above mars.
p(1) = 0; % Define the initial range as zero - arbitrary.
[P(1),T(1),rho(1) a(1) mu(1)] = marsatmosphere2(h(1));
vx(1) = 0;
vy(1) = 0;
v(1) = 0;
M(1) = 0;
LoD(1) = 0;
r = rmars;
iter = 1;
gam(1) = pi/2;
r = rmars;
alph = 0;
vsurface = 2*pi*rmars/(24.6597*60^2); % The relative velocity of Mars' surface to lander.
begin = 1;
tf(1) = 600;

%while t(i) <= 257
% This loop gets the script to the correct altitude.
%while v(i) < vtarget - vsurface
    %%%%%%%%%%%%%%%%%%%%%%%%%%%% STEER METHOD 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while tf(i) > 15
    if h(i) >= turnalt
        if begin == 1
             [gam(i), tf(i), ind] = steer(h(i),vx(i),vy(i),htarget,m(i),Thrust,65);
             begin = 0;
        elseif abs(h(i) - htarget) > 50
             [gam(i), tf(i), ind] = steer(h(i),vx(i),vy(i),htarget,m(i),Thrust,ind);
        else
             gam(i) = gam(i-1);
             tf(i) = tf(i-1);
        end
    else
        gam(i) = pi/2;
    end
    if i / 20 == round(i / 20) && h(i) > turnalt
        disp(['Time is ' num2str(t(i)) ' seconds. Time remaining is allegedly ' num2str(tf(i)) '
seconds.']);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    F(i) = Thrust;
    [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(h(i));
    q(i) = 1/2*rho(i)*v(i)^2; % Using mars
    if a(i) ~= 0
        M(i) = v(i)/a(i);
    else
        M(i) = 0;
    end
    dmdt = -F(i)/(Isp*g);
    D(i) = Cd*q(i)*A;
    Dy(i) = -sign(vy(i)*sin(gam(i)))*sin(gam(i))*D(i);
    Dx(i) = -sign(vx(i)*cos(gam(i)))*cos(gam(i))*D(i);
    LoD(i) = 0;
    L(i) = LoD(i)*D(i);
    Ly(i) = L(i)*sin(gam(i)-pi/2);
    Lx(i) = L(i)*cos(gam(i)-pi/2);
    r = h(i) + rmars;
    dvydt(i) = (F(i)*sin(gam(i)) - m(i)*GM/(r/1000)^2*1000 +Dy(i)+Ly(i))/m(i);
    dvxdt(i) = (Dx(i) + F(i)*cos(gam(i))+Lx(i))/m(i);
    vy(i+1) = vy(i) + dvydt(i)*dt;
    vx(i+1) = vx(i) + dvxdt(i)*dt;
    h(i+1) = h(i) + vy(i)*dt;
    p(i+1) = p(i) + vx(i)*dt;
    t(i+1) = t(i) + dt;
    m(i+1) = m(i) + dmdt*dt;
    v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
    LoD(i+1) = 0;
    M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.

    %%%%%%%%%%%%%%%%%%%%%%%%%% STEER METHOD 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      if h(i) > turnalt
%          performance = ((htarget-h(i))/htarget)^turnfactor;
%          gam(i+1) = performance*pi/2;
%      else
%          gam(i+1) = pi/2;
%      end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if vx(i) ~= 0
        blargh(i) = atan(vy(i)/vx(i));
    else
        blargh(i) = pi/2;
    end

    % delta-V components.
    dVprop(i) = F(i)/m(i);
    dVgrav(i) = -GM/(r/1000)^2*1000*sin(gam(i));
    dVdrag(i) = -D(i)/m(i);
    dVsteer(i)= -F(i)/m(i)*(1-cos(gam(i)-blargh(i)));
    i = i + 1;
    tf(i) = tf(i-1);
end
% Sum delta-V components.
dVprop = sum(dVprop)*dt;
dVgrav = sum(dVgrav)*dt;
dVdrag = sum(dVdrag)*dt;
dVsteer = sum(dVsteer)*dt;
dV = v(i);
plot(p,h);

tfinal = tf(i) + t(i);
disp(['Final time is ' num2str(tfinal) ' seconds.']);
dVp = Isp*g*log(m(1)-Thrust/(Isp*g)*tfinal)/log(m(1));
disp(['Propulsive delta-V is ' num2str(dVp) ' m/s.']);
```

Code 6 – Steering Rule for MLV Ascent ("steer.m")

```
% This script was modified by John Stalbaum to act as a steering law for a
% launch to a circular orbit on mars.
function [alfsteer, tf, successind] = steer(y0,Vx0,Vy0,yf,m,F,indguess)
warning off MATLAB:ode45:IntegrationTolNotMet;
warning off MATLAB:singularMatrix;
warning off MATLAB:nearlySingularMatrixUMFPACK;
% Adjust inputs so that if initial gamma is close to the actual target
% value, it never falls on the wrong side of it.
%echo on

%    Bells trajectory optimization
%
%   Find a thrust vector schedual that will get
%    you into orbit in minimum time.
%
%   This is identical to run13 except the initial
%    guess is done by propagation from assumed ICs
%    The propagation is done with ode45.
%
%    state equations: (for x,y,Vx,Vy)
%   x' = Vx
%   y' = Vy
%   Vx'= Fm*cos(alf)
%   Vy'= Fm*sin(alf) - g
%
%   with x,y,Vx,Vy given at t=0
%        y,Vx Vy given at end,t=tf
%
%        tf is free (but is to be minimized)
%        alf(t) is thrust direction... free
%
%   Fm=thrust/mass
%   g = given constant
%
%    co-state equations : (bet2,alf)
%     (from Euler-Langrange optimization)
%     definitions:  Q=bet2*sin(alf)
%
%     bet2'= bet2*Q
%     alf' =-bet2*cos(alf)
%
%
%  State vector for odebvp:
%   Y=[x,y,Vx,Vy,bet2,alf];
%  "eigenvalue" for odebvp
%   tf=unknown final time
%
%  See also BVP13 ODEBVP

Ntau=10;
tau=linspace(0,1,Ntau);

% set parameters
rmars = 3397e3; % m
mu = 4.2828e4*1000^3; % m^3/s^2
g=mu/(y0+rmars)^2;
f = F/m;
vsurface = 2*pi*rmars/(24.6597*60^2); % The relative velocity of Mars' surface to lander.
Vxf = sqrt(mu/(rmars+yf))-vsurface;

paras={f,g};

% Initial Conditions
 x0=0;
% y0=0;
% Vx0=0;
% Vy0=0;
U1=[x0;y0;Vx0;Vy0];

% Final Conditions:
% yf=alt;
% Vxf=vc;
 Vyf=0;
U2=[yf;Vxf;Vyf];

% initial guess: (by propagation from assumed values
%  of tf,bet1,bet2,alf at tau=1
% tf=406;
bet20=.002;
% alf0=1.23;

alf = linspace(0,pi/2,100);
tfvect = linspace(0,1000,length(alf));
alfsteer = 2*pi;
tf = NaN;
if indguess ~= 0
    ind = indguess;
else
    ind = 1;
end
successind = 0;
while ind <= length(alf) && (~isfinite(tf) || abs(alfsteer) > pi || ~isfinite(alfsteer))
    U0=[U1',bet20,alf(ind)];
    [Y,tf,conv] = odebvp('bvp_steer',tau,U0,tfvect(ind),4,[1,2],U1,U2,paras);
    alfsteer = Y(1,6);
    if ~(ind <= 9 && (~isfinite(tf) || abs(alfsteer) > pi || ~isfinite(alfsteer)))
        successind = ind;
    end
    if ind == indguess
        indguess = 0;
        ind = 1;
    else
        ind = ind + 1;
    end
end
ind = ind - 1;

%echo off
```

Code 7 – Preliminary Method for MLV Descent Simulation ("mlvdescent.m")

```
% This code numerically integrates the equations of motion of the lander to
% propagate the trajectory.
% Various properties must be fiddled with in each new case of mass which is
% applied to the problem.  Be forwarned.
% You must change the damping ratio so that the lift-to-drag ratio and
% G-loading don't become too extreme during the trajectory.  The initial
% angle that you enter at (defined from the horizontal) also affects this.
% The ways in which these affect the trajectory are kind of unpredictable.
% FOR THIS STAGE: the parachute mass and aeroshell mass must be manually
% staged away, so if these change, they must be edited in this code.

% By John Stalbaum and Jackie Jaron

% Items of particular interest that can be extracted from this code (set
% forth by sense, and the thermal group):
% 1) Time
% 2) Velocity (m/s)
% 3) Mach number
% 4) Atmospheric density (kg/m^3)
% 5) Atmospheric viscosity
% 6) Atmospheric temperature
% 7) Gamma (I don't know what it's called but it's 1.4 for air or something -
% if it's possible to get it)
% 8) Mass
% 9) Delta-V estimates.
% 10.) Angle of attack.
% Brief answer to gamma: 1.3ish. This will vary with flow conditions in the
% Hypersonic range, but that's tough to actually get.

% Future plans for this code: Implement a lower G-force generating,
% staged paracute scheme.

% Establish outputs in line with requirements:
function [t, vprime, M, rho, mu, T, m, dV, pprime, hprime,F] = mlvdescent(gam0,m0,damping)
% clear all; close all; clc;
% m0 = 3.0253e+004;
% gam0 = 7.3*pi/180;       % defined horizontal
% damping = 0.069;


% Function inputs
h0 = 100e3;
Isp = 465;
A = pi*4^2; % Reference area
% Chute characteristics.
Cdchute = 0.7;
Achute = pi*21.4^2;
htarget1 = 12e3; % The target for the aeroshell phase to end
htarget2 = 700; % The taget for the parachute phase to end.

% Based on the initial entry angle and interface altitude, this script
% calculates the initial velocity and delta-V for de-orbit.  Written by
% George Pollock.
```

```
[delV_deorb, v_entry] = entry_interface(-gam0*180/pi, h0/10^3); % Code requires degrees, km input
delV_deorb = 10^3*delV_deorb; % Returns delta-V and v_entry in km/s; must get in m/s.
v0 = 10^3*v_entry;

% Constants
dt = 0.1; % Timestep
rmars = 3397e3; % Radius of mars, meters.
vsurface = 2*pi*rmars/(24.6597*60^2); % The relative velocity of Mars' surface to lander.
Cd = 0.8; % Aeroshell coefficient of drag.
gmars = 3.72; % Martian gravitational acceleration.
g = 9.8; % Earth's g, for specific impulse calculations.

% A few words on frames: the default frame is fixed in inertially
% The prime frame is fixed on the mars surface.

% Define initial conditions
i = 1;
v(1) = v0;
t(1) = 0;
m(1) = m0; % This function estimates the initial mass.
gam(1) = pi-gam0; % The initial, inputted inclination angle.
h(1) = h0;
p(1) = 0; % Define the initial range as zero - arbitrary.
[P(1),T(1),rho(1) a(1) mu(1)] = marsatmosphere2(h(1));
vx(1) = -v(1)*cos(gam(1));
vy(1) = -v(1)*sin(gam(1));
v(1) = sqrt(vx(1)^2 + vy(1)^2);
if a(1) > 0
    M(1) = v(1)/a(1);
else
    M(1) = 0;
end
hprime(1) = h(1);
vxprime(1) = vx(1) - vsurface;
vyprime(1) = vy(1);
vprime(1) = sqrt(vxprime(1)^2+vyprime(1)^2);
pprime(1) = 0;
LoD(1) = damping*(hprime(1)-htarget1)/(hprime(1)-htarget1);
gam(1) = pi+atan(vyprime(1)/vxprime(1));

% Main functional loops
q(i) = 700;
% Aerodescent portion:
exit = 0;
%while exit == 0 && hprime(i) > 0
while M(i) > 2 && hprime(i) > 0
    F(i) = 0;
    [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
    q(i) = 1/2*rho(i)*vprime(i)^2;
    if a(i) ~= 0
        M(i) = vprime(i)/a(i);
    else
        M(i) = 0;
    end
    dmdt = -F(i)/(Isp*g);
    D(i) = Cd*q(i)*A;
    Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
    Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
    L(i) = LoD(i)*D(i);
    Ly(i) = L(i)*sin(gam(i)-pi/2);
    Lx(i) = L(i)*cos(gam(i)-pi/2);
    dvydt(i) = (F(i)*sin(gam(i)) - m(i)*gmars +Dy(i)+Ly(i))/m(i);
    dvxdt(i) = (Dx(i) + F(i)*cos(gam(i))+Lx(i))/m(i);
    r = rmars + hprime(i);
    vy(i+1) = vy(i) + dvydt(i)*dt;
    vx(i+1) = vx(i) + dvxdt(i)*dt;
    h(i+1) = h(i) + vy(i)*dt;
    p(i+1) = p(i) + vx(i)*dt;
    t(i+1) = t(i) + dt;
    m(i+1) = m(i) + dmdt*dt;
    v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
```

```
          hprime(i+1) = sqrt((rmars+h(i+1))^2+p(i+1)^2)-rmars;
          vxprime(i+1) = vx(i+1) - vsurface; % Velocity with respect to the surface
          vyprime(i+1) = (hprime(i+1)-hprime(i))/dt;
          vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
          pprime(i+1) = pprime(i) + vxprime(i)*dt;
          gam(i+1) = pi+atan(vyprime(i+1)/vxprime(i+1));
          LoD(i+1) = damping*(hprime(1)-htarget1)/(hprime(i+1)-htarget1);
          if LoD(i+1) > 2 || LoD(i+1) < 0
              LoD(i+1) = 2;
          end
          M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
          if (q(i) < 600 && hprime(i) < 80e3)
             exit = 1;
          end
          i = i + 1;
     end
     iparachute = i;
     m(i) = m(i) - 1000;

     %blargh = input('Pause!');
     % Parachute portion
     while hprime(i) > htarget2
         F(i) = 0;
         [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
         q(i) = 1/2*rho(i)*vprime(i)^2; % Using mars
         if a(i) ~= 0
             M(i) = vprime(i)/a(i);
         else
             M(i) = 0;
         end
         dmdt = -F(i)/(Isp*g);
         D(i) = q(i)*(Cd*A+Cdchute*Achute);
         Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
         Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
         L(i) = LoD(i)*D(i);
         Ly(i) = L(i)*sin(gam(i)-pi/2);
         Lx(i) = L(i)*cos(gam(i)-pi/2);
         dvydt(i) = (F(i)*sin(gam(i)) - m(i)*gmars +Dy(i)+Ly(i))/m(i);
         dvxdt(i) = (Dx(i) + F(i)*cos(gam(i))+Lx(i))/m(i);
         r = rmars + hprime(i);
         vy(i+1) = vy(i) + dvydt(i)*dt;
         vx(i+1) = vx(i) + dvxdt(i)*dt;
         h(i+1) = h(i) + vy(i)*dt;
         p(i+1) = p(i) + vx(i)*dt;
         t(i+1) = t(i) + dt;
         m(i+1) = m(i) + dmdt*dt;
         v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
          hprime(i+1) = sqrt((rmars+h(i+1))^2+p(i+1)^2)-rmars;
          vxprime(i+1) = vx(i+1) - vsurface; % Velocity with respect to the surface
          vyprime(i+1) = (hprime(i+1)-hprime(i))/dt;
          vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
          pprime(i+1) = pprime(i) + vxprime(i)*dt;
          gam(i+1) = pi+atan(vyprime(i+1)/vxprime(i+1));
          LoD(i+1) = 0;
          M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
          i = i + 1;
     end
     m(i) = m(i) - 1400;

     % Third portion - forced deceleration to 0 m/s at 100 m
     % The acceleration is enough to arrest all velocity at the particular 100 m
     % target.
     % The situation is simplified, at this point.  The normal frame is
     % dropped.  The vehicle is assumed to be close enough to the surface that
     % it can be approximated as flat.
     %
     vx(i) = vxprime(i);
     vy(i) = vyprime(i);
     ithrust = i;

     while hprime(i) > 100 && i < ithrust+1000
```

```
        [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
        q(i) = 1/2*rho(i)*vprime(i)^2; % Using mars
        if a(i) ~= 0
            M(i) = vprime(i)/a(i);
        else
            M(i) = 0;
        end
        D(i) = q(i)*(Cd*A+Cdchute*Achute);
        Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
        Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
        L(i) = LoD(i)*D(i);
        Ly(i) = L(i)*sin(gam(i)-pi/2);
        Lx(i) = L(i)*cos(gam(i)-pi/2);
        % Acceleration that would hit change the velocity in the distance
        % interval (from kinematics).
        if abs(hprime(i)-100) > 10
            aydesired = -sign(vyprime(i))*(vyprime(i))^2/(2*(hprime(i)-100));
            timedecel = abs(vyprime(i))/aydesired;
            axdesired = -vxprime(i)/timedecel;
        else
            axdesired = 0;
            aydesired = 0;
        end
        Fy(i) = (m(i)*aydesired+m(i)*gmars-Dy(i)-Ly(i));
        Fx(i) = (m(i)*axdesired-Dx(i)-Lx(i));
        F(i) = sqrt(Fx(i)^2+Fy(i)^2);
        dmdt = -F(i)/(Isp*g);
        dvydt(i) = (Fy(i) - m(i)*gmars +Dy(i)+Ly(i))/m(i); % F(i)*sin(gam(i))
        dvxdt(i) = (Dx(i) + Fx(i)+Lx(i))/m(i); % *cos(gam(i))
        r = rmars + hprime(i);
        vy(i+1) = vy(i) + dvydt(i)*dt;
        vx(i+1) = vx(i) + dvxdt(i)*dt;
        h(i) = hprime(i);
        hprime(i+1) = hprime(i) + vy(i)*dt;
        p(i+1) = p(i) + vx(i)*dt;
        t(i+1) = t(i) + dt;
        m(i+1) = m(i) + dmdt*dt;
        v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
         vxprime(i+1) = vx(i+1); % Velocity with respect to the surface
         vyprime(i+1) = vy(i+1);
         vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
         pprime(i+1) = pprime(i) + vxprime(i)*dt;
         gam(i+1) = pi+atan(vyprime(i+1)/vxprime(i+1));
         LoD(i+1) = 0;
         M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
         dV(i) = F(i)/m(i);
         i = i + 1;
    end
    idescend = i;
    vx(i) = 0;
    vy(i) = -5/3;
    hprime(i) = 100;

    % The final descent, with 100 meters of cross-range capability built in.
    % The time interval.
    for i = idescend : idescend + 60/dt
        % Atmospheric script
        [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
        q(i) = 1/2*rho(i)*vprime(i)^2; % Using mars
        % Mach number, protected from division by zero.
        if a(i) ~= 0
            M(i) = vprime(i)/a(i);
        else
            M(i) = 0;
        end
        % Various aerodynamic forces defined.
        D(i) = q(i)*(Cd*A);
        Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
        Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
        L(i) = LoD(i)*D(i);
        Ly(i) = L(i)*sin(gam(i)-pi/2);
        Lx(i) = L(i)*cos(gam(i)-pi/2);
        vy(i+1) = -5/3; % Decelerates to 0 m/s in 60 seconds.
        if i < idescend + 30/dt; % If it's not done accelerating,
            dvxdt(i+1) = -2*100/30^2; % Accelerate.
            gam(i+1) = pi/2-abs(atan(vx(i)/vy(i)));
        else % If not,
            dvxdt(i+1) = 2*100/30^2; % Decelerate.
            gam(i+1) = pi/2+abs(atan(vx(i)/vy(i)));
        end
        dvydt(i+1) = 0; % Descending at a constant rate of speed.
        vx(i+1) = vx(i) + dvxdt(i)*dt; % Update for cross-range.
        % Acceleration that would hit change the velocity in the distance
        % interval (from kinematics).
        Fy(i) = (m(i)*gmars-Dy(i)-Ly(i)); % Obtain the force components
        Fx(i) = (m(i)*dvxdt(i)-Dx(i)-Lx(i)); % required for acceleration.
        F(i) = sqrt(Fx(i)^2+Fy(i)^2); % Magnitude of force.
        dmdt = -F(i)/(Isp*g); % Flow rate, based on force magnitude.
        % Various updating of variables, and updating of things which aren't
        % used in this final trajectory which approxiates for reference frames.
        r = rmars + hprime(i);
        h(i) = hprime(i);
        hprime(i+1) = hprime(i) + vy(i)*dt;
        p(i+1) = p(i) + vx(i)*dt;
        t(i+1) = t(i) + dt;
        m(i+1) = m(i) + dmdt*dt;
        v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
         vxprime(i+1) = vx(i+1); % Velocity with respect to the surface
         vyprime(i+1) = vy(i+1);
         vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
         pprime(i+1) = pprime(i) + vxprime(i)*dt;
         gam(i+1) = pi/2;
         LoD(i+1) = 0;
         M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
         dV(i) = F(i)/m(i);
    end

    % Not yet implemented.
    for i = 1 : length(F)
        if F(i) > 1e5
            F(i) = 1e5;
        end
    end
    dV = sum(dV)*dt;

    % Finally, produce a set of plots that are characteristic of the motion of
    % the craft. (For version of code which is not a function).
    % plot(t,LoD);
    % xlabel('Time (s)');
    % ylabel('Lift-to-Drag Ratio (unitless)');
    % title('Lift-to-drag Ratio of the Lander Entry Trajectory');
    % dvdt = sqrt(dvydt.^2+dvxdt.^2);
    % Gs = dvdt/9.8;
    % figure(2);
    % blargh = plot(t(1:length(vprime)),vprime);
    % set(blargh,'linewidth',2);
    % xlabel('Time (s)');
    % ylabel('Velocity (m/s)');
    % title('Velocity Profile of the Lander');
    % figure(3);
    % blargh = plot(t(1:length(dvdt)),dvdt/9.8);
    % set(blargh,'linewidth',2);
    % xlabel('Time (s)');
    % ylabel('Acceleration (Gs)');
    % title('G-Loading of the Lander');
    %
    % figure(4);
    % blargh = plot(pprime,hprime);
    % set(blargh,'linewidth',2);
    % xlabel('Range (m)');
    % ylabel('Altitude (m)');
    % title('Flight Path of the Lander');
```

```
%
% figure(5);
% blargh = plot(t(1:length(rho)),rho);
% set(blargh,'linewidth',2);
% xlabel('Time (s)');
% ylabel('Density (kg/m^3)');
% title('Density Profile of Trajectory');


% End of Script
```

Code 8 – Lander Initial Mass Estimation Script

```
% AAE 450 lander weight estimation script
% by John Stalbaum
% This script estimates the initial lander (MLV, Ares) mass based upon the
% final mass, delta-V for the orbital injection, and delta-V for the
% transfer into the rendezvous orbit.  It does this by doing a 3-stage
% rocket problem, considering the first stage as the descent stage, the
% second stage as the first ascent stage, and the third stage as the second
% ascent stage.
% Inputs:
% mfinal = Lander payload mass (final mass in that will be lifted into the
% rendezvous orbit
% deltaVrend = velocity increment from 350 km circular orbit to the orbit
% in which the lander and CTV rendezvous.
% deltaVinj = velocity increment from the parking orbit to orbital
% injection.


function mtotal = m0lander(mfinal,deltaVrend,deltaVinj)

% Constants
g = 9.81;
% The code is done vectorially; all quantities are defined for a stage in
% the respective element of each vector.

% delta-V assignments  for various stages.  The descent stage delta-V comes
% from initial estimates of each stage, the number "3731" comes from a
% trade study done by Jackie Jaron (presentation listed on the website),
% and deltaVrend and deltaVinjection are inputs.  The two ascent stages
% have the delta-V achieved by each stage divided between them equally
% (which isn't necessarily optimum, but can be improved later).
deltav = [(1000+deltaVinj) ((3731+deltaVrend)/2) ((3731 + deltaVrend)/2)];

% Assuming the optimistic specific impulse given by Rob Anderson for an
% NTO/MMH propulsion system.
Isp = [460 460 460];

% Same with mass fractions.
lamb = [0.9 0.9 0.9];


k = exp(deltav./(Isp*g)); % m0/mf
% Each stage has three equations giving information on it:
% m0/mf = (minert + mpayload + mpropellant)/(minert + mpayload) (1)
% lambda = mpropellant/(mpropellant + minert)                   (2)
% mpayload = m(previous stages mass)                            (3)
% These two equations can be rearranged to form the linear system:
% A*x = b, implying the augmented matrix [A b]
% The vector x is defined as:
% x = [mpayload_1, mpayload_2, mpayload_3, minert_1, minert_2,
% minert_3,mpropellant_1, mpropellant_2, mpropellant_3]

% The matrix A, mentioned above.
A = [0 0 1 0 0 0 0 0 0; % A matrix for linear system defined in
    0 -1 1 0 0 1 0 0 1; % handwritten work
    -1 1 0 0 1 0 0 1 0;
    0 0 0 lamb(1) 0 0 lamb(1)-1 0 0
    0 0 0 0 lamb(2) 0 0 lamb(2)-1 0
    0 0 0 0 0 lamb(3) 0 0 lamb(3)-1
    1-k(1) 0 0 1-k(1) 0 0 1 0 0
    0 1-k(2) 0 0 1-k(2) 0 0 1 0
    0 0 1-k(3) 0 0 1-k(3) 0 0 1];

% The matrix b, mentioned above.
B = [mfinal; 0; 0; 0; 0; 0; 0; 0; 0];

% The solution to the linear system is given by the right column of the
% reducted row-eschelon form of the augmented matrix.
soln = rref([A B]);

% At this point, all of the desired variables are recovered from the
% system.
soln = soln(:,10);
mp(1) = soln(1);
mp(2) = soln(2);
mp(3) = soln(3);
mi(1) = soln(4);
mi(2) = soln(5);
mi(3) = soln(6);
mf(1) = soln(7);
mf(2) = soln(8);
mf(3) = soln(9);

% Finally, the total vehicle mass is going to be the sum of the inert,
% payload, and propellant mass of the first stage (since all other stages
% are included in the payload mass of this vehicle).
mtotal = mf(1) + mi(1) + mp(1)

% End of Script
```

References

[1] Longuski, James. AAE 508 Notes. West Lafayette, IN: Copymat, 2004.

38.3 Descent Code

By: John Stalbaum

Contributors: Rob Anderson, Jackie Jaron, Angela Long, Hafid Long, Mark Vahle

This is the code used to simulate the MLV descent. The first code, “descent.m,” is described in its chapter 2 sections on MLV aerodynamic and powered descent. The source of these equations of motion is given by reference [1]. Some code is included which calculates values necessary for sizing the heat shield, also. The second code, “eom.m,” is simply the equation of motion file the “ascent.m” uses. The third code, “mlvdescent.m,” is the code that was used in a more preliminary analysis of the descent. It yields similar results to the code which is formally used; a comparison between the two codes under identical initial conditions and vehicle characteristics is given below (The new method is designated as “Angela” because Angela Long introduced the usage of the new equations of motion in her Earth entry simulation):

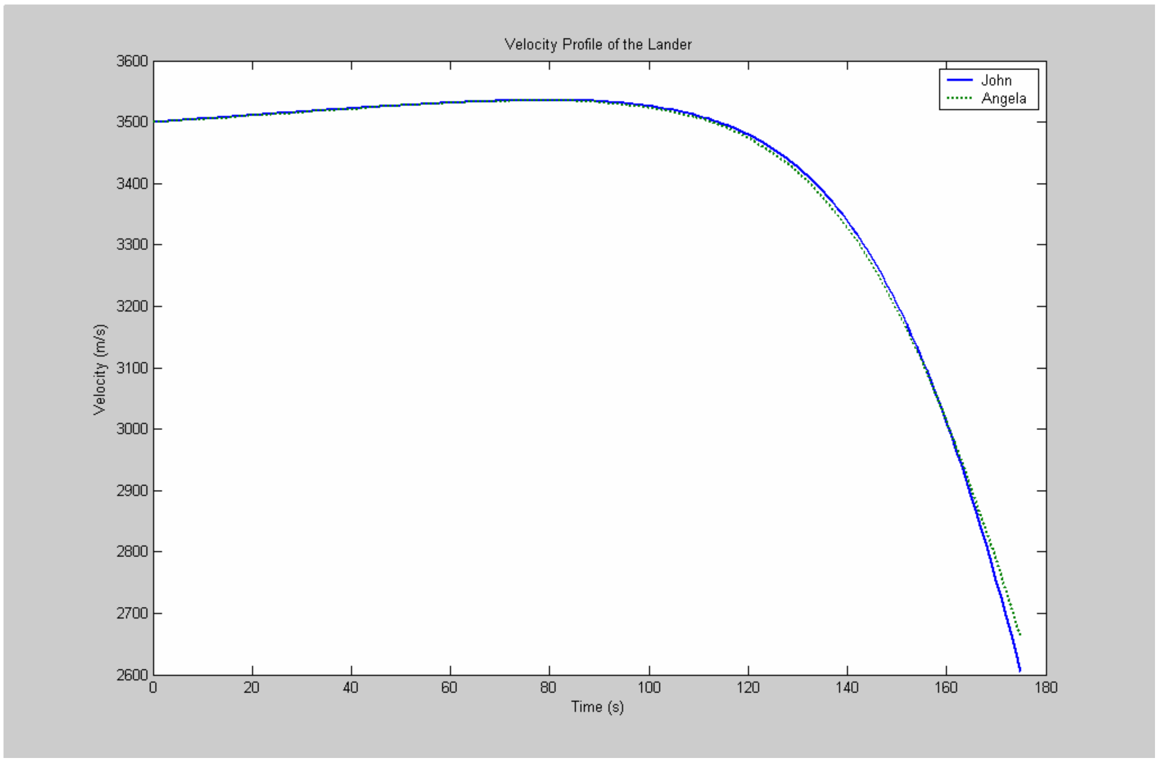


Figure 6 – Comparison of Velocity Profile for Old and New Trajectory Propagation Methods

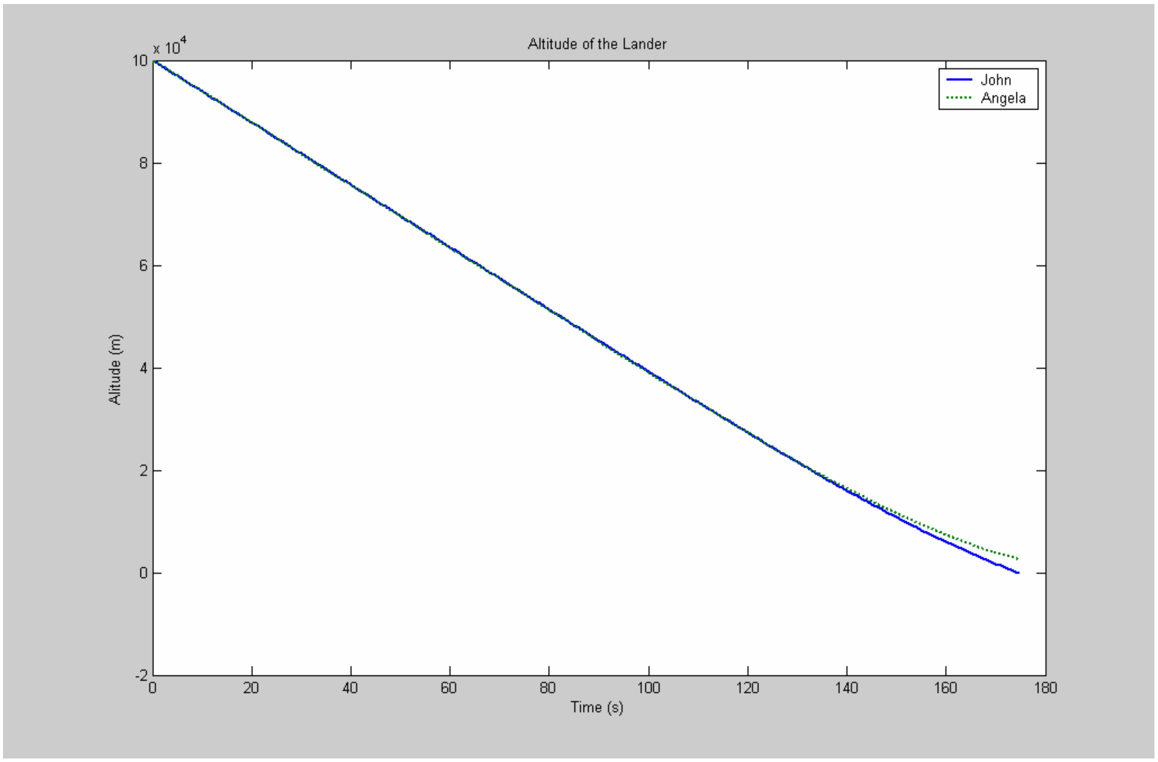


Figure 7 – Comparison of Altitude vs. Time for Old and New Trajectory Propagation Methods

The old code was run until the MLV hit the surface, and the current code was ran for that time interval. Comparing the results, final velocity is 1.2 percent off and final altitude is 3 percent off. An analysis independent of the current one verified the results that we obtain, so this lends credibility to our analysis.

The fourth code, "m0lander.m," estimates the initial lander mass by using estimated delta-Vs. It then feeds this into the old trajectory propagation code (developed before these initial masses had been arrived at), "mlvdescent.m."

Code 9 – MLV Descent Simulation Code ("descent.m")

```
% AAE 450 Mars atmospheric descent model.
% By John Stalbaum
% Thermal code adapted from Pascal code written by Prof. Schneider by Hafid
% Long and John Stalbaum

clear all; close all; clc;
warning off MATLAB:divideByZero;
% Globals, for passage into EOM.
global Cd LoD S slast h0 damping htarget mdot ncp l rnose Cpnose rhonose Cpw qr rhow ...
    sig emiss Pr Cpe sarc Thwall cqwnose rmars GM;
slast = 1;
```

```
%%%%%%%%%%%%%%%%%% CONTROL PANEL - CHANGE THINGS HERE %%%%%%%%%%%%%%%%%%%%%%
damping = 1/100;
htarget = 30e3;
gam0 = -8*pi/180;

%%%%%%%%%%%%%%%%%%%%%%%%%%% MASS PROPERTIES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m0 = 33000; % initial mass, kg

%%%%%%%%%%%%%%%%%%%%%%%%% AERODESCENT STAGE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Cdaero = 0.8;
Saero = pi*5^2;
LoD_aero = 2;
t_aero_end = 995;
m_hs = 2001.95+3395.1;
m_inert_hs = 2001.95;

%%%%%%%%%%%%%%%%%%%%%%%% PARACHUTE STAGE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Cdchute = 1.2;
Schute = pi*21.4^2;
LoD_chute = 0;
t_chute_end = 100;

%%%%%%%%%%%%%%%%%%%%%%% FORCED DESCENT STAGE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Cdforced = 0.8;
Sforced = pi*4^2;
LoD_forced = 0;
Isp = 465;
dt = 0.1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




% Code portion - Shouldn't be changed.
mdot_hs = -(m_hs-m_inert_hs)/t_aero_end; % Heat shield ablation rate.

ncp = 10; % Number of control points.

% Define constants:
rmars = 3397e3; % km
GM = 4.2828e4*1000^3; % m^3/s^2
gearth = 9.8; % m/s^2 (for Isp calculations)

%%%%%%%%%%%%%%%%%%%%%%%% HEAT SHIELD SIZING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
l = 11.02;     % length, m
rnose = 0.305;  % radius of nose, meters (CHANGE THIS LATER, PERHAPS?)

% Heat shield material properties
%  Nose (UHTC)
%Cpnose = 1172.3;           % J/(kg*K)
%rhonose = 480;             % kg/m^3;
Cpnose = 750;   % J/(kg*K)
rhonose = 3400; % kg/m^3

%  Remainder (SLA-561V)
Cpw = 1172.3;           % J/(kg*K)
qr = 0;
rhow = 480;             % kg/m^3;
%thickw = 3;             % cm
%gw = rhow*Cpw*thickw;  % Total wall heat capacity per cm^2
sig = 5.669e-8;         % W/(m^2*K^4)
emiss = 0.9;
Pr = 0.72;
Cpe = 1100;             % Specific heat for air, J/(kg*K)

% Setup for state variable equations for heating:
for i = 2:ncp
```

```
        sarc(i) = ((i-1)/ncp)^2*l;
        if sarc(i) < 1/4
            Thwall(i) = 0.05;
        else
            Thwall(i) = 0.05;
        end
    end
    %  Stagnation point heat flux:
    cqwnose = 1e4*1.35e-8;          % Convert to W/m^2

    m_chute = 0.031*(m0-m_hs); % Heat shield is 3.1 percent of total mass.

    %%%%%%%%%%%%%%%%%% INTEGRATION LOOP FOR AERODESCENT %%%%%%%%%%%%%%%%%%%%%%%


    % Define initial conditions:
    h0 = 103e3;
    theta0 = 0;
    phi0 = 0;
    T0 = 201;
    % Code requires degrees, km input
    [delV_deorb, v0] = entry_interface(gam0*180/pi, h0/10^3);
    v0 = v0*10^3;
    psi0 = 0;
    r0 = h0+rmars;

    % Define storage vectors:
    h = h0;
    theta = theta0;
    phi = phi0;
    v = v0;
    gam = gam0;
    psi = psi0;
    t = 0;
    m = m0;
    Tstag = T0;
    T1 = T0;
    T2 = T0;
    T3 = T0;
    T4 = T0;
    T5 = T0;
    T6 = T0;
    T7 = T0;
    T8 = T0;
    T9 = T0;

    % Loop control parameters.
    pars = [Saero, Cdaero, LoD_aero, t_aero_end, mdot_hs, ...
        m_inert_hs; Schute, Cdchute, LoD_chute, t_chute_end, 0, m_chute];
    [ro, co] = size(pars);

    % Main loop - applies ode45 to each parameter set.
    for ind = [1 : ro]
        S = pars(ind,1);
        Cd = pars(ind,2);
        x0 = [h(length(h))+rmars, theta(length(theta)), phi(length(phi)),...
            v(length(v)), gam(length(gam)), psi(length(psi)), m(length(m)),...
            Tstag(length(Tstag)), T1(length(T1)), T2(length(T2)), ...
            T3(length(T3)), T4(length(T4)), T5(length(T5)), T6(length(T6)),...
            T7(length(T7)), T8(length(T8)), T9(length(T9))];
        LoD = pars(ind,3);
        tspan = [t(length(t)) t(length(t))+pars(ind,4)];
        mdot = pars(ind,5);
        % Call ode45 to integrate.
        [tstore, x] = ode45('eom', tspan, x0);
        % Extract states into variables.
        h = [h; x(:,1) - rmars];
        theta = [theta; x(:,2)];
        phi = [phi; x(:,3)];
        v = [v; x(:,4)];
        gam = [gam; x(:,5)];
        psi = [psi; x(:,6)];
        m = [m; x(:,7)];
        Tstag = [Tstag; x(:,8)];
        T1 = [T1; x(:,9)];
        T2 = [T2; x(:,10)];
        T3 = [T3; x(:,11)];
        T4 = [T4; x(:,12)];
        T5 = [T5; x(:,13)];
        T6 = [T6; x(:,14)];
        T7 = [T7; x(:,15)];
        T8 = [T8; x(:,16)];
        T9 = [T9; x(:,17)];
        t = [t; tstore];
        transitionind(ind) = length(t)-1;
        m(length(m)) = m(length(m))-pars(ind,6);
    end

    % Truncate unneeeded vector elements.
    h = h(2:length(h));
    theta = theta(2:length(theta));
    phi = phi(2:length(phi));
    v = v(2:length(v));
    gam = gam(2:length(gam));
    psi = psi(2:length(psi));
    t = t(2:length(t));
    m = m(2:length(m));
    Tstag = Tstag(2:length(Tstag));
    T1 = T1(2:length(T1));
    T2 = T2(2:length(T2));
    T3 = T3(2:length(T3));
    T4 = T4(2:length(T4));
    T5 = T5(2:length(T5));
    T6 = T6(2:length(T6));
    T7 = T7(2:length(T7));
    T8 = T8(2:length(T8));
    T9 = T9(2:length(T9));

    slast = 1;
    % Create atmospheric properties vectors and dynamic pressure.
    for ind = 1 : length(t)
        [P(ind,1),T(ind,1),rho(ind,1) a(ind,1) mu(ind,1)] =...
            marsatmosphere2(h(ind));
        q(ind,1) = 1/2*rho(ind)*v(ind)^2;
        if abs(phi(ind)) > 0.5*pi/180
            if phi(ind) > 0
                s = -1;
            else
                s = 1;
            end
        else
            s = slast;
        end
        slast = s;
        if h(ind) >= htarget && h(ind) <= h0
            sigplot(ind) = real(pi/2*((h(ind)-htarget)/...
                (h0 - htarget))^damping)*s;
        elseif h(ind) < htarget
            sigplot(ind) = 0;
        elseif h(ind) > h0
            sigplot(ind) = pi/2*s;
        end
    end

    %%%%%%%%%%%%%%%%%%%%% CALCULATE HEATING RATES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    states = [h + rmars, theta, phi, v, gam, psi, m, Tstag, T1, T2, T3, T4,...
        T5, T6, T7, T8, T9];
    Vinf = v;
    Twnose = Tstag;
    Cpwnose = Cpnose;
    qwnose = cqwnose*sqrt(rho/rnose).*Vinf.^3.04.*(1-(Cpwnose*Twnose...
        ./(0.5*Vinf.^2)));

    %  Compute point of laminar-turbulent transition:
    Minf = Vinf./a;
    alpha = 10*pi/180;
    Me = Minf*cos(alpha);
    exparg = 1.209e-4*Me.^2.641;
    muinf = 1.716e-5*(T/273).^(1/5).*(273+111)./(T+111);

    for ind = 1 : length(t)
        if exparg(ind) > 1.5
            Rext(ind,1) = 1e30;
        else
            Rext(ind,1) = 10^(6.421*exp(exparg(ind)));
        end
    end

    %  Compute transition Reynolds:
    xtrans = Rext.*muinf./(rho.*Vinf);

    %  Setup state variable for heating at nose:
    x1b = rnose*pi/2;
    g1b = Cpwnose*Twnose./(0.5*Vinf.^2);
    for ind = 1 : length(t)
        if x1b < xtrans(ind)
            cq(ind,1) = 1e4*2.53e-9*cos(alpha)^(1/2)*sin(alpha)*(1-g1b(ind))...
                /sqrt(x1b);
            expN(ind,1) = 0.5;
            expM(ind,1) = 3.2;
        else
            expN(ind,1) = 0.8;
            if Vinf(ind) <= 3962
                cq(ind,1) = 1e4*3.35e-8*cos(alpha)^(1.78)*sin(alpha)^(1.6)...
                    *x1b^(-0.2)*(Twnose(ind)/556)^(-1/4)*(1-1.11*g1b(ind));
                expM(ind,1) = 3.37;
            else
                cq(ind,1) = 1e4*2.2e-9*cos(alpha)^(2.08)*sin(alpha)^(1.6)...
                    *x1b^(-0.2)*(1-1.11*g1b(ind));
                expM(ind,1) = 3.7;
            end
        end
    end

    qw1b = cq.*rho.^expN.*Vinf.^expM;

    qw(:,1) = qwnose/4+qw1b/4-emiss*sig*Twnose.^4;
    AoA = alpha;            % <= *

    %  Setup state variable for heating for remainder of body
    for i = 9:ncp+7
        itpt = i-7;
        for ind = 1 : length(t)
            Tw = states(ind,i);
            xarc = rnose*pi/2+sarc(itpt);
            g = Cpw*Tw/(0.5*Vinf(ind)^2);
            if Minf(ind)*sin(AoA) < 1
                %disp('Warning: Angle is too small for Tauber equation.');
                % Compute reference temperature
                Tref = T(ind)*(0.5+0.039*Minf(ind)^2+0.5*Tw/T(ind));
                rhoref = rho(ind)*(T(ind)/Tref);
                muref = 1.716e-5*(Tref/273)^(1/5)*(273+111)/(Tref+111);
                Rexe = rho(ind)*Vinf(ind)*xarc/muinf(ind);
                cstar = (Tref/T(ind))^(-1/3);
                if xarc < xtrans(ind)
                    cfe = 0.664*sqrt(cstar/Rexe);
                    che = 0.5*cfe/Pr^(2/3);
                    Taw = T(ind)*(1+sqrt(Pr)*0.2*Minf(ind)^2);
                    qwconvect = che*rho(ind)*Vinf(ind)*Cpe*(Taw-Tw);
                else
                    Rexref = rhoref*Vinf(ind)*xarc/muref;
                    cfe = 0.027/Rexref^(1/7);
                    che = 0.5*cfe/Pr^(2/3);
                    Taw = T(ind)*(1+(Pr^(1/3))*0.2*Minf(ind)^2);
                    qwconvect = che*rho(ind)*Vinf(ind)*Cpe*(Taw-Tw);
                end
            else
                if xarc < xtrans(ind)
                    cq(ind,itpt) = 1e4*2.53e-9*cos(AoA)^(1/2)*sin(AoA)*(1-g)...
                        /sqrt(xarc);
                    expN(ind,itpt) = 0.5;
                    expM(ind,itpt) = 3.2;
                else
                    expN(ind,itpt) = 0.8;
                    if Vinf(ind) <= 3962
                        cq(ind,itpt) = 1e4*3.35e-8*cos(AoA)^(1.78)*...
                            sin(AoA)^(1.6)*xarc^(-0.2)*...
                            (Tw/556)^(-1/4)*(1-1.11*g);
                        expM(ind,itpt) = 3.37;
                    else
                        cq(ind,itpt) = 1e4*2.2e-9*cos(AoA)^(2.08)*...
                            sin(AoA)^(1.6)*xarc^(-0.2)*(1-1.11*g);
                        expM(ind,itpt) = 3.7;
                    end
                end
                qwconvect = cq(ind,itpt)*rho(ind)^expN(ind,itpt)*...
                    Vinf(ind)^expM(ind,itpt);
            end
            qw(ind,itpt) = qwconvect-emiss*sig*Tw^4;
        end
    end

    %%%%%%%%%%%%%%%%% FORCED DECELERATION TO 0 M/S AT 100 M %%%%%%%%%%%%%%%%%%

    % At this point, the aerodescent portion is completed.  The equations of
    % motion for aeroentry are scrapped, in leiu of a flat planet assumption.

    ithrust = length(t);
    i = ithrust;
    vx(i) = v(i)*cos(gam(i));
    vy(i) = v(i)*sin(gam(i));
    p(i) = 0;
    while h(i) > 100% && i < ithrust+1000
        [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(h(i));
        q(i) = 1/2*rho(i)*v(i)^2;
        if a(i) ~= 0
            M(i) = v(i)/a(i);
        else
            M(i) = 0;
        end
        D(i) = q(i)*(Cd*Sforced+Cdchute*Schute);
        Dy(i) = -sign(vy(i)*sin(gam(i)))*sin(gam(i))*D(i);
        Dx(i) = -sign(vx(i)*cos(gam(i)))*cos(gam(i))*D(i);
        % Acceleration that would hit change the velocity in the distance
        % interval (from kinematics).
        if abs(h(i)-100) > 10
            aydesired = -sign(vy(i))*(vy(i))^2/(2*(h(i)-100));
            timedecel = abs(vy(i))/aydesired;
            axdesired = -vx(i)/timedecel;
        else
            axdesired = 0;
            aydesired = 0;
        end
        r = rmars + h(i);
        Fy(i) = ( m(i) * aydesired + m(i) * (GM/r^2) - Dy(i) );
        Fx(i) = ( m(i) * axdesired - Dx(i) );
        F(i) = sqrt(Fx(i)^2+Fy(i)^2);
        dmdt = -F(i)/(Isp*gearth);
        dvydt(i) = ( Fy(i) - m(i) * (GM/r^2) + Dy(i) ) / m(i);
        dvxdt(i) = ( Dx(i) + Fx(i) ) / m(i);
        vy(i+1) = vy(i) + dvydt(i)*dt;
        vx(i+1) = vx(i) + dvxdt(i)*dt;
        h(i+1) = h(i) + vy(i)*dt;
```

```
        p(i+1) = p(i) + vx(i)*dt;
        t(i+1) = t(i) + dt;
        m(i+1) = m(i) + dmdt*dt;
        v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
        gam(i+1) = pi+atan(vy(i+1)/vx(i+1));
        theta(i+1) = theta(i) + (p(i+1)-p(i))/(h(i) + rmars);
        M(i+1) = M(i); % This just makes sure that M is defined.
        dV(i) = F(i)/m(i);
        i = i + 1;
    end

    idescend = i;

    %%%%%%%%%%%%% FINAL DESCENT + CROSS-RANGE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for i = idescend : idescend + 60/dt
        % Atmospheric script
        [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(h(i));
        q(i) = 1/2*rho(i)*v(i)^2; % Using mars
        % Mach number, protected from division by zero.
        if a(i) ~= 0
            M(i) = v(i)/a(i);
        else
            M(i) = 0;
        end
        % Various aerodynamic forces defined.
        D(i) = q(i)*(Cd*Sforced);
        Dy(i) = -sign(vy(i)*sin(gam(i)))*sin(gam(i))*D(i);
        Dx(i) = -sign(vx(i)*cos(gam(i)))*cos(gam(i))*D(i);
        vy(i+1) = -5/3; % Decelerates to 0 m/s in 60 seconds.
        if i < idescend + 30/dt; % If it's not done accelerating,
            dvxdt(i+1) = -2*100/30^2; % Accelerate.
            gam(i+1) = pi/2-abs(atan(vx(i)/vy(i)));
        else % If not,
            dvxdt(i+1) = 2*100/30^2; % Decelerate.
            gam(i+1) = pi/2+abs(atan(vx(i)/vy(i)));
        end
        dvydt(i+1) = 0; % Descending at a constant rate of speed.
        vx(i+1) = vx(i) + dvxdt(i)*dt; % Update for cross-range.
        % Acceleration that would hit change the velocity in the distance
        % interval (from kinematics).
        r = rmars + h(i);
        Fy(i) = (m(i)*(GM/r^2)-Dy(i)); % Obtain the force components
        Fx(i) = (m(i)*dvxdt(i)-Dx(i)); % required for acceleration.
        F(i) = sqrt(Fx(i)^2+Fy(i)^2); % Magnitude of force.
        dmdt = -F(i)/(Isp*gearth); % Flow rate, based on force magnitude.
        % Update various variables:
        h(i) = h(i);
        h(i+1) = h(i) + vy(i)*dt;
        p(i+1) = p(i) + vx(i)*dt;
        t(i+1) = t(i) + dt;
        m(i+1) = m(i) + dmdt*dt;
        v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
        LoD(i+1) = 0;
        M(i+1) = M(i); % This just makes sure that M is defined.
        theta(i+1) = theta(i) + (p(i+1)-p(i))/(h(i) + rmars);
        dV(i) = F(i)/m(i);
    end

    %%%%%%%%%%%%%%%%%%%%%% CREATE PLOTS AND FINALIZE %%%%%%%%%%%%%%%%%%%%%%%%%%


    [P(i+1),T(i+1),rho(i+1) a(i+1) mu(i+1)] = marsatmosphere2(h(i+1));
    q(i+1) = 1/2*rho(i+1)*v(i+1)^2; % Make vectors the same length.
    dV = sum(dV)*dt;
    p(1:transitionind(2)) = theta(1:transitionind(2))*rmars;
    p(transitionind(2)+1:length(p)) = p(transitionind(2)) + ...
        p(transitionind(2)+1:length(p));
    thetamars = linspace(0,2*pi);
    %subplot(2,1,1);
    approach = [rmars-rmars*tan(gam0),-rmars; (rmars+h(1)),0]/10^3;
```

```
plot(approach(:,1),approach(:,2),':',approach(2,1),approach(2,2),'g^',...
    (h+rmars).*cos(theta)/10^3,(h+rmars).*sin(theta)/10^3,'k:',...
    (h(length(h))+rmars).*cos(theta(length(theta)))/10^3,...
    (h(length(h))+rmars).*sin(theta(length(theta)))/10^3,'r>',...
    (rmars).*cos(thetamars)/10^3,(rmars).*sin(thetamars)/10^3,'r')
axis equal;
ylabel('y (km)');
xlabel('x (km)');
title('Path of the MLV');
legend('Approach','Atmospheric Interface','Entry Trajectory',...
    'Touchdown','Mars');
figure
%subplot(2,1,2);
plot(p/10^3,h/10^3,'k:',p(1)/10^3,h(1)/10^3,'g^',...
    p(transitionind(1))/10^3,h(transitionind(1))/10^3,'^',...
p(transitionind(2))/10^3,h(transitionind(2))/10^3,'g>',...
p(length(p))/10^3,h(length(h))/10^3,'r>',[0;p(length(p))/10^3],[0; 0],...
'r-');
title('Flat Surface Approximation');
xlabel('Range (km)');
ylabel('Altitude (km)');
axis equal;
legend('Entry Trajectory','Atmospheric Interface',...
    'Aeroentry-Parachute Transition','Parachute-Propulsive Transition',...
    'Touchdown','Mars');
axis([0, max(p), 0, max(h)]/10^3);

figure;
subplot(2,1,1);
plot(t,v,t(transitionind(1)),v(transitionind(1)),'^',...
t(transitionind(2)),v(transitionind(2)),'>');
ylabel('Velocity (m/s)');
legend('Data','Aerodescent-Parachute Transition',...
    'Parachute-Propulsive Transition');
title('Various Properties of MLV Descent');
a = abs(diff(v)./diff(t)/9.8);
a(idescend) = a(idescend-1);
subplot(2,1,2);
plot(t(1:length(a)),a,t(transitionind(1)),a(transitionind(1)+1),'^',...
t(transitionind(2)),a(transitionind(2)),'>');
xlabel('Time (s)');
ylabel('Acceleration (Gs)');

figure;
plot(t(1:length(F)),F/10^3,[0;t(length(t))],[110 110],[0;t(length(t))],[11 11]);
xlabel('Time (s)');
ylabel('Force (kN)');
legend('Thrust Profile','Maximum Thrust','Minimum Throttle');
title('Thrust Profile of Trajectory');

figure;
subplot(2,1,1);
plot(t(1:length(sigplot)),sigplot*180/pi)
ylabel('Bank angle (degrees)');
subplot(2,1,2);
plot(t(1:length(phi)),phi*180/pi);
ylabel('Lattitude (degrees)');
xlabel('Time (s)');

figure;
plot(t,q,t(transitionind(1)),q(transitionind(1)),'^',...
    t(transitionind(2)),q(transitionind(2)),'>',...
    [0 length(t)],[600 600],':');
xlabel('Time (s)');
ylabel('Dynamic Pressure (Pa)');
legend('Data','Aerodescent-Parachute Transition',...
    'Parachute-Propulsive Transition','Q Loading Limit');

%%%%%%%%%%%%%%%%%%% CREATE TEXT FILE OUTPUT FOR HEATING %%%%%%%%%%%%%%%%%

fid = fopen('thermal.txt','w');
```

```
for ind = 1 : transitionind(2)

fprintf(fid,'%3.3f\t\t%8.8f\t\t%8.8f\t\t%8.8f\t\t%8.8f\t\t%8.8f\t\t%8.8f\t\t%8.8f\t\t%8.8f\t\t%8.8f\t\
t%8.8f\n', t(ind),
qw(ind,1),qw(ind,2),qw(ind,3),qw(ind,4),qw(ind,5),qw(ind,6),qw(ind,7),qw(ind,8),qw(ind,9),qw(ind,10));
end
fclose(fid);
```

Code 10 – Equation of Motion File for Descent Simulation (“eom.m”)

```
% EOM for Mars descent.
% By John Stalbaum
function xdot = eom(t,x)

global Cd LoD slast S damping htarget mdot ncp l rnose Cpnose rhonose Cpw qr rhow ...
    sig emiss Pr Cpe sarc Thwall cqwnose rmars GM h0;


% Ensures that the craft ends up near the equator, by pointing the bank
% angle in the correct direciton.
if abs(x(3)) > 0.5*pi/180
    if x(3) > 0
        s = -1;
    else
        s = 1;
    end
else
    s = slast;
end

slast = s;

h = x(1) - rmars;
if h >= htarget && h <= h0
    sigm = real(pi/2*((h-htarget)/(h0 - htarget))^damping)*s;
elseif h < htarget
    sigm = 0;
elseif h > h0
    sigm = pi/2*s;
end
[p, T, rho, a, visc] = marsatmosphere2(h);
Cl = Cd*LoD;

% State variable equations for motion
% Distance form center of mars, r
xdot(1) = x(4)*sin(x(5));
% Longitude (Around planet), theta
xdot(2) = x(4)*cos(x(5))*cos(x(6))/(x(1)*cos(x(3)));
% Lattitude (from poles), phi
xdot(3) = x(4)*cos(x(5))*sin(x(6))/x(1);
% Velocity, v
xdot(4) = -rho*S*Cd*x(4)^2/(2*x(7))-GM/(x(1))^2*sin(x(5));
% Flight Path Angle, gam
xdot(5) = rho*S*Cl*x(4)/(2*x(7))*cos(sigm)-(GM/(x(1))^2-x(4)^2/x(1))*cos(x(5))/x(4);
% Heading angle, psi
xdot(6) = rho*S*Cl*x(4)/(2*x(7)*cos(x(5)))*sin(sigm)-x(4)/x(1)*cos(x(5))*cos(x(6))*tan(x(3));
% Mass
xdot(7) = mdot;
%  Stagnation point heat flux:
Vinf = x(4);

%if Vinf > 30e3 || Vinf < 300
%    disp('Error! Bad Vinf');
%end

% Nose temperature
Twnose = x(8);
Cpwnose = Cpnose;
qwnose = cqwnose*sqrt(rho/rnose)*Vinf^3.04*(1-(Cpwnose*Twnose/(0.5*Vinf^2)));
```

```
%  Compute point of laminar-turbulent transition:
Minf = Vinf/a;
alpha = 10*pi/180;
Me = Minf*cos(alpha);
exparg = 1.209e-4*Me^2.641;
if exparg > 1.5
    Rext = 1e30;
else
    Rext = 10^(6.421*exp(exparg));
end

%  Compute transition Reynolds:
muinf = 1.716e-5*(T/273)^(1/5)*(273+111)/(T+111);
xtrans = Rext*muinf/(rho*Vinf);

%  Setup state variable for heating at nose:
x1b = rnose*pi/2;
g1b = Cpwnose*Twnose/(0.5*Vinf^2);
if x1b < xtrans
    cq = 1e4*2.53e-9*cos(alpha)^(1/2)*sin(alpha)*(1-g1b)/sqrt(x1b);
    expN = 0.5;
    expM = 3.2;
else
    expN = 0.8;
    if Vinf <= 3962
        cq = 1e4*3.35e-8*cos(alpha)^(1.78)*sin(alpha)^(1.6)*x1b^(-0.2)*(Twnose/556)^(-1/4)*(1-
1.11*g1b);
        expM = 3.37;
    else
        cq = 1e4*2.2e-9*cos(alpha)^(2.08)*sin(alpha)^(1.6)*x1b^(-0.2)*(1-1.11*g1b);
        expM = 3.7;
    end
end

qw1b = cq*rho^expN*Vinf^expM;

if Twnose > 200
    qw(1) = qwnose/4+qw1b/4-emiss*sig*Twnose^4;
else
    qw(1) = 0;
end

xdot(8) = (2/(rnose*Cpwnose*rhonose))*qw(1); % Wall temperature at stagnation point.

%  Setup state variable for heating for remainder of body
for i = 9:ncp+7
    itpt = i-7;
    Tw = x(i);
    xarc = rnose*pi/2+sarc(itpt);
    g = Cpw*Tw/(0.5*Vinf^2);
    phi = alpha;            % <= *
    if Minf*sin(phi) < 1
        %disp('Warning: Angle is too small for Tauber equation.');
        % Compute reference temperature
        Tref = T*(0.5+0.039*Minf^2+0.5*Tw/T);
        rhoref = rho*(T/Tref);
        muref = 1.716e-5*(Tref/273)^(1/5)*(273+111)/(Tref+111);
        Rexe = rho*Vinf*xarc/muinf;
        cstar = (Tref/T)^(-1/3);
        if xarc < xtrans
            cfe = 0.664*sqrt(cstar/Rexe);
            che = 0.5*cfe/Pr^(2/3);
            Taw = T*(1+sqrt(Pr)*0.2*Minf^2);
            qwconvect = che*rho*Vinf*Cpe*(Taw-Tw);
        else
            Rexref = rhoref*Vinf*xarc/muref;
            cfe = 0.027/Rexref^(1/7);
            che = 0.5*cfe/Pr^(2/3);
            Taw = T*(1+(Pr^(1/3))*0.2*Minf^2);
            qwconvect = che*rho*Vinf*Cpe*(Taw-Tw);
        end
    else
        if xarc < xtrans
            cq = 1e4*2.53e-9*cos(phi)^(1/2)*sin(phi)*(1-g)/sqrt(xarc);
            expN = 0.5;
            expM = 3.2;
        else
            expN = 0.8;
            if Vinf <= 3962
                cq = 1e4*3.35e-8*cos(phi)^(1.78)*sin(phi)^(1.6)*xarc^(-0.2)*(Tw/556)^(-1/4)*(1-
1.11*g);
                expM = 3.37;
            else
                cq = 1e4*2.2e-9*cos(phi)^(2.08)*sin(phi)^(1.6)*xarc^(-0.2)*(1-1.11*g);
                expM = 3.7;
            end
        end
        qwconvect = cq*rho^expN*Vinf^expM;
    end
    qw(itpt) = qwconvect-emiss*sig*Tw^4;
    xdot(i) = qw(itpt)*1/(Cpw*rhow*Thwall(itpt));
end

xdot = xdot';

% End of Function
```

Code 11 – Preliminary Method for MLV Descent Simulation (“mlvdescent.m”)

```
% This code numerically integrates the equations of motion of the lander to
% propagate the trajectory.
% Various properties must be fiddled with in each new case of mass which is
% applied to the problem.  Be forwarned.
% You must change the damping ratio so that the lift-to-drag ratio and
% G-loading don't become too extreme during the trajectory.  The initial
% angle that you enter at (defined from the horizontal) also affects this.
% The ways in which these affect the trajectory are kind of unpredictable.
% FOR THIS STAGE: the parachute mass and aeroshell mass must be manually
% staged away, so if these change, they must be edited in this code.

% By John Stalbaum and Jackie Jaron

% Items of particular interest that can be extracted from this code (set
% forth by sense, and the thermal group):
% 1) Time
% 2) Velocity (m/s)
% 3) Mach number
% 4) Atmospheric density (kg/m^3)
% 5) Atmospheric viscosity
% 6) Atmospheric temperature
% 7) Gamma (I don't know what it's called but it's 1.4 for air or something -
% if it's possible to get it)
% 8) Mass
% 9) Delta-V estimates.
% 10.) Angle of attack.
% Brief answer to gamma: 1.3ish. This will vary with flow conditions in the
% Hypersonic range, but that's tough to actually get.

% Future plans for this code: Implement a lower G-force generating,
% staged paracute scheme.

% Establish outputs in line with requirements:
function [t, vprime, M, rho, mu, T, m, dV, pprime, hprime,F] = mlvdescent(gam0,m0,damping)
% clear all; close all; clc;
% m0 = 3.0253e+004;
% gam0 = 7.3*pi/180;       % defined horizontal
% damping = 0.069;


% Function inputs
h0 = 100e3;
```

```
Isp = 465;
A = pi*4^2; % Reference area
% Chute characteristics.
Cdchute = 0.7;
Achute = pi*21.4^2;
htarget1 = 12e3; % The target for the aeroshell phase to end
htarget2 = 700; % The taget for the parachute phase to end.

% Based on the initial entry angle and interface altitude, this script
% calculates the initial velocity and delta-V for de-orbit.  Written by
% George Pollock.
[delV_deorb, v_entry] = entry_interface(-gam0*180/pi, h0/10^3); % Code requires degrees, km input
delV_deorb = 10^3*delV_deorb; % Returns delta-V and v_entry in km/s; must get in m/s.
v0 = 10^3*v_entry;

% Constants
dt = 0.1; % Timestep
rmars = 3397e3; % Radius of mars, meters.
vsurface = 2*pi*rmars/(24.6597*60^2); % The relative velocity of Mars' surface to lander.
Cd = 0.8; % Aeroshell coefficient of drag.
gmars = 3.72; % Martian gravitational acceleration.
g = 9.8; % Earth's g, for specific impulse calculations.

% A few words on frames: the default frame is fixed in inertially
% The prime frame is fixed on the mars surface.

% Define initial conditions
i = 1;
v(1) = v0;
t(1) = 0;
m(1) = m0; % This function estimates the initial mass.
gam(1) = pi-gam0; % The initial, inputted inclination angle.
h(1) = h0;
p(1) = 0; % Define the initial range as zero - arbitrary.
[P(1),T(1),rho(1) a(1) mu(1)] = marsatmosphere2(h(1));
vx(1) = -v(1)*cos(gam(1));
vy(1) = -v(1)*sin(gam(1));
v(1) = sqrt(vx(1)^2 + vy(1)^2);
if a(1) > 0
    M(1) = v(1)/a(1);
else
    M(1) = 0;
end
hprime(1) = h(1);
vxprime(1) = vx(1) - vsurface;
vyprime(1) = vy(1);
vprime(1) = sqrt(vxprime(1)^2+vyprime(1)^2);
pprime(1) = 0;
LoD(1) = damping*(hprime(1)-htarget1)/(hprime(1)-htarget1);
gam(1) = pi+atan(vyprime(1)/vxprime(1));

% Main functional loops
q(i) = 700;
% Aerodescent portion:
exit = 0;
%while exit == 0 && hprime(i) > 0
while M(i) > 2 && hprime(i) > 0
    F(i) = 0;
    [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
    q(i) = 1/2*rho(i)*vprime(i)^2;
    if a(i) ~= 0
        M(i) = vprime(i)/a(i);
    else
        M(i) = 0;
    end
    dmdt = -F(i)/(Isp*g);
    D(i) = Cd*q(i)*A;
    Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
    Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
    L(i) = LoD(i)*D(i);
    Ly(i) = L(i)*sin(gam(i)-pi/2);
```

```
        Lx(i) = L(i)*cos(gam(i)-pi/2);
        dvydt(i) = (F(i)*sin(gam(i)) - m(i)*gmars +Dy(i)+Ly(i))/m(i);
        dvxdt(i) = (Dx(i) + F(i)*cos(gam(i))+Lx(i))/m(i);
        r = rmars + hprime(i);
        vy(i+1) = vy(i) + dvydt(i)*dt;
        vx(i+1) = vx(i) + dvxdt(i)*dt;
        h(i+1) = h(i) + vy(i)*dt;
        p(i+1) = p(i) + vx(i)*dt;
        t(i+1) = t(i) + dt;
        m(i+1) = m(i) + dmdt*dt;
        v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
         hprime(i+1) = sqrt((rmars+h(i+1))^2+p(i+1)^2)-rmars;
         vxprime(i+1) = vx(i+1) - vsurface; % Velocity with respect to the surface
         vyprime(i+1) = (hprime(i+1)-hprime(i))/dt;
         vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
         pprime(i+1) = pprime(i) + vxprime(i)*dt;
         gam(i+1) = pi+atan(vyprime(i+1)/vxprime(i+1));
         LoD(i+1) = damping*(hprime(1)-htarget1)/(hprime(i+1)-htarget1);
         if LoD(i+1) > 2 || LoD(i+1) < 0
             LoD(i+1) = 2;
         end
         M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
         if (q(i) < 600 && hprime(i) < 80e3)
            exit = 1;
         end
         i = i + 1;
    end
    iparachute = i;
    m(i) = m(i) - 1000;

    %blargh = input('Pause!');
    % Parachute portion
    while hprime(i) > htarget2
        F(i) = 0;
        [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
        q(i) = 1/2*rho(i)*vprime(i)^2; % Using mars
        if a(i) ~= 0
            M(i) = vprime(i)/a(i);
        else
            M(i) = 0;
        end
        dmdt = -F(i)/(Isp*g);
        D(i) = q(i)*(Cd*A+Cdchute*Achute);
        Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
        Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
        L(i) = LoD(i)*D(i);
        Ly(i) = L(i)*sin(gam(i)-pi/2);
        Lx(i) = L(i)*cos(gam(i)-pi/2);
        dvydt(i) = (F(i)*sin(gam(i)) - m(i)*gmars +Dy(i)+Ly(i))/m(i);
        dvxdt(i) = (Dx(i) + F(i)*cos(gam(i))+Lx(i))/m(i);
        r = rmars + hprime(i);
        vy(i+1) = vy(i) + dvydt(i)*dt;
        vx(i+1) = vx(i) + dvxdt(i)*dt;
        h(i+1) = h(i) + vy(i)*dt;
        p(i+1) = p(i) + vx(i)*dt;
        t(i+1) = t(i) + dt;
        m(i+1) = m(i) + dmdt*dt;
        v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
         hprime(i+1) = sqrt((rmars+h(i+1))^2+p(i+1)^2)-rmars;
         vxprime(i+1) = vx(i+1) - vsurface; % Velocity with respect to the surface
         vyprime(i+1) = (hprime(i+1)-hprime(i))/dt;
         vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
         pprime(i+1) = pprime(i) + vxprime(i)*dt;
         gam(i+1) = pi+atan(vyprime(i+1)/vxprime(i+1));
         LoD(i+1) = 0;
         M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
         i = i + 1;
    end
    m(i) = m(i) - 1400;

    % Third portion - forced deceleration to 0 m/s at 100 m
```

```
% The acceleration is enough to arrest all velocity at the particular 100 m
% target.
% The situation is simplified, at this point.  The normal frame is
% dropped.  The vehicle is assumed to be close enough to the surface that
% it can be approximated as flat.
%
vx(i) = vxprime(i);
vy(i) = vyprime(i);
ithrust = i;

while hprime(i) > 100 && i < ithrust+1000
    [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
    q(i) = 1/2*rho(i)*vprime(i)^2; % Using mars
    if a(i) ~= 0
        M(i) = vprime(i)/a(i);
    else
        M(i) = 0;
    end
    D(i) = q(i)*(Cd*A+Cdchute*Achute);
    Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
    Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
    L(i) = LoD(i)*D(i);
    Ly(i) = L(i)*sin(gam(i)-pi/2);
    Lx(i) = L(i)*cos(gam(i)-pi/2);
    % Acceleration that would hit change the velocity in the distance
    % interval (from kinematics).
    if abs(hprime(i)-100) > 10
        aydesired = -sign(vyprime(i))*(vyprime(i))^2/(2*(hprime(i)-100));
        timedecel = abs(vyprime(i))/aydesired;
        axdesired = -vxprime(i)/timedecel;
    else
        axdesired = 0;
        aydesired = 0;
    end
    Fy(i) = (m(i)*aydesired+m(i)*gmars-Dy(i)-Ly(i));
    Fx(i) = (m(i)*axdesired-Dx(i)-Lx(i));
    F(i) = sqrt(Fx(i)^2+Fy(i)^2);
    dmdt = -F(i)/(Isp*g);
    dvydt(i) = (Fy(i) - m(i)*gmars +Dy(i)+Ly(i))/m(i); % F(i)*sin(gam(i))
    dvxdt(i) = (Dx(i) + Fx(i)+Lx(i))/m(i); % *cos(gam(i))
    r = rmars + hprime(i);
    vy(i+1) = vy(i) + dvydt(i)*dt;
    vx(i+1) = vx(i) + dvxdt(i)*dt;
    h(i) = hprime(i);
    hprime(i+1) = hprime(i) + vy(i)*dt;
    p(i+1) = p(i) + vx(i)*dt;
    t(i+1) = t(i) + dt;
    m(i+1) = m(i) + dmdt*dt;
    v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
     vxprime(i+1) = vx(i+1); % Velocity with respect to the surface
     vyprime(i+1) = vy(i+1);
     vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
     pprime(i+1) = pprime(i) + vxprime(i)*dt;
     gam(i+1) = pi+atan(vyprime(i+1)/vxprime(i+1));
     LoD(i+1) = 0;
     M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
     dV(i) = F(i)/m(i);
     i = i + 1;
end
idescend = i;
vx(i) = 0;
vy(i) = -5/3;
hprime(i) = 100;

% The final descent, with 100 meters of cross-range capability built in.
% The time interval.
for i = idescend : idescend + 60/dt
    % Atmospheric script
    [P(i),T(i),rho(i) a(i) mu(i)] = marsatmosphere2(hprime(i));
    q(i) = 1/2*rho(i)*vprime(i)^2; % Using mars
    % Mach number, protected from division by zero.
```

```
        if a(i) ~= 0
            M(i) = vprime(i)/a(i);
        else
            M(i) = 0;
        end
        % Various aerodynamic forces defined.
        D(i) = q(i)*(Cd*A);
        Dy(i) = -sign(vyprime(i)*sin(gam(i)))*sin(gam(i))*D(i);
        Dx(i) = -sign(vxprime(i)*cos(gam(i)))*cos(gam(i))*D(i);
        L(i) = LoD(i)*D(i);
        Ly(i) = L(i)*sin(gam(i)-pi/2);
        Lx(i) = L(i)*cos(gam(i)-pi/2);
        vy(i+1) = -5/3; % Decelerates to 0 m/s in 60 seconds.
        if i < idescend + 30/dt; % If it's not done accelerating,
            dvxdt(i+1) = -2*100/30^2; % Accelerate.
            gam(i+1) = pi/2-abs(atan(vx(i)/vy(i)));
        else % If not,
            dvxdt(i+1) = 2*100/30^2; % Decelerate.
            gam(i+1) = pi/2+abs(atan(vx(i)/vy(i)));
        end
        dvydt(i+1) = 0; % Descending at a constant rate of speed.
        vx(i+1) = vx(i) + dvxdt(i)*dt; % Update for cross-range.
        % Acceleration that would hit change the velocity in the distance
        % interval (from kinematics).
        Fy(i) = (m(i)*gmars-Dy(i)-Ly(i)); % Obtain the force components
        Fx(i) = (m(i)*dvxdt(i)-Dx(i)-Lx(i)); % required for acceleration.
        F(i) = sqrt(Fx(i)^2+Fy(i)^2); % Magnitude of force.
        dmdt = -F(i)/(Isp*g); % Flow rate, based on force magnitude.
        % Various updating of variables, and updating of things which aren't
        % used in this final trajectory which approxiates for reference frames.
        r = rmars + hprime(i);
        h(i) = hprime(i);
        hprime(i+1) = hprime(i) + vy(i)*dt;
        p(i+1) = p(i) + vx(i)*dt;
        t(i+1) = t(i) + dt;
        m(i+1) = m(i) + dmdt*dt;
        v(i+1) = sqrt(vx(i+1)^2 + vy(i+1)^2);
         vxprime(i+1) = vx(i+1); % Velocity with respect to the surface
         vyprime(i+1) = vy(i+1);
         vprime(i+1) = sqrt(vxprime(i+1)^2+vyprime(i+1)^2);
         pprime(i+1) = pprime(i) + vxprime(i)*dt;
         gam(i+1) = pi/2;
         LoD(i+1) = 0;
         M(i+1) = M(i); % This just makes sure that M is defined, so the loop can run.
         dV(i) = F(i)/m(i);
    end

    % Not yet implemented.
    for i = 1 : length(F)
        if F(i) > 1e5
            F(i) = 1e5;
        end
    end
    dV = sum(dV)*dt;

    % Finally, produce a set of plots that are characteristic of the motion of
    % the craft. (For version of code which is not a function).
    % plot(t,LoD);
    % xlabel('Time (s)');
    % ylabel('Lift-to-Drag Ratio (unitless)');
    % title('Lift-to-drag Ratio of the Lander Entry Trajectory');
    % dvdt = sqrt(dvydt.^2+dvxdt.^2);
    % Gs = dvdt/9.8;
    % figure(2);
    % blargh = plot(t(1:length(vprime)),vprime);
    % set(blargh,'linewidth',2);
    % xlabel('Time (s)');
    % ylabel('Velocity (m/s)');
    % title('Velocity Profile of the Lander');
    % figure(3);
    % blargh = plot(t(1:length(dvdt)),dvdt/9.8);
```

```
% set(blargh,'linewidth',2);
% xlabel('Time (s)');
% ylabel('Acceleration (Gs)');
% title('G-Loading of the Lander');
%
% figure(4);
% blargh = plot(pprime,hprime);
% set(blargh,'linewidth',2);
% xlabel('Range (m)');
% ylabel('Altitude (m)');
% title('Flight Path of the Lander');
%
% figure(5);
% blargh = plot(t(1:length(rho)),rho);
% set(blargh,'linewidth',2);
% xlabel('Time (s)');
% ylabel('Density (kg/m^3)');
% title('Density Profile of Trajectory');


% End of Script
```

Code 12 – Lander Initial Mass Estimation Script

```
% AAE 450 lander weight estimation script
% by John Stalbaum
% This script estimates the initial lander (MLV, Ares) mass based upon the
% final mass, delta-V for the orbital injection, and delta-V for the
% transfer into the rendezvous orbit.  It does this by doing a 3-stage
% rocket problem, considering the first stage as the descent stage, the
% second stage as the first ascent stage, and the third stage as the second
% ascent stage.
% Inputs:
% mfinal = Lander payload mass (final mass in that will be lifted into the
% rendezvous orbit
% deltaVrend = velocity increment from 350 km circular orbit to the orbit
% in which the lander and CTV rendezvous.
% deltaVinj = velocity increment from the parking orbit to orbital
% injection.


function mtotal = m0lander(mfinal,deltaVrend,deltaVinj)

% Constants
g = 9.81;
% The code is done vectorially; all quantities are defined for a stage in
% the respective element of each vector.

% delta-V assignments  for various stages.  The descent stage delta-V comes
% from initial estimates of each stage, the number "3731" comes from a
% trade study done by Jackie Jaron (presentation listed on the website),
% and deltaVrend and deltaVinjection are inputs.  The two ascent stages
% have the delta-V achieved by each stage divided between them equally
% (which isn't necessarily optimum, but can be improved later).
deltav = [(1000+deltaVinj) ((3731+deltaVrend)/2) ((3731 + deltaVrend)/2)];

% Assuming the optimistic specific impulse given by Rob Anderson for an
% NTO/MMH propulsion system.
Isp = [460 460 460];

% Same with mass fractions.
lamb = [0.9 0.9 0.9];


k = exp(deltav./(Isp*g)); % m0/mf
% Each stage has three equations giving information on it:
% m0/mf = (minert + mpayload + mpropellant)/(minert + mpayload) (1)
% lambda = mpropellant/(mpropellant + minert)                   (2)
% mpayload = m(previous stages mass)                            (3)
% These two equations can be rearranged to form the linear system:
% A*x = b, implying the augmented matrix [A b]
% The vector x is defined as:
% x = [mpayload_1, mpayload_2, mpayload_3, minert_1, minert_2,
% minert_3,mpropellant_1, mpropellant_2, mpropellant_3]

% The matrix A, mentioned above.
A = [0 0 1 0 0 0 0 0 0; % A matrix for linear system defined in
    0 -1 1 0 0 1 0 0 1; % handwritten work
    -1 1 0 0 1 0 0 1 0;
    0 0 0 lamb(1) 0 0 lamb(1)-1 0 0
    0 0 0 0 lamb(2) 0 0 lamb(2)-1 0
    0 0 0 0 0 lamb(3) 0 0 lamb(3)-1
    1-k(1) 0 0 1-k(1) 0 0 1 0 0
    0 1-k(2) 0 0 1-k(2) 0 0 1 0
    0 0 1-k(3) 0 0 1-k(3) 0 0 1];

% The matrix b, mentioned above.
B = [mfinal; 0; 0; 0; 0; 0; 0; 0; 0];

% The solution to the linear system is given by the right column of the
% reducted row-eschelon form of the augmented matrix.
soln = rref([A B]);

% At this point, all of the desired variables are recovered from the
% system.
soln = soln(:,10);
mp(1) = soln(1);
mp(2) = soln(2);
mp(3) = soln(3);
mi(1) = soln(4);
mi(2) = soln(5);
mi(3) = soln(6);
mf(1) = soln(7);
mf(2) = soln(8);
mf(3) = soln(9);

% Finally, the total vehicle mass is going to be the sum of the inert,
% payload, and propellant mass of the first stage (since all other stages
% are included in the payload mass of this vehicle).
mtotal = mf(1) + mi(1) + mp(1);

% End of Script
```

References

[1] Vinh, Nguyen X., Busemann, Adolf, Culp, Robert D. Hypersonic and planetary *entry* flight mechanics. Ann Arbor : University of Michigan Press, 1980.

39 Statler, Chris

39.1 MHV Descent Trajectory Appendix

Author: Chris Statler

39.1.1 The Effect of a Ballute on the Mars Habitat Vehicle (MHV) Entry Trajectory

Although the use of a ballute might be concerning to some, the reasons behind the decision to use on were well thought out and verified mathematically. The purpose of this section is to demonstrate the effect of the ballute and the consequences incurred if it is not used.

Using the same code that produced the results in Chapter 2, a simulation was done without the ballute and compared to the solution with the ballute. The lifting body, parachute and powered descent phases all remained the same to show the significant effect of the ballute.

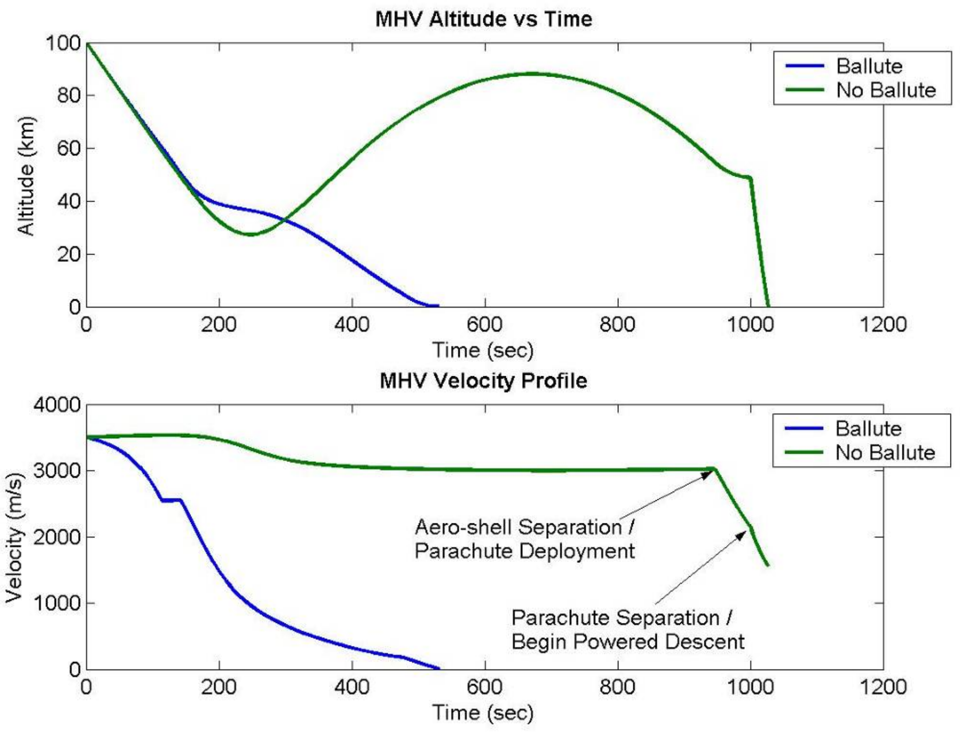


Figure 39-1 Altitude and Velocity Profiles

Figure 39-1 shows the altitude and velocity profiles without the use of a ballute. The wave-like altitude behavior that is demonstrated for the first 900 seconds or so is due to the fact that the aero-shell is a lifting body. Without the ballute, the MHV is not able to deploy its parachutes until nearly

1,000 seconds into the mission, at an altitude of 54 km and velocity of 3020 m/s. The mission is designed such that the parachutes separate at an altitude of 5 km (to give the parachutes enough time to separate and the powered descent enough time to slow the vehicle), which here corresponds to a velocity of 2150 m/s. At this point the powered descent initiates, burning for 26 seconds, only creating a ΔV of 590 m/s, and takes more than 2000 kg of propellant. This is not adequate to land the MHV successfully, as it hits the surface with a velocity of 1550 m/s. As we said before, this case was given the same inputs as the solution provided in Chapter 2, but without the ballute. There are a number of things that could be changed to maximize the amount of time the parachutes are deployed or have a longer powered descent. However, the bottom line still remains: the mass of propellant to bring the vehicle to an acceptable landing velocity at the surface (<10 m/s), without the use of a ballute, is very large. In this case, with RD-0146 engines, the propellant mass needed to bring the MHV down to the successful velocity can be obtained from Equation 39-2 to be 21,245 kg.

$$\frac{m_f}{m_0} = \exp\left(\frac{\Delta V}{I_{sp} * g_0}\right)$$

Equation 39-1 Rocket Equation

This would give the MHV an atmospheric entry mass (minus heat shield and aero-shell) of approximately 77,000 kg. According to current heat shield knowledge and material limitations, it is unreasonable to assume a mass this large can be shielded from the intense heat of atmospheric entry. The mass of the MHV (minus heat shield and aero-shell) in the final configuration presented in Chapter 2 is approximately 55,000 kg. This is a difference of 22,000 kg, and very costly with respect to protecting that mass from entry temperatures. A basic example of the cost is to analyze Equation 39-2 for the ballistic coefficient.

$$\beta = \frac{W}{S * C_d}$$

Equation 39-2 Ballistic Coefficient

In order for it to remain constant (same as with the ballute) with the increase in entry mass, the reference area (S) must go up by the same percentage as the weight (a 40% increase). With the 40% increase in reference area stated above, sizing a shield of this magnitude returns problems of knowledge and material limitations. A further detailed analysis of heat shield designs and limitations can be seen in the MHV Heat Shield section of Chapter 2, written by Jason Friel.

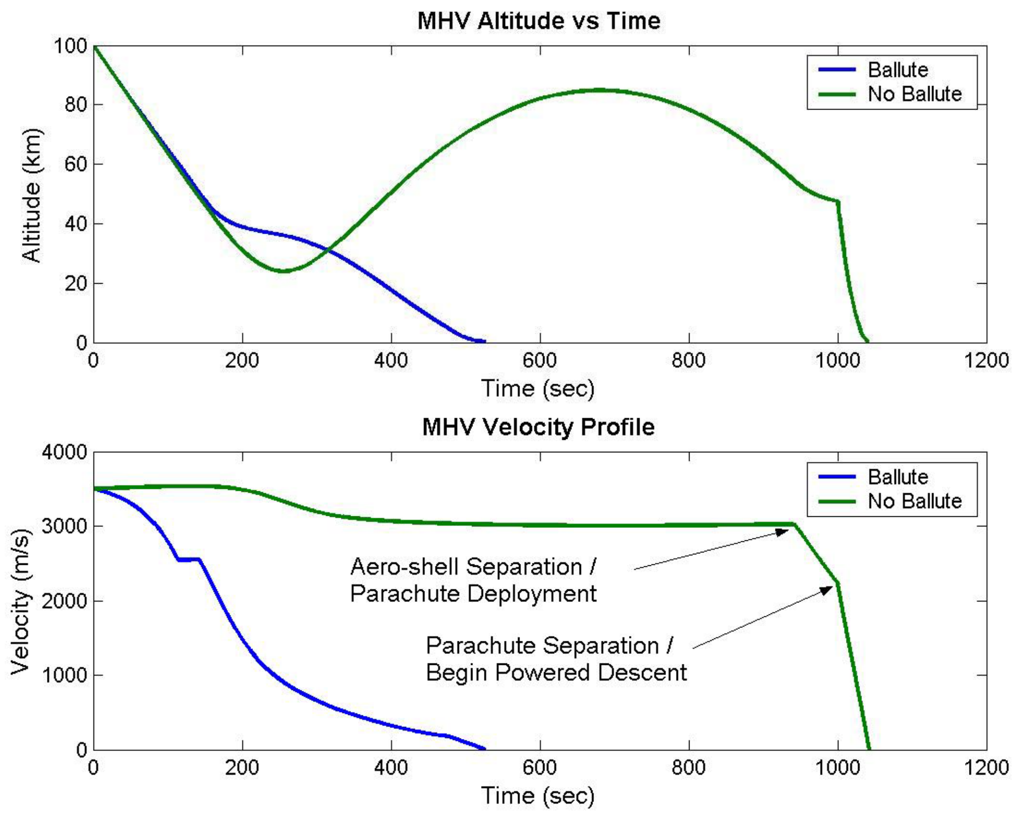


Figure 39-2 MHV Altitude and Velocity with no Ballute

Using the estimation for propellant mass, we ran a simulation to have the MHV land successfully on the surface, producing Figure 39-2 above. The estimation of 21,425 kg propellant turns out to be very accurate. Again, this creates the problem that the entry mass without the aero-shell and heat shield are around 75,000 kg. A detailed mass and G-loading profile is shown in Figure 39-3 on the next page.

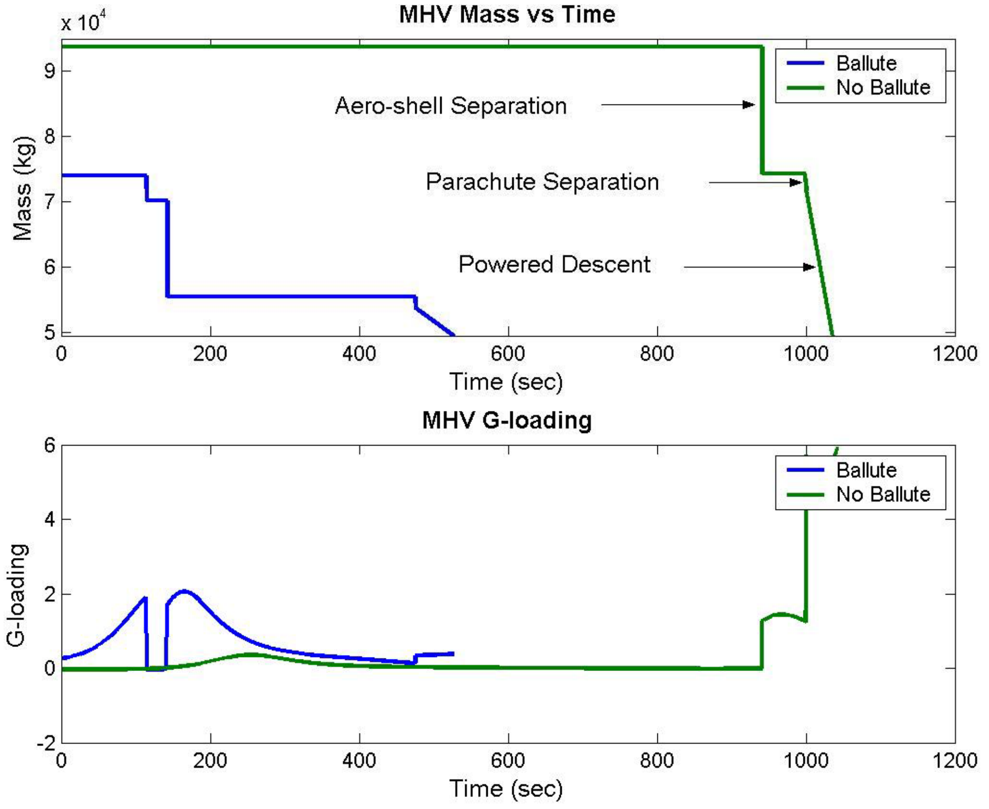


Figure 39-3 Mass and G-loading for MHV with no Ballute

In analyzing these results, we determine that the ballute is necessary to have a technically feasible solution for landing the MHV on the Martian surface. Without the ballute, the propellant masses required for a safe landing velocity are extremely large and push the realistic solutions for heat shield design and material limitations. Again, more in depth analysis on heat shield design can be seen in the MHV Heat Shield section, written by Jason Friel.

39.1.2 Code for MHV Descent

The following is the MATLAB code used to simulate the equations of motion. There are three files: the EOM file which just defines the equations of motion listed in Chapter 2, the EOM file which models the powered descent and the program to simulate these equations using the inputs as the characteristics of the MHV at each of the stages. The main program outputs altitude, range, velocity, mass, dynamic pressure, g-loading, and atmospheric density versus time.

eom.m

Variables:

CD = drag coefficient of MHV

CL = lift coefficient of MHV

m0 = mass of the MHV (kg)

sigma = bank angle (rad)

S = reference area of MHV (m^2)

rm = radius of Mars (km)

q = dynamic pressure (N/m^2)

rho = density (kg/m^3)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Chris Statler
%AAE 450 - Spacecraft Design - Spring 2005
%Entry Trajectory for MHV
%
%Equations of Motion from "Hypersonic and Planetary Entry Flight Mechanics", University
%of Michigan Press.  Assumes spherical and non-rotating planet.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pdot = eom(t,p)

global CD CL m0 mu sigma S rm

[Pressure, Temp, rho, a, visc] = marsatmosphere2(p(1)-rm);

q = 0.5*rho*p(4)^2;
pdot(1) = p(4)*sin(p(5));
pdot(2) = p(4)*cos(p(5))*cos(p(6))/(p(1)*cos(p(3)));
pdot(3) = p(4)*cos(p(5))*sin(p(6))/p(1);
pdot(4) = -rho*S*CD*p(4)^2/(2*m0) - mu/(p(1))^2*sin(p(5));
pdot(5) = rho*S*CL*p(4)/(2*m0)*cos(sigma) - (mu/(p(1))^2 - p(4)^2/p(1))*cos(p(5))/p(4);
pdot(6) = rho*S*CL*p(4)/(2*m0*cos(p(5)))*sin(sigma) - p(4)/p(1)*cos(p(5))*cos(p(6))*tan(p(3));

pdot = pdot';
```

eom2.m

Variables

g = Martian gravity constant (m/s^2)

F = thrust provided by engines (N)

mdot = mass flow rate of engine (kg/s)

pdot(1) = dx/dt

pdot(2) = dv/dt

pdot(3) = dm/dt

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Chris Statler
%AAE 450 - Spacecraft Design - Spring 2005
%Entry Trajectory for MHV
%
%Model of powered descent, assuming no transverse velocity, constant
%density and non-rotating planet
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pdot = eom2(t,p)

global CD CL m0 mu sigma S rm g F mdot

rho = 0.0151; %assume constant density for this portion

pdot(1) = p(2);
pdot(2) = F/p(3) + 0.5*rho*p(2)^2*CD*S/p(3) - g;
pdot(3) = -mdot;

pdot = pdot';
```

mhv.m

Variable descriptions are commented into the code. The eom.m and eom2.m files above must be present for this to run. The inputs are listed, as well as each of the descent phases. The characteristics for each phase are in the first five to six lines under each heading. In order to change the characteristics of the ballute, for example, change the "d1", "d2", "S", "CD", etc. under the "ballute descent" heading.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Chris Statler
%AAE 450 - Spacecraft Design
%Entry Trajectory for MHV
%
%This code inputs the characteristics of a ballute, lifting body aeroshell,
%parachutes, and powered descent system and calculates the entry
%trajectory.  The eom files are named eom.m and eom2.m, with the equations of motion
% for eom.m coming from "Hypersonic and Planetary Entry Flight Mechanics", University
%of Michigan Press.  Both of these files need to be present in order to run
%this code successfully.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc

global CD CL m0 mu sigma S rm g F mdot


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Timestep and integration time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
step = 0.1; %timestep to create time vector
tb = 1000; %simulation time for first three phases (arbitrary)


%%%%%%%%%%%%%%%%%%%%%%%%
%Martian Characteristics
%%%%%%%%%%%%%%%%%%%%%%%%
mu = 4.2828e4*1000^3; %gravitational parameter of Mars (m^3/s^2)
rm = 3397e3; %radius of Mars (m)
sigma =0*pi/180; %bank angle (rad)
g = 3.71; %Martian gravity constant (m/s^2)
g0 = 9.81; %Earth gravity constant (m/s^2)


%%%%%%%%%%%%%%%%%%%%
%MHV Characteristics
%%%%%%%%%%%%%%%%%%%%
m0 = 74000; %initial vehicle mass (kg)
mballute = 3800; %mass of ballute
mheat = 1/5*m0; %mass of heat shield/aeroshell
F = 370e3; %thrust for powered descent
Isp = 451; %Isp for powered descent
mdot = F/(Isp*g0); %mass flow rate of propellant


%%%%%%%%%%%%%%%%%%%
%Initial Conditions
%%%%%%%%%%%%%%%%%%%
h0 = 100e3; %m
theta0 = 0*pi/180; %rad
phi0 = 0*pi/180; %lattitude (rad)
v0 = sqrt(mu/(h0+rm)); %velocity
gamma0 = -6*pi/180; %flight path angle (rad)
psi0 = 0*pi/180; %heading angle (rad)
```

```
%%%%%%%%%%%%%%%%
%Ballute Descent
%%%%%%%%%%%%%%%%
d1 = 120; %outer diameter (m)
d2 = 100; %inner diameter (m)
S = pi/4*(d1^2 - d2^2); %reference area (m^2)
CD = 1.5; %drag coefficient
LOD = 0; %lift to drag ratio
CL = LOD*CD; %lift coefficient

t1 = linspace(0,tb,tb/0.1);
ic = [h0+rm theta0 phi0 v0 gamma0 psi0]; %initial conditions

[t1,p1] = ode45('eom',t1,ic);

z = 0; %define counter variable that counts through the simulation


for w1 = 1:length(p1)
    if p1(w1,1)-rm > 60e3 %assume ballute burns up at 60 km altitude
        z = z+1;
        [P, Temp, rho(z), a, visc] = marsatmosphere2(p1(w1,1)-rm);
        alt(z) = p1(w1,1)-rm;
        vel(z) = p1(w1,4);
        mass(z) = m0;
        theta(z) = p1(w1,2);
        t(z) = t1(w1);
    end
end


tfinal_1 = t(length(t)) %time at end of ballute phase
n = z;
q1 = 0.5*rho.*alt.^2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%
%Lifting Body Decent
%%%%%%%%%%%%%%%%%%%%
S = 8*6; %reference area (m^2)
CD = 0.7; %drag coefficient
LOD = 1.5; %lift to drag ratio
CL = LOD*CD; %lift coefficient
m0 = m0 - mballute; %define new mass w/o ballute

t2 = linspace(tfinal_1+step,tb,tb/0.1);
ic2 = p1(z,:); %initial conditions, conditions at end of ballute descent
[t2,p2] = ode45('eom',t2,ic2);


%create dynamic pressure vector to find optimal alt to deploy chutes
for w2 = 1:length(p2)
    n = n+1;
    [P2, Temp2, rho2(n), a2, visc2] = marsatmosphere2(p2(w2,1)-rm);
    q2(w2) = 0.5*rho2(n).*p2(w2,4).^2;
end
q2min = min(q2);


%find time @ acceptable dynamic pressures
z2 = 0;
e = 0;
for w2 = 1:length(p2)
    if q2(w2) < 600
        e = e+1;
        qgood(e) = q2(w2);
        tq(e) = t2(w2);
```

```
            count(e) = w2;
        end
    end

    %add to alt and vel vectors, using only points up until point at
    %which q<600 so that parachutes can be deployed at the end of the lifting body descent
    for w2 = 1:count(length(count))
        z = z+1;
        z2 = z2+1;
        rho(z) = rho2(z+w2);
        alt(z) = p2(w2,1)-rm;
        vel(z) = p2(w2,4);
        mass(z) = m0;
        theta(z) = p2(w2,2);
        t(z) = t2(w2);
    end

    tfinal_2 = t(length(t))%time at end of lifting body phase
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    %%%%%%%%%%%%%%%%%
    %Parachute Decent
    %%%%%%%%%%%%%%%%%
    d = 50; %diameter (m)
    S = pi/4*d^2; %reference area (m^2)
    CD = 0.8; %drag coefficient
    LOD = 0.3; %lift to drag ratio
    CL = LOD*CD; %lift coefficient
    m0 = m0-mheat; %define new mass, w/o heat shield/aeroshell

    t3 = linspace(tfinal_2+step,tb,tb/0.1);
    ic3 = p2(z2,:); %initial conditions, conditions at end of lifting body descent
    [t3,p3] = ode45('eom',t3,ic3);

    z3 = 0;
    for w3 = 1:length(p3)
        if p3(w3,1)-rm > 5e3; %altitude for end of parachute phase
            z = z+1;
            z3 = z3+1;
            [P, Temp, rho(z), a, visc] = marsatmosphere2(p3(w3,1)-rm);
            alt(z) = p3(w3,1)-rm;
            vel(z) = p3(w3,4);
            mass(z) = m0;
            theta(z) = p3(w3,2);
            t(z) = t3(w3);
        end
    end

    theta_end = theta(length(theta));
    tfinal_3 = t(length(t)) %time at end of parachute phase
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    %%%%%%%%%%%%%%%%
    %Powered Descent
    %%%%%%%%%%%%%%%%
    d = 8; %diameter (m)
    S = pi/4*d^2; %reference area (m^2)
    CD = 0.8; %drag coefficient
    LOD = 0; %lift to drag ratio
    CL = LOD*CD; %lift coefficient
    mchutes = 0.031*m0; %mass of parachutes calculated as 3.1% vehicle wght
    m0 = m0 - mchutes; %define new mass w/o parachutes

    tb = 2000; %timespan to simulate powered descent

    t4 = linspace(tfinal_3+step,tb,tb/0.1);
    ic4 = [alt(z) -vel(z) m0]; %initial conditions, conditions at end of parachute descent
```

```
[t4,p4] = ode45('eom2',t4,ic4);

z4 = 0;

for w4 = 2:length(p4)
    if p4(w4,1) > 0 && p4(w4,1) < p4(w4-1,1)
        z = z+1;
        z4 = z4+1;
        [P, Temp, rho(z), a, visc] = marsatmosphere2(p4(w4,1));
        alt(z) = p4(w4,1);
        vel(z) = -p4(w4,2);
        theta(z) = theta_end;
        t(z) = t4(w4);
        mass(z) = p4(w4,3);
    end
end

tfinal_4 = t(length(t)) %time at end of powered descent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Pressure, Temperature, Density, Speed of Sound and Viscosity throughout entry
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for x = 1:length(alt)
[P(x), Temp(x), rho_f(x), a(x), visc(x)] = marsatmosphere2(alt(x));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%
%G-loading throughout entry
%%%%%%%%%%%%%%%%%%%%%%%%%%%
gloading = (diff(vel)./diff(t))/g0;
for x = 1:length(gloading)
    if gloading(x) == NaN
        b = 1;
        gloading(x) = gloading(x-1);
    end
end


%%%%%%%%%%%%%%%%
%Calculate Range
%%%%%%%%%%%%%%%%
range=diff(theta).*(alt(1:length(alt)-1)+rm);
for a = 1:length(range)
    tot_range(a) = sum(range(1:a));
end
tot_range(length(alt)) = tot_range(a); %range throughout trajectory

%dummy variables to plot atmosphere, surface and parking orbit
tparking = linspace(0,800);
parking = 150*ones(1,length(tparking));
surface = 0*ones(1,length(tparking));
atmos = 100*ones(1,length(tparking));
%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%


%%%%%%%%
%Figures
%%%%%%%%
figure(1)
subplot(2,1,1)
blah = plot(t,alt/1000);
set(blah,'linewidth',2)
title('MHV Altitude vs Time','fontsize',12,'fontweight','b')
xlabel('Time (sec)','fontsize',12)
ylabel('Altitude (km)','fontsize',12)
subplot(2,1,2)
```

```
blah = plot(t,vel);
set(blah,'linewidth',2)
title('MHV Velocity Profile','fontsize',12,'fontweight','b')
xlabel('Time (sec)','fontsize',12)
ylabel('Velocity (m/s)','fontsize',12)

figure(2)
subplot(2,1,1)
blah = plot(t,mass);
set(blah,'linewidth',2)
title('MHV Mass vs Time','fontsize',12,'fontweight','b')
xlabel('Time (sec)','fontsize',12)
ylabel('Mass (kg)','fontsize',12)
subplot(2,1,2)
blah = plot(t(1:length(t)-1),-gloading);
set(blah,'linewidth',2)
title('MHV G-loading','fontsize',12,'fontweight','b')
xlabel('Time (sec)','fontsize',12)
ylabel('G-loading','fontsize',12)

figure(3)
blah = plot(tot_range/1000,alt/1000,tparking,parking,'--g');
set(blah,'linewidth',2)
hold on
plot(tparking,atmos,'c-',tparking,surface,'r')
hold off
axis([0 800 -100 700])
title('Flight Path of MHV','fontsize',12,'fontweight','b')
xlabel('Range (km)','fontsize',12)
ylabel('Altitude (km)','fontsize',12)
legend('Flight Path','Parking Orbit','Atmosphere','Mars')
patch([0 0 800 800],[-100 0 0 -100],'r')

figure(4)
subplot(2,1,1)
blah = plot(t,0.5*rho_f.*vel.^2);
set(blah,'linewidth',2)
title('Dynamic Pressure for MHV Descent','fontsize',12,'fontweight','b')
xlabel('Time (sec)','fontsize',12)
ylabel('Dynamic Pressure (N/m^3)','fontsize',12)
subplot(2,1,2)
blah = plot(t,rho_f);
set(blah,'linewidth',2)
title('Air Density During Entry','fontsize',12,'fontweight','b')
xlabel('Time (sec)','fontsize',12)
ylabel('Density (kg/m^3)','fontsize',12)
```

40 Szamborski, Timothy

40.1 Plant Growth Chamber Appendix

Author: Timothy Szamborski

40.1.1 Rationale for Growing Plants

Plants offer the greatest opportunity for self-sufficiency and cost reduction for long duration missions. Since our astronauts will be too far away from Earth to be able to continuously send them resupply ships, self-sufficiency plays a key role in being able to set up a potential colony. The growing of plants on this mission will help pave the way for future long duration space explorations.

A major concern for the mission is being able to store food for the required 10 year shelf life. Currently, the technology to store food for that duration of time has not been proven, mainly because no human mission of this length has been conducted before. By growing plants, we prevent the problem of food storage from ever occurring. Once the astronauts walk into the MHV, they are able to grow fresh food.

Another advantage in growing plants is the ability they have to produce fresh oxygen for the humans, while taking all of the waste that humans produce. Plants use carbon dioxide, water, and light to photosynthesize and grow. Conveniently, humans produce carbon dioxide, and waste water that we produce can be processed into clean water. As the plants scrub the carbon dioxide out of the air and take in our waste water, they grow and produce food and oxygen back for the humans.

Plants also take advantage of human waste and inedible food biomass. Human waste and food biomass can be processed in a bio-reactor, which will break down the waste into nutrients that the plants can make use of. These nutrients act as fertilizer for the plants, which help in the growing process. So aside from providing food and oxygen for the humans, plants also clean the air the astronauts breathe as well as take in all of the waste that humans produce.

40.1.2 Theory and Assumptions

To support a fully closed system, an estimate of 40 m^2 per crewmember is needed. Plants and humans form a symbiotic relationship in that everything humans produce, plants can use, and everything that plants produce, humans can use. This is what we mean by a totally closed system. Humans produce carbon dioxide and waste water, which the plants can then use to photosynthesize and grow. During the plant growth period, oxygen and food is produced, which the humans can then take advantage of. Extra human waste and plant bio-mass waste can then be processed in a bio-reactor, which will break down the waste into usable nutrients for the plants. These nutrients can then be added back to the plants and act like fertilizer. Figure 1-1 illustrates a totally closed system.

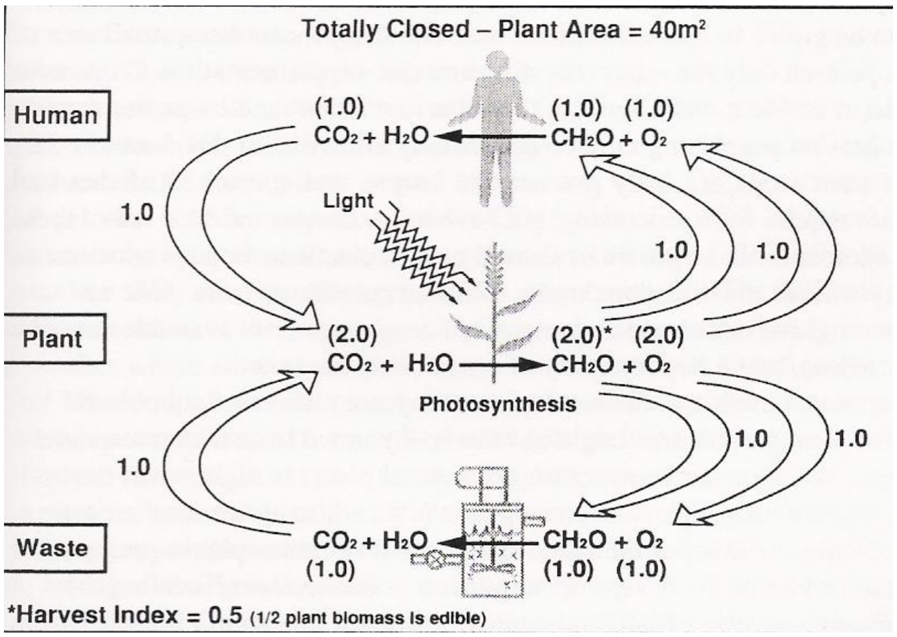


Figure 1-1 Totally Closed Plant System (Space Flight)

A fully-closed system, though is very efficient, would be too great of a cost to provide. An estimated total of 160 m^2 would need to be provided just for plants for a fully-closed system for our mission. Also, the mass would be too great given the duration of our mission. If we were setting up a permanent base on Mars, we would be far better off using a fully-closed system since it is completely self-sufficient, and we would not need to send resupply ships to the Martian surface like we do for the space station today. Mars is too far away from Earth to make this an economical choice.

However, our limited-stay on the Martian surface makes fully-closed systems too great of a cost in mass and is the main reason we are implementing a partially-closed system. An extra 80 m^2 of plants would need to be brought along to have a self-sustaining system. Aside from the extra mass in plants, we'd also need to account for extra structure to contain the extra volume of plants. The mass from the extra structure is far too great to make for a feasible mission. Therefore, as a cost savings, we will only grow 80 m^2 of plants.

Lighting will be provided through the use of light emitting diodes (LEDs). LEDs provide the exact lighting scheme that plants need to optimally grow. LEDs are also far more efficient that the current high-sodium lamps currently used in growing plants. We place the LEDs on icicle-like sticks and distribute them throughout the growing area, so that the canopy of the taller plants will not block out the light from the shorter plants. Again, this helps the plants optimally grow since the whole plant will be receiving light.

To optimally grow, lighting will be provided between 12-16 hours each day to the plants, depending on the species of plant. A common misconception is that if light helps the plants grow, why not provide light 24 hours a day? Plants begin to deteriorate with constant lighting and become extremely inefficient. Like human, plants need a "rest" period, where they actually operate in reverse order; they take in oxygen and produce carbon dioxide. These levels during the rest period are very minimal and have not been taken into account in this analysis.

Plant growth rates depend on the type of plant (species and cultivar) and the growth conditions. Optimal temperatures range from 20˚C-26˚C, again, depending on the type of plant. The plants will be arranged so that each species can be provided its optimal growing conditions.

All plants will also be grown using hydroponics. Hydroponics is the cultivation of plants in nutrient solution rather than in soil. These nutrients are provided from human waste and inedible biomass from the plants during harvesting.

40.1.3 500-day Mission

This initial mission outline was to provide enough resources for the astronauts for a 500-day stay on the Martian surface. Initial studies of this mission were conducted and are presented here. However, the feasibility of this length of stay on the surface proved to be infeasible. However, the data is still provided for future analyses of this system.

40.1.3.1 Type of Plants Grown

Initially, 80 m^2 was allocated for plant growth (or 20 m^2 for each crewmember). To provide the nutritional requirements, certain percentages of plants were grown, and have summarized in Table 1-1 below.

Table 1-1 Plants Grown for a 500-day mission

Plant	Growing Area [m^2]	Percentage of Growing Area
Cabbage	7	8.75
Carrot	4	5
Dry Bean	6	7.5
Lettuce	5	6.25
Onion	3	3.75
Peanut	4	5
Radish	3	3.75
Rice	3	3.75
Soybean	4	5
Spinach	3	3.75
Sweet Potato	6	7.5
Tomato	4	5
Wheat	24	30
White Potato	4	5
Total	80	100

40.1.3.2 Food Production

We planned on growing enough plants to provide half the food required for humans. We estimated that we would need approximately 20 m^2 per crewmember, or 80 m^2 of total plant growing area. Since we estimate that one astronaut consumes approximately 2.5 kg of food per day, for a 500 day mission, all four astronauts would require approximately 5,000 kg of food for the entire mission. So to provide half the food supply, we would need to grow approximately 2,500 kg of food. Using the Baseline Values and Assumptions Document written by NASA, nominal values for edible mass of a given plant were taken to determine how much food we could produce with our given quantity of plants. Table 1-2 illustrates how much of each plant we'd grow, and their respective edible biomass values.

Table 1-2 Edible Biomass Values for a 500-day mission

Plant	Growing Area [m^2]	Edible Biomass [kg]

Cabbage	7	220.1
Carrot	4	127.2
Dry Bean	6	27.7
Lettuce	5	310.0
Onion	3	110.5
Peanut	4	9.4
Radish	3	130.6
Rice	3	12.8
Soybean	4	8.1
Spinach	3	102.9
Sweet Potato	6	128.8
Tomato	4	288.4
Wheat	24	229.7
White Potato	4	155.0
Total	80	1,861.3

40.1.3.3 Oxygen Production

Using the Baseline Values and Assumptions Document (BVAD), values for oxygen production were manipulated for our 500-day stay on the Martian surface. The values given are the nominal assumptions made when the plant is fully grown. The assumption we made is that each plant always produces the amount given in the BVAD report, even before it is fully grown. This was an assumption made during the preliminary design process. For further analysis, we could account for the different levels of oxygen production during the plant growth period. Table 1-3 summarizes the values of oxygen production for each plant given its growing area.

As can be seen from Table 1-3, 1071.6 kg of oxygen is produced over the 500-day stay. From the BVAD report, it is assumed that each astronaut will consume approximately 0.835 kg of oxygen per day. Therefore, plants will supply approximately 64.2% of the oxygen needed for the astronauts.

Table 1-3 Oxygen Production Values for a 500-day mission

Plant	Growing Period [d]	O_2 Production [kg]
Cabbage	85	20.9
Carrot	75	27.8
Dry Bean	85	76.4
Lettuce	28	18.4
Onion	50	16.2
Peanut	104	56.8

Radish	25	16.9
Rice	85	45.5
Soybean	97	22.4
Spinach	30	11.0
Sweet Potato	85	102.4
Tomato	85	43.8
Wheat	79	565.8
White Potato	132	47.4
Total	N/A	1,071.6

40.1.3.4 Carbon Dioxide Uptake

Again, using the Baseline Values and Assumptions Document (BVAD), values for carbon dioxide uptake were manipulated for our 500-day stay on the Martian surface. The values given are once more the nominal assumptions made when the plant is fully grown. The assumption we made is that each plant always takes in the amount given in the BVAD report, even before it is fully grown. This was an assumption made during the preliminary design process. For further analysis, we could account for the different levels of carbon dioxide uptake during the plant growth period. Table 1-4 summarizes the values of carbon dioxide uptake for each plant given its growing area.

As can be seen from Table 1-4, plants can take in approximately 1474.8 kg of carbon dioxide over the 500-day stay mission. From the BVAD report, it is assumed that each astronaut will produce approximately 0.998 kg of carbon dioxide per day. Therefore, plants can take in approximately 73.9% of the carbon dioxide produced by the astronauts.

Table 1-4 Carbon Dioxide Uptake Values for a 500-day mission

Plant	Growing Period [d]	CO_2 Uptake [kg]
Cabbage	85	28.7
Carrot	75	38.3
Dry Bean	85	105.0
Lettuce	28	25.3
Onion	50	22.3
Peanut	104	78.1
Radish	25	23.2

Rice	85	62.6
Soybean	97	30.8
Spinach	30	15.1
Sweet Potato	85	140.8
Tomato	85	60.2
Wheat	79	778.0
White Potato	132	66.6
Total	N/A	1,474.8

40.1.3.5 Psychological Factors

The psychological factors regarding plant growth are extraordinary. Being in isolation for a couple of years with virtually no human interaction can cause severe psychological damage to a person. Growing plants allows the astronauts to not only feel like they are at “home”, but also lets the astronauts know that they are not the only living organisms out there. Just by seeing something they are used to growing and living there right next to them can easily be a comfort to the astronauts. The green color provided by plant leaves will also be a huge psychological benefit for the astronauts.

40.1.4 250-day Mission

After realizing the lack of feasibility for the 500-day mission due to extreme masses, the mission requirements regarding the duration of stay changed. Staying on the Martian surface for a shorter length of time significantly decreases masses for all systems; plants in particular. The requirement for providing enough food for half of the stay was also changed. The new requirement of the plants was to provide enough oxygen for half of the stay, or 125 days.

40.1.4.1 Type of Plants Grown

For a 250-day mission on the Martian surface, fewer plants are needed to provide the given requirements for oxygen, food, and waste uptake. Therefore, we chose to grow 60 m^2 of plants. The distribution of the plants grown can be seen in Table 1-5.

Table 1-5 Plants Grown for a 250-day mission

Plant	Growing Area [m^2]	Percentage of Growing Area
Carrot	5	8.33
Lettuce	10	16.67
Spinach	5	8.33
Tomato	10	16.67

Wheat	30	50
Total	60	100

The choice of plants and quantity of each was chosen based on the requirement to be able to produce enough oxygen for half of the mission stay. Since wheat produces the most oxygen and can scrub the most carbon dioxide, half of the growing area will consist of wheat. But for variety purposes, other plants are grown, such as lettuce, carrots, spinach, and tomatoes. This will add color to the plants, as well as still be able to provide different nutrients for the astronauts.

40.1.4.2 Food Production

Again, edible biomass values were taken from the BVAD report and were manipulated based on growing areas and the duration of the mission. As can be seen, lettuce provides the most food after harvesting, since the majority of it is edible. Even though it is half of our growing area, not much food is provided growing wheat. But again, the main reason for growing so much wheat was because of its ability to provide a lot of oxygen and scrub a lot of carbon dioxide form the atmosphere.

Table 1-6 Edible Biomass Values for a 250-day mission

Plant	Growing Area [m^2]	Edible Biomass [kg]
Carrot	5	65.5
Lettuce	10	291.6
Spinach	5	171.5
Tomato	10	286.7
Wheat	30	116.6
Total	60	931.9

40.1.4.3 Oxygen Production

Again, using the Baseline Values and Assumptions Document (BVAD), values for oxygen production were manipulated for our 250-day stay on the Martian surface. The values given are once again the nominal assumptions made when the plant is fully grown. The assumption we made is that each plant always produces the amount given in the BVAD report, even before it is fully grown. However, as this is our final analysis, we tried to account for the different levels of oxygen production during the growing periods of the plants.

A linear relationship was set up for the plants' ability to produce oxygen. Equation 1-1 below describes this relationship:

$$T = \sum P \times \left(\frac{d}{G}\right) \times A \qquad (1\text{-}1)$$

where T is the total amount of oxygen produced from the day the seeds are planted until the day the plant is fully matured. P is the amount of oxygen each plant is able to produce when it is fully matured; d is the day number and is what is being varied from day one until the day it is fully matured. G is the number days it takes each plant to fully grow, and A is the growing area (m^2).

We realize that this model is entirely accurate, but is far more precise than assuming that the plant is constantly producing a level of oxygen equal to that of when it is fully matured, even when the seeds are first planted. A further analysis on the growth patterns of plants could be taken into account, but this serves as a step in the right direction and produces reasonable values. Table 1-7 summarizes the values of oxygen production for each plant given its growing area.

As can be seen from Table 1-7, 454.3 kg of oxygen is produced over the 250-day stay. From before, it is assumed that each astronaut consumes approximately 0.835 kg of oxygen per day. Therefore, plants will supply approximately 54.4% of the oxygen needed for the astronauts.

Table 1-7 Oxygen Production Values for a 250-day mission

Plant	Growing Period [d]	O_2 Production [kg]
Carrot	75	17.4
Lettuce	28	18.4
Spinach	30	9.2
Tomato	85	54.8
Wheat	79	354.5
Total	N/A	454.3

40.1.4.4 Carbon Dioxide Uptake

Again, using the Baseline Values and Assumptions Document (BVAD), values for carbon dioxide uptake were manipulated for our 250-day stay on the Martian surface. The values given are once again the nominal assumptions made when the plant is fully grown. The assumption we made is that each plant always takes in the amount given in the BVAD report, even before it is fully grown. However, just like the oxygen production section, we tried to account for the different levels of oxygen production during the growing periods of the plants.

A linear relationship was set up for the plants' ability to produce oxygen. Equation 1-2 below describes this relationship:

$$T = \sum U \times \left(\frac{d}{G} \right) \times A \qquad (1\text{-}1)$$

where T is the total amount of carbon dioxide taken in from the day the seeds are planted until the day the plant is fully matured. P is the amount of carbon dioxide each plant is able to scrub when it is fully matured; d is the day number and is what is being varied from day one until the day it is fully matured. G is the number days it takes each plant to fully grow, and A is the growing area (m^2).

Again, this model is far more accurate than that used for the 500-day stay mission. A further analysis on the growth patterns of plants could be taken into account, but this serves as a step in the right direction and produces reasonable values. Table 1-8 summarizes the values of carbon dioxide uptake for each plant given its growing area.

Table 1-8 Carbon Dioxide Uptake Values for a 250-day mission

Plant	Growing Period [d]	CO_2 Uptake[kg]
Carrot	75	24.0
Lettuce	28	25.3
Spinach	30	12.6
Tomato	85	75.4
Wheat	79	487.4
Total	N/A	624.7

As can be seen from Table 1-8, plants can take in approximately 624.7 kg of carbon dioxide over the 250-day stay mission. From before, it is assumed that each astronaut produces approximately 0.998 kg of carbon dioxide per day. Therefore, plants can take in approximately 62.6% of the carbon dioxide produced by the astronauts.

40.1.4.5 Psychological Factors

Again, the psychological factors regarding plant growth are extraordinary. Although there are fewer plants, as well as a smaller variety of plants, the effects of seeing the green color and something living are enormous. Aside from all of the physical characteristics that plants provide for the astronauts, growing plants still has a big psychological factor for the astronauts.

40.1.5 References

[1] Adams, Carsbie C. Space Flight. McGraw-Hill Companies, Inc. 1958

[2] Hanford, Anthony J. Advanced Life Support Baseline Values and Assumptions Document. Houston: Lockheed Martin Space Operations. 2004.

[3] Larson, Wiley J., and Pranke, Linda K. Human Spaceflight: Mission Analysis and Design. New York: The McGraw-Hill Companies, Inc.

40.2 Food Packaging and Preservation Appendix

Author: Timothy Szamborski

Storing food and increasing its shelf life are two main concerns of the overall life support system aboard the MHV. Over time, food begins to ‘break down’. When exposed to air, food begins to bio-degrade and is no longer nutritious nor appealing to eat.

We preserve the food by placing it in vacuum-sealed packaging. This packaging will separate the food from the outside air so that the food has a longer shelf life. Retort pouches serve as these vacuum-sealed packages. This packaging material is used for foods that will be thermally processed at high temperatures. The basic flexible package material will begin with a lamination of an external protective film such as polyester and/or nylon; aluminum foil as the principal oxygen/water vapor/microbial barrier; and cast polypropylene as the internal sealant. Oxygen and water vapor permeability should effectively be zero. This technology is already being used in the Shuttle and ISS food systems as well as for the Military Meals-Ready-to-Eat program (MREs), with a TRL rating of 9 [1].

Another method of preservation is to remove the water content in the food. By removing the water content in the food, the water activity (the water available for chemical reactions) is reduced to a level that slows down or minimizes microbial growth, enzymatic activity and biochemical activity. The water is then added back into the food when the crewmembers are ready to eat. Again, this technology is currently being used on the Shuttle and ISS.

To further increase the shelf life, we store all of the food within a long-term freezer. The cold air within the freezer maintains the cell structure of food and therefore tends to maintain more of the original taste and aroma. Also, by freezing the food, the different molecules within the food will not be able to interact with each other, and thus prevent the structure of the food to break down. The molecules will essentially be 'frozen' and will not have enough energy to interact with other molecules. With proper packaging, these foods will limit microbial growth providing a safe food for an extended period of time (5-20 years) [2].

40.2.1 References

[1] Perchonok, Michele. Advanced Food Technology Workshop Report – Volume I. Houston: Habitability and Environmental Factors Division – NASA. 10 Mar. 2003.

[2] Perchonok, Michele. Advanced Food Technology Workshop Report – Volume II. Houston: Habitability and Environmental Factors Division – NASA. 10 Mar. 2003.

40.3 Water Storage and Recycling Appendix

Author: Timothy Szamborski

Table 1-1 provides usage rates during nominal operation of the life support system.

Table 1-1 Steady-State Values for Vehicle Water Usage

Crewmember Water Allotments	Nominal [kg/CM-d]
Flush Water	0.5[1]
Handwash/Shower	6.8[1]
Hygiene	0.37[1]
Laundry	6.4
Potable Drinking Water	2[2]
Total	16.07

To determine the nominal usage of water needed for the laundry system, the following steps were taken. From the Baseline Values and Assumptions Document, a washer uses approximately 51.3 kg of water per load of laundry. Given the mass of the clothing along with capacity that a washer can contain, 4 flight suits or 41 pairs of undergarments can be washed per load. It is assumed that the undergarments will be changed everyday, and that flight suits will be changed every 4 days. Using this data, Table 1-2 compiles the number of loads conducted per month for the trip

Table 1-2 Number of Laundry Loads per Month for Clothing

Clothing	Loads per month
Underwear	3
Bras	1.5
Flight Suits	10
Total	15

Given that there is 15 load of laundry per month, the total water needed for laundry per crewmember per day was found using the following equation:

$$L = \frac{l \times T_l}{P \times d_M} \qquad (1\text{-}1)$$

where L is the total amount of water needed in laundry per crewmember per day, l is the amount of water needed per load of laundry, T_l is the total number of loads of laundry per month, P is the number of crewmembers, and d_M is the number of days per month. Therefore, Eq. 1-1 becomes:

$$L = \frac{\left(51.3 \ \frac{kg}{load}\right) \times \left(15 \ \frac{loads}{month}\right)}{\left(4 \ CM\right) \times \left(30 \ \frac{days}{month}\right)}$$

$$L = 6.4125 \frac{kg}{CM - d}$$

Table 1-1 tells us that one crewmember will require approximately 16.07 kg of clean water per day. Since we have four crewmembers, to get the daily needs for the entire group, we multiply the total by 4. Therefore, for 4 crewmembers, a total of 64.28 kg per day is needed to meet the crewmembers' daily needs.

Without recycling, the crewmembers need a fresh supply of water for 1,100 days. So multiplying the total water needed among the astronauts for one day by 1,100 days, we estimate a total of 70,708 kg of water. This is the amount of water that would be needed without a method to recycle the water. Since 70,708 kg just for water is not entirely feasible, a method for recycling the water is needed.

An assumption was made that approximately 90 percent of the water is recycled. Using a similar physico-chemical process to recycle water, the European Space Agency (ESA) water recycling system for the Antarctic Concordia Station recycles water with an efficiency of approximately 95 percent [4]. However, we feel that the technology was not yet proven with that efficiency rate for our specific application. Using a conservative estimate to help prove the feasibility of the mission, we assume that the WPA can recycle water with 90 percent efficiency.

In determining how much water is needed to be brought aboard, the following equation was used:

$$T = \sigma + (0.1 \times \sigma) \times d \quad (1\text{-}2)$$

where T (kg) is the total amount of water needed, σ (kg/day) is the water needed per day for the entire crew, and d is the duration of the trip (in days). This equation was developed using the following logic: We need an initial amount of water to start Day 1 of the mission. Then assuming a 10 percent loss in the Water Processing Assembly, we multiply the loss of water for one day, by the total amount of days for the trip. This equates to the total amount of water needed for the trip.

Since σ = 64.28 kg/day and d = 1,100 days, Eq. 1-2 becomes:

$$T = 64.28\ kg + (0.1 \times 64.28\ kg) \times 1{,}100\ \ days$$

$$T = 7{,}135 kg$$

Therefore, the total amount of water needed for the trip is 7,135 kg, using an efficiency rate of 90 percent.

Before recycling can begin, there has to be some water to start with. With a nominal usage rate of 64.28 kg water per day, we estimate a total of 7,135 kg of water is needed for the trip. Contingency Water Containers (CWCs), which can be seen in Fig. 1-1, stores the water. Each container is roughly the size and shape of a duffle bag and can store approximately 40.8 kg of water. So to determine the total number of CWCs needed, the total amount of water needed was divided by the capacity of storage for each container.

$$Q = \frac{T}{C} \qquad (1\text{-}3)$$

Where Q is the number of CWCs needed, T is the total water needed, and C is the capacity of storage for each CWC. The total amount of water needed was previously derived to be 7,135 kg, and each CWC can store approximately 40.8 kg. Therefore, Eq. 1-3 becomes:

$$Q = \frac{7{,}135 \;\; kg}{40.8 \;\; kg}$$

$$Q = 174.9$$

Therefore, a total of 175 CWCs are brought aboard the CTV.

Figure 1-1 Contingency Water Container (Science@NASA)

To determine the volume needed to store the water, the following equation is used:

$$V = \frac{T}{\rho} \qquad (1\text{-}4)$$

where T is the total water needed, ρ is the density of water, and V is the total volume the water takes up in storage. Again, the total amount of water needed is 7,135kg, and the density of water is 1,000 kg/m^3. Therefore, Eq. 1-4 becomes:

$$V = \frac{7{,}135 \;\; kg}{1{,}000 \;\; \frac{kg}{m^3}}$$

$$V = 7.135 \;\; m^3$$

Hence, the total volume needed for the water is approximately 7.1 m^3. However, to account for the extra volume needed for the Contingency Water Containers (CWCs), we estimated the total volume of the water and storage containers to be approximately 8 m^3.

40.3.1 References

[1] Hanford, Anthony J. Advanced Life Support Baseline Values and Assumptions Document. Houston: Lockheed Martin Space Operations. 2004.

[2] Larson, Wiley J., and Pranke, Linda K. Human Spaceflight: Mission Analysis and Design. New York: The McGraw-Hill Companies, Inc.

[3] Science@NASA. "Water on the Space Station." 2 Nov. 2000. NASA. 5 Feb. 2005 <http://science.nasa.gov/headlines/y2000/ast02nov_1.htm>

[4] "Water Recycling Systems for the Antarctic Concordia Station." Antarctic Concordia Station. Jan. 2004. European Space Agency. 8 Feb 2005 <http://esamultimedia.esa.int/docs/TTP-Concordia-Water-Recycling.pdf>

41 Vahle, Mark W.

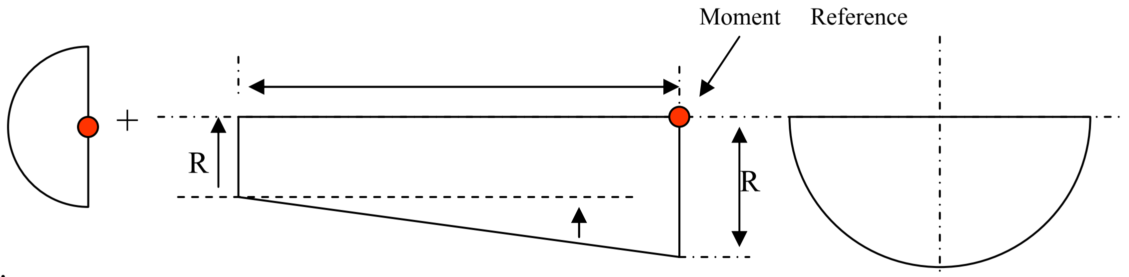
41.1.1 Vahle 1 - Appendix – Descent Aeroshell

Author(s): Mark Vahle

41.1.1.1 Analysis

Our motivation for doing a hypersonic analysis on the Mars Lander Vehicle's (MLV) lifting body is to determine whether the lifting body is actually producing the amount of lift we need to maintain a controlled descent. From previous studies, we found that we need to attain an L/D$_{max}$ of 2 at a reasonable angle of attack. The method we use to analyze the aerodynamic forces on the body is outlined in Clark and Trimmer's Newtonian hypersonic aerodynamics paper. [1]

Our first step is to combine simple shapes and frustra until the shapes look like our lifting body design. Figure 1-4 is the resulting approximation we used for our



analysis.

Figure 1-4 Lifting Body Approximation (Mark Vahle)

Using the equations outlined in Clark and Trimmer's paper, we calculated the normal, axial, and yaw coefficients for the vehicle. The coefficients do not depend on the velocity of the vehicle, but rather the angle of attack and the geometric angles of the shapes. Using MatLab, I wrote a program (aeromars.m) that calculates all the coefficients and divides the lift coefficient by the drag coefficient to yield the L/D as a function of angle of attack (Figure 1-5).

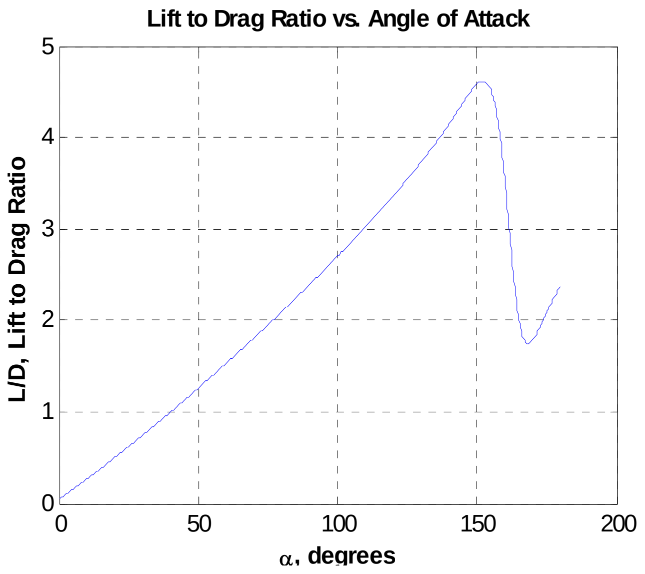


Figure 1-5 Output of aeromars.m (Mark Vahle)

From Figure 1-5 we can see that we reach our goal of L/D = 2 at roughly 75 degrees angle of attack. The analysis proves that we can control the descent of the MLV with our lifting body design.

41.1.1.2 References

1. Clark, E.L. and Trimmer L.L. Equations and Charts for the Hypersonic Aerodynamic Characteristics of Lifting Configurations by the Newtonian Theory. Arnold Engineering Development Center. March 1964.

41.1.1.3 Code

Aeromars.m

```
%Author - Mark Vahle
%AAE 450
%Numerical Integration of Mars entry bodies
clc
close all
clear all

k = 2;     %max cp correction multiplier
alp = linspace(0,180,1000);
alpha = linspace(0,pi,1000);
lambda = [90 80 75 70 65 60]*(pi/180);

%Hemispherical Cap

r = 3;
S1 = 50;   %Area 1 - Surface area of a half sphere

%C = ((k*r^2)./S1);
```

```
for n=1:6
    for k=1:1000
        CN(n,k) = (sin(alpha(k))./2).*(cos(alpha(k)).*sin(lambda(n)).*(pi/2
+atan(cos(lambda(n))./tan(alpha(k)))) + atan(sin(alpha(k)).*tan(lambda(n))));
        CA(n,k) = .25.*(((sin(alpha(k)).^2).*sin(lambda(n))+3.*(cos(alpha(k)).^2).*sin(lambda(n))-
(cos(alpha(k)).^2).*(sin(lambda(n)).^3)).*(pi/2
+atan(cos(lambda(n))./tan(alpha(k))))+2.*cos(alpha(k)).*atan(sin(alpha(k)).*tan(lambda(n)))+sin(alpha(
k)).*cos(alpha(k)).*sin(lambda(n)).*cos(lambda(n)));
    end
end
%
% figure(1)
% plot(alp,CN(1,:),alp,CN(2,:),alp,CN(3,:),alp,CN(4,:),alp,CN(5,:),alp,CN(6,:)),grid
% xlabel('\bf\alpha, degrees')
% ylabel('\bfC_N, Normal Force Coefficient (Lift)')
% title('\bfLift Coefficient vs. Angle of Attack for Spherical-Wedge Blunt Nose Body')
% legend('\bf\Lambda = 90 deg','80 deg','75 deg','70 deg','65 deg','60 deg')
% figure(2)
% plot(alp,CA(1,:),alp,CA(2,:),alp,CA(3,:),alp,CA(4,:),alp,CA(5,:),alp,CA(6,:)),grid
% xlabel('\bf\alpha, degrees')
% ylabel('\bfC_A, Axial Force Coefficient (Drag)')
% title('\bfDrag Coefficient vs. Angle of Attack for Spherical-Wedge Blunt Nose Body')
% legend('\bf\Lambda = 90 deg','80 deg','75 deg','70 deg','65 deg','60 deg')

%LD_1 = CN./CA;


%Waveride Type body

del = [0 8 20 30 40 50]*pi/180;


for n=1:6
    for m=1:1000
        CN2(n,m) =
(pi/2).*cos(alpha(m)).*sin(alpha(m)).*sin(del(n)).*cos(del(n))+(cos(alpha(m)).^2).*(sin(del(n)).^2)+(2
/3).*(sin(alpha(m)).^2).*(cos(del(n)).^2);
        %CN2(n,m) = cos(alpha(m)).*sin(alpha(m)).*sin(del(n)).*cos(del(n)).*(pi/2 +
asin(tan(del(n))./tan(alpha(m))))+((2.*(sin(alpha(m)).^2).*(cos(del(n)).^2)+(sin(del(n)).^2).*(cos(alp
ha(m)).^2))./(3.*sin(alpha(m)).*cos(del(n)))).*sqrt(sin(alpha(m)).^2-sin(del(n)).^2);
        CA2(n,m) =
tan(del(n)).*(2.*cos(alpha(m)).*sin(alpha(m)).*sin(del(n)).*cos(del(n))+(pi/2).*((cos(alpha(m)).^2).*(
sin(del(n)).^2)+.5*((sin(alpha(m)).^2).*(cos(del(n)).^2))));
        %CA2(n,m) =
(tan(del(n))./2).*((2.*(cos(alpha(m)).^2).*(sin(del(n)).^2)+(sin(alpha(m)).^2).*(cos(del(n)).^2)).*(pi
/2 + asin(tan(del(n))./tan(alpha(m)))) + 3.*cos(alpha(m)).*sin(del(n)).*sqrt(sin(alpha(m)).^2-
sin(del(n)).^2));
    end
end

% figure(3)
% plot(alp,CN2(1,:),alp,CN2(2,:),alp,CN2(3,:),alp,CN2(4,:),alp,CN2(5,:),alp,CN2(6,:)),grid
% xlabel('\bf\alpha, degrees')
% ylabel('\bfC_N, Normal Force Coefficient (Lift)')
% title('\bfLift Coefficient vs. Angle of Attack for Half-Cone Frustum')
% legend('\bf\delta = 0 deg','15 deg','20 deg','30 deg','40 deg','50 deg')
% axis([0 120 0 .9])
% figure(4)
% plot(alp,CA2(1,:),alp,CA2(2,:),alp,CA2(3,:),alp,CA2(4,:),alp,CA2(5,:),alp,CA2(6,:)),grid
% xlabel('\bf\alpha, degrees')
% ylabel('\bfC_A, Axial Force Coefficient (Drag)')
% title('\bfDrag Coefficient vs. Angle of Attack for Half-Cone Frustum')
% legend('\bf\delta = 0 deg','15 deg','20 deg','30 deg','40 deg','50 deg')
% axis([0 120 0 1.5])

CNcon = CN2(2,:)+CN(1,:);
CAcon = CA2(2,:)+CA(1,:);
LD_con = CNcon./CAcon;
LD_1 = CN(1,:)./CA(1,:);
```

```
figure(5)
% subplot(311),plot(alp,CNcon),grid
% xlabel('\bf\alpha, degrees')
% ylabel('\bfC_L, Lift Coefficient')
% title('\bfComposite Body Aerodynamic Forces')
%
% subplot(312),plot(alp,CAcon),grid
% xlabel('\bf\alpha, degrees')
% ylabel('\bfC_D, Drag Coefficient')
%
% subplot(313)
plot(alp,LD_con),grid
xlabel('\bf\alpha, degrees')
ylabel('\bfL/D, Lift to Drag Ratio')
```

41.1.2 Vahle 2 – Appendix – Parachute System Failure and Deployment Zones

Author(s): Mark Vahle

Contributor: John Stalbaum

41.1.2.1 Analysis

One of our main concerns for the parachute system is whether a failure in one or more of the parachutes results in a total failure or a mission failure. I used a simplified model to initially calculate the differences in terminal velocity that would result if we lost a third or two-thirds of the parachute surface area. Since the parachute system is based on The program chute.m calculates the terminal velocity of the MLV using its mass and the radius of the parachute. It uses an atmospheric model developed by the NASA Ames Research Center. Figure 6-1 shows us the resulting study that was done using this program.

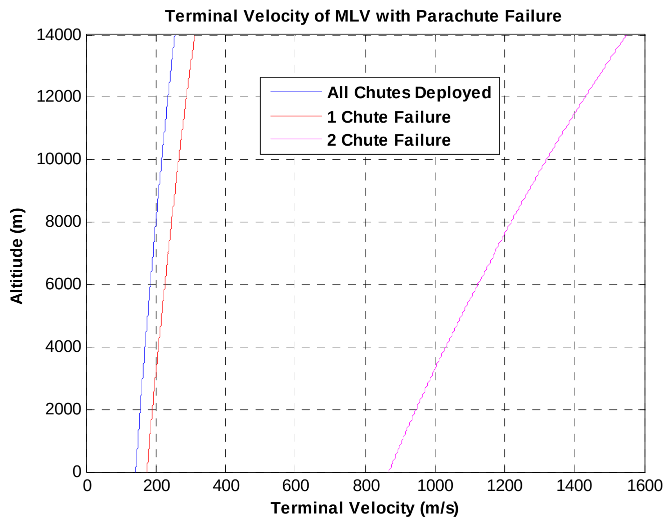


Figure 6-1 Terminal Velocity Difference with Parachute Failure (Mark Vahle)

We can see that it looks close mission requirement if 1 chutes fail, however if 2 parachutes fail the MLV impacts the ground (neglecting drag effects on the vehicle) at 900 m/s.

Using code developed by John Stalbaum, we later reiterated this problem and found that a loss in one or more of the parachutes will not cause a total failure (fatality) but seriously jeopardizes the success of the mission. The reason for this is because we set aside 60 seconds of hover time for the MLV to increase its cross range capability. The loss of a parachute cuts into the cross-range capability but is still able to make a successful landing. The analysis concluded that a failure of one parachute requires an additional 51 m/s of propulsive ΔV which is covered in our cross-range requirement. The landing will occur 7.8 km downrange of the target. A failure in two parachutes will require an additional 171 m/s of propulsive ΔV which is also covered in our cross-range requirement. The landing occurs 18.8 km downrange of the target. A failure of all three chutes, however unlikely, results in an additional 20,000 kg of fuel to land safely. We have designated a loss of three parachutes a total failure scenario.

Parachutes are rated at a maximum dynamic pressure as shown in Equation $q = \frac{1}{2}\rho V^2$ (1-3).

$$q = \frac{1}{2}\rho V^2 \quad \textbf{(1-3)}$$

If the dynamic pressure is higher than the parachutes rating, then the parachute will fail. The maximum dynamic pressure for these parachutes is 702 N/m^2 and the recommend deployment dynamic pressure is 600 N/m^2 [1]. The two dynamic pressures give us a set of deployment zones as a function of altitude and velocity. Figure 1-7 shows us the parachute deployment zones (Output generated by chutefail.m).

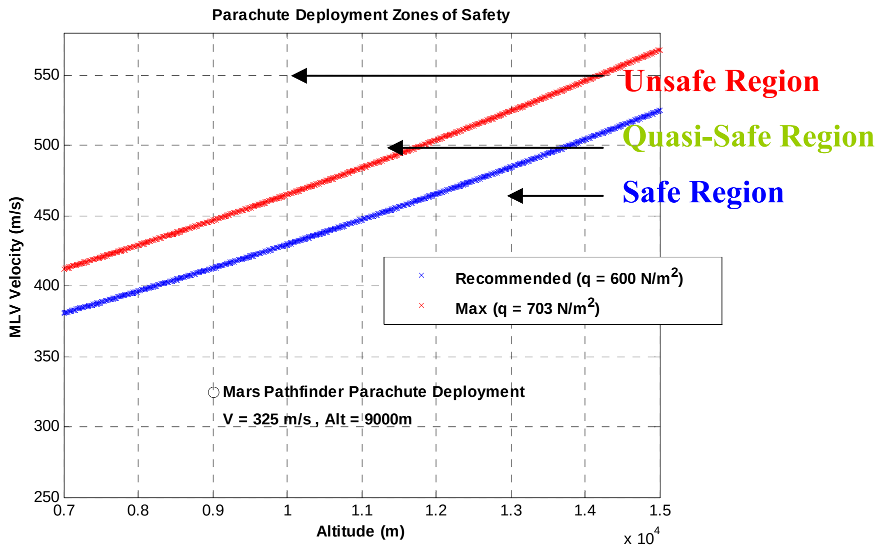


Figure 1-7 Parachute Deployment Zones (Mark Vahle)

NASA has performed thousands of computer simulations on these deployment dynamic pressures [1]. They concluded that if deployed in the recommended (safe) region, the parachute system would have a 99.7% success rate. Therefore, we modeled our entry code to deploy the parachutes once the dynamic pressure reached 600N/m^2.

41.1.2.2 References

1. Braun, R.D. “Mars Pathfinder Atmospheric Entry Navigation Operations.” AIAA Journal. Pg 6.

41.1.2.3 Code

Chute.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                         %
% AAE 450 Senior Design                                                   %
% Author - Mark Vahle                                                     %
% Group - Aerodynamics                                                    %
% Date - 29 Jan 2005                                                      %
%                                                                         %
% Description of Program:                                                 %
% This program calculates the terminal velocity of the MLV given the      %
% lander weight.                                                          %
%                                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
clc

% CHANGEABLE VALUES
```

```
w1 = 36000;     %MLV Mass (kg)
cutoff = 8000;  %Parachute Cutoff altitude (m)
para_rad = 21.2;  %Parachute Radius (m)

% INPUTS

grav_earth = 9.81;  %m/s^2
gm = .39*grav_earth; %m/s^2 Gravity on mars
Cd = .7;   %Drag coefficient for parachute, Calculated Empirically
alt_hi = linspace(7001,14000,1000);     %7000 - 14000m altitude
alt_lo = linspace(0,7000,1000);         %SL - 7000m altitude

% Atmospheric Model
% Source: http://www.lerc.nasa.gov/WWW/K-12/airplane/atmosmrm.html

p_hi = .699.*exp(-.00009*alt_hi);
T_hi = -23.4 -.00222.*alt_hi;       %Celsius
T_lo = -31 - .000998*alt_lo;
p_lo = .699.*exp(-.00009*alt_lo);    %KPa
rho_lo = p_lo./(.1921*(T_lo+273.1)); %kg/m^3
rho_hi = p_hi./(.1921*(T_hi+273.1));

% plot(alt_lo, rho_lo,alt_hi,rho_hi),grid
% xlabel('\bfAltitude (m)')
% ylabel('\bfAtmospheric density (kg/m^3)')

S = pi*(para_rad^2);    %Parachute Area
S1 = [pi*(para_rad^2) pi*(17.39^2) pi*(12.29)];
w = w1*gm;              %Weight of lander (kg)
tv = linspace(190,250, 1000);
for n=1:3
    for k=1:1000
        Vel_t(k,n) = sqrt((2.*w)./(rho_hi(k).*S1(n).*Cd));
        Vel_t2(k,n)= sqrt((2.*w)./(rho_lo(k).*S1(n).*Cd));
    end
end
% V1 = sqrt((2.*w)./(rho_hi.*S.*Cd));
% V2 = sqrt((2.*w)./(rho_lo.*S.*Cd));
% plot(V1, alt_hi, V2, alt_lo)

plot(Vel_t(:,1),alt_hi,'b'),grid;
hold on
plot(Vel_t(:,2),alt_hi,'r')
plot(Vel_t(:,3),alt_hi,'m')
plot(Vel_t2(:,1),alt_lo,'b')
plot(Vel_t2(:,2),alt_lo,'r')
plot(Vel_t2(:,3),alt_lo,'m')
legend('\bfAll Chutes Deployed','\bf1 Chute Failure','\bf2 Chute Failure')
%
%
% h2 = plot(tv,cutoff,'r');
% xlabel('\bfTerminal Velocity (m/s)')
% ylabel('\bfAltitiude (m)')
% title('\bfTerminal Velocity of MLV with Parachute Failure')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mass and Volume Measurement (Kevlar-29 reinforced parachute design)   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rho_para = 50*.45359*35.3147;  %kg/m^3 Pack Density for reentry kevlar parachute (50 lbs/ft^3 *
conversion)
% Source: Peterson, C.W. "Application of Kevlar to Parachute System
% Design." AIAA Journal Paper.

para_rho = .8610;    %kg/m^2 Parachute weight based on area
%Source:Pepper, William. "Aerodynamic Decelerators - A Engineering Review".
%AIAA Journal Paper. pg 5.


Para_w = S*para_rho                %Total Parachute Weight kg
```

```
mass_per = (Para_w/w1)*100;         %Percent of the total mass the parachute takes up
Para_V = Para_w/rho_para;           %Total Parachute Volumn m^3
```

Chutefail.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                        %
% AAE 450 Senior Design                                                  %
% Author - Mark Vahle                                                    %
% Group - Aerodynamics                                                   %
% Date - 13 Feb 2005                                                     %
%                                                                        %
% Description of Program:                                                %
% This program analyzes the failure of the MLV chutes probability        %
%                                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

alt_hi = linspace(7001,15000,1000);     %7000 - 80000m altitude
alt_lo = linspace(0,7000,1000);         %SL - 7000m altitude

% Atmospheric Model
% Source: http://www.lerc.nasa.gov/WWW/K-12/airplane/atmosmrm.html

p_hi = .699.*exp(-.00009*alt_hi);
T_hi = -23.4 -.00222.*alt_hi;       %Celsius
T_lo = -31 - .000998*alt_lo;
p_lo = .699.*exp(-.00009*alt_lo);    %KPa
rho_lo = p_lo./(.1921*(T_lo+273.1)); %kg/m^3
rho_hi = p_hi./(.1921*(T_hi+273.1));

q1 = 600;   %Reccomended N/m^2
q2 = 703;   % Max q N/m^2

V_rec = sqrt((2.*q1)./rho_hi);
V_max = sqrt((2.*q2)./rho_hi);

%Source: Mars pathfinder atmospheric entry navigation operations

plot(alt_hi,V_rec,'bx'),grid
hold on
plot(alt_hi,V_max,'rx')
plot(9000,325,'ko')
ylabel('\bfMLV Velocity (m/s)')
xlabel('\bfAltitude (m)')
title('\bfParachute Deployment Zones of Safety')
legend('\bfRecommended (q = 600 N/m^2)','\bfMax (q = 703 N/m^2)')
axis([7000 15000 250 580])
text(9000,325,'  \bfMars Pathfinder Parachute Deployment')
text(9000,305,'  \bfV = 325 m/s , Alt = 9000m')
```

42 Verbeke, Matthew

42.1 Actual Mars Orbit Rendezvous and Dock

Author: Matt Verbeke

42.1.1 Mars Rendezvous Timing and Docking Technique

Herein we discuss the details for the actual Mars Lander Vehicle (MLV) rendezvous with the Crew Transport Vehicle (CTV) in Martian orbit. This section is identical to section 2.4.2.5 of the main report but is an excellent description of the analysis we completed.

42.1.2 Theory and Assumptions

We make the following assumptions in our analysis which affect the solution to the rendezvous problem:

1. We place the CTV in an elliptic orbit with a 1-Martian day period and periapsis altitude of 350 km.
2. The crew separates the MLV from the CTV at apoapsis and performs a burn to target the Martian atmosphere.
3. Atmospheric entry occurs when the MLV's flight path angle is -7 deg and the altitude is 100 km.
4. The MLV rendezvous with the CTV occurs 1.5 Martian days in excess of the stay time in whole Martian days after separation.
5. After ascending from the Martian surface, the crew achieves a 120 km altitude orbit. We assume that orbit maintenance at this altitude is very costly and therefore immediately perform a Hohmann transfer to a 350 km altitude orbit.
6. The MLV travels in the 350 km altitude circular orbit as long as needed for the rendezvous to occur.
7. A high relative velocity rendezvous is possible provided a sophisticated finite burn scheme can be implemented, although the details of the scheme are beyond the scope of the analysis.

Figure **42-1** depicts the general solution to this problem, note that orbits are not to scale.

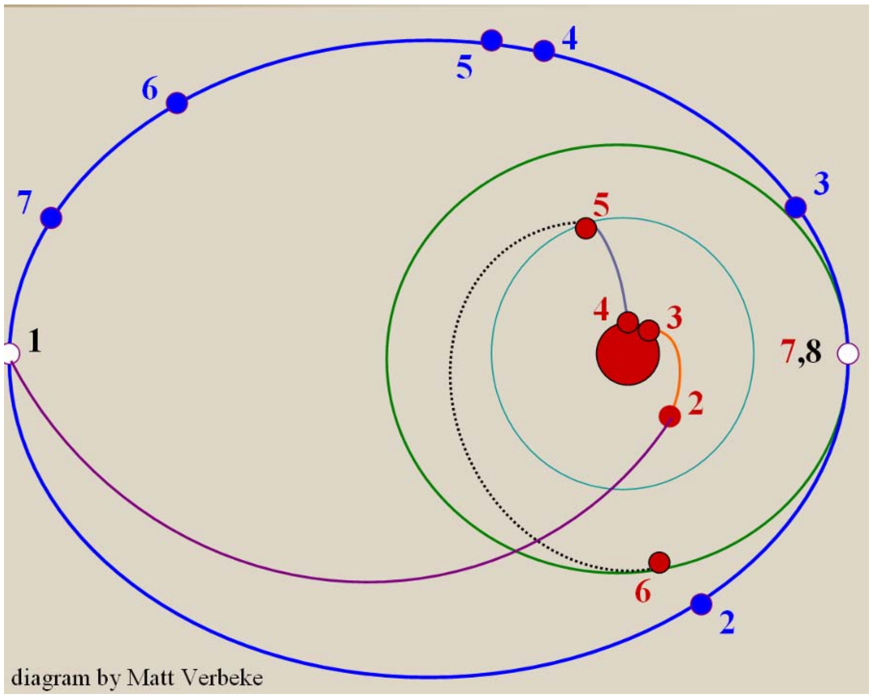


Figure 42-1 General Rendezvous Geometry

In Figure **42-1** the numbers represent the position of each vehicle at a significant event during the MLV's flight. The blue numbers represent the location of the CTV, red numbers represent the location of the MLV, and black numbers represent those locations occupied by both vehicles. Table **42-1** details the significance of each event. Also each trajectory is given a different color for distinction. The blue orbit is the CTV's elliptical parking orbit, purple is the MLV's atmospheric targeting ellipse, orange is the descent trajectory, grey is the ascent trajectory, teal is the 120 km circular orbit, black is the Hohmann transfer, and green is the 350 km circular orbit.

Table 42-1 Significant Events During Rendezvous

Position	Event During Rendezvous
1	Separation
2	Atmospheric Interface
3	Soft Landing on Martian Surface
4	Launch Site
5	1st Hohmann Maneuver Point
6	2nd Hohmann Maneuver Point
7	1st time we reach periapsis (Rendezvous)
8	Rendezvous

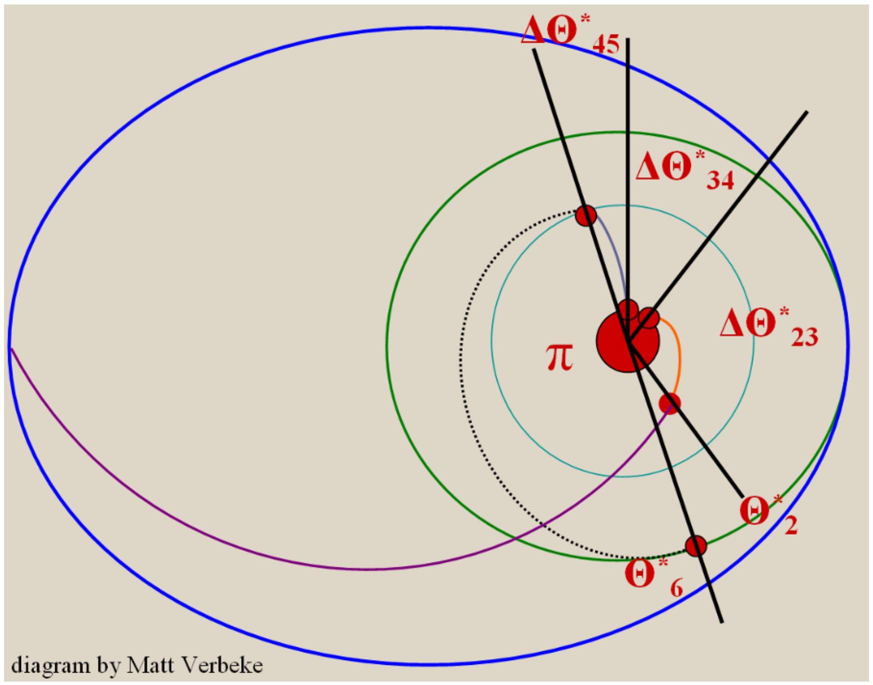


Figure 42-2 Diagram of Relevant Variables

Looking back at assumption four we see that the MLV rendezvous with the CTV must occur 1.5 Martian days in excess of a whole Martian day stay from the time of separation. For this analysis we assume a stay time of zero whole days. Note that once the MLV lands on the Martian surface the respective positions of the MLV and CTV remain fixed. Therefore this analysis is equally valid for a long surface stay mission. Equation (42-1) describes the total time of flight of the MLV to the rendezvous point:

$$T = T_{12} + T_{23} + T_s + T_{45} + T_{56} + \left(1 - \frac{\theta^*_{current}}{2\pi}\right) P_{350} + nP_{350}$$

(42-1)

Where each term indicates the time of flight (TOF) for one of the colored trajectories in Figure **42-1**. T_{12} represents the TOF for the atmospheric targeting ellipse, T_{23} represents the TOF for the descent trajectory, T_s represents the surface stay time, as measured from soft landing, T_{45} represents the TOF for the ascent trajectory, T_{56} represents the TOF for the Hohmann transfer, $\left(1 - \frac{\theta^*_{current}}{2\pi}\right) P_{350}$ represents the initial time to periapsis in the 350 km altitude orbit, and nP_{350} represents the total time for n more revolutions.

To further simplify we look more closely at assumption three which defines the conditions for atmospheric entry. With this assumption the shape of the atmospheric targeting ellipse and point of atmospheric interface on the trajectory are determined; which essentially means we know T_{12}. From similar means we know T_{56} and P_{350} as well. Also T_{23} and T_{45} are known based on the ascent and descent trajectory work done in sections 2.4.2.2 through 2.4.2.4. Now Ts, $\Theta^{*}_{current}$, and n are the only unknown variables in Equation (42-1).However, the initial time to periapsis ($\Theta^{*}_{current}$) is only a function of the true anomaly of the MLV when it enters the 350 km circular orbit. See Equation (42-2).

$$\theta^{*}_{current*} = \theta^{*}_{6} - floor\left(\frac{\theta^{*}_{6}}{2\pi}\right)2\pi$$

(42-2)

The true anomaly is in turn a function of the surface stay time as seen in Equations (42-3) and (42-4).

$$\theta^{*}_{6} = \theta^{*}_{2} + \Delta\theta^{*}_{23} + \Delta\theta^{*}_{34} + \Delta\theta^{*}_{45} + \pi$$

(42-3)

$$\Delta\theta^{*}_{34} = \Omega_{Mars}\left[\frac{T_s}{24.6229} - floor\left(\frac{T_s}{24.6229}\right)\right]24.6229$$

(42-4)

Therefore the number of complete revolutions and the Martian surface stay time determine the solutions to the rendezvous problem. Figure 42-2 illustrates the physical meaning of some of the variables found in Equations (42-2) and (42-3).

42.1.3 Analysis and Results

Now that we know the number of complete revolutions and surface stay time determine if a rendezvous occurs, we write a simple code to find the solutions. We expect to find a surface stay time for each value of n such that a rendezvous occurs. Therefore the code chooses a value of n and calculates the total time to rendezvous, T from Equation (42-1), for all the T_s values ranging from zero to one Martian day. The code then increases the number of complete revolutions by one and reiterates. We present the result for the one revolution case in graphical form as Figure 42-3.

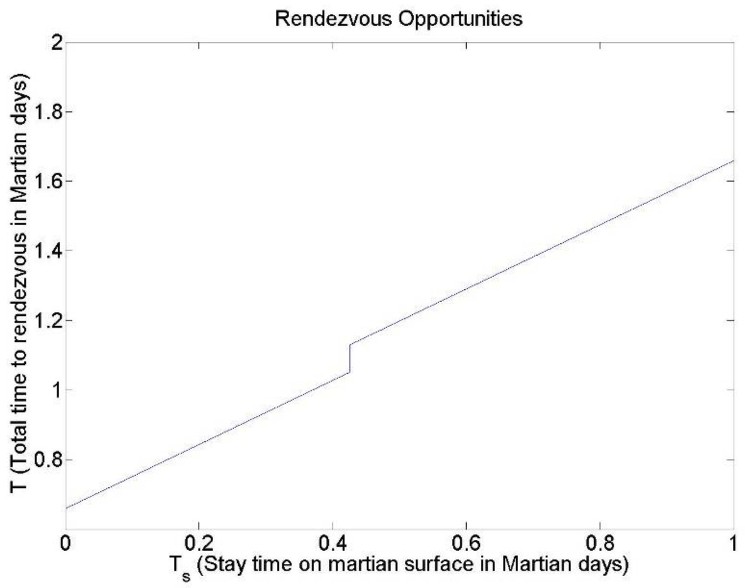


Figure 42-3 T vs. T_s - An indication of rendezvous opportunities

The code then extracts the value of T_s where T equals 1.5 Martian days and gives the value as the solution to the rendezvous problem. In this analysis we find there are 10 opportunities for the rendezvous to occur. See
Table 42-2.

Table 42-2

Number of Revolution "n"	Martian Surface Stay [fraction of Mars day from touchdown]
0	0.913
1	0.828
2	0.742
3	0.657
4	0.572
5	0.486
6	-
7	0.401
8	0.316
9	0.231
10	0.145
11	0.060

12	-

We choose the 1 revolution case as the nominal rendezvous. By doing so, we allow for one launch slip and avoid a long duration in orbit. Since we have essentially no radiation protection on the MLV, reducing the time in orbit reduces the total exposure of the crew during this period. Finally the rendezvous scheme requires four in-space maneuvers which are the same for any of the various options. We quantify these maneuvers in Table **42-3**.

Table 42-3

Maneuver Number	Location of Burn	ΔV [m/s]
1	1	18
2	5	55
3	6	54
4	8 (or 7)	1176

42.1.4 Computer Code

We present the computer code below which contains the details of the Martian orbit rendezvous scheme employed. The inputs are the time of flights and angular displacements for the descent and ascent phases and atmospheric entry conditions. The outputs are plots illustrating the total time to rendezvous for all values of n, as well as the vector row containing the Martian surface stay times, as measured from soft landing, for each possible value of n, the number of revolutions in the 350 km orbit, which result in a rendezvous of the two bodies. It is important to note that the vector value determines the validity of a solution based on proximity to zero. For example entry 7 of value equals 0.02. Since this value is much farther from zero than the other cases a rendezvous does not occur for this number of revolutions. It is also important to note that the first entry of row corresponds to the zero complete revolution case. For more clarity we suggest running the code in MATLAB.

```
close all; clear all; clc;

%Runs George's script to find entry ellipse
gamma_entry = -7;alt_entry = 100;entry_interface;
mars_day = 24.6229*3600; %[s]
w_mars = 2*pi/mars_day; %[rads/s]

%Inputs
theta_e = 37.09*pi/180; % Angular displacement during entry from John's code
theta_a = 5.3*pi/180; %Angular displacement during ascent from John's code

%Hohmann Transfer
rp_r = R_mars+120;
ra_r = R_mars+350;
```

```
a_r = 0.5*(rp_r+ra_r);
e_r = 1-rp_r/a_r;
p_r = a_r*(1-e_r^2);
Pr = 2*pi*sqrt(a_r^3/mu_mars);

%Known TOF's
dt_e = 1255.1;  %TOF entry [s] from John's code
dt_a = 286;   %TOF ascent [s]
dt_hoh = pi*sqrt(a_r^3/mu_mars); %TOF Hohmann transfer [s]

%Bodies' position
rl = [r_a_e alt_entry+R_mars R_mars R_mars R_mars+120 R_mars+350 R_mars+350 R_mars+350];

%MLV True and Eccentric anomaly
TSL = [pi 0];
EL = zeros(1,8);
TOFL = zeros(1,7);

%Periapsis Conditions
EL(1) = acos((a_e-rl(1))/(a_e*e_e));
TSL(2) = 2*pi-acos(1/e_e*(p_e/rl(2)-1)); %@ Atm. Int.
EL(2) = 2*pi-acos((a_e-rl(2))/(a_e*e_e));
TOFL(1) = sqrt(a_e^3/mu_mars)*(EL(2)-EL(1)+e_e*(sin(EL(1))-sin(EL(2))));

crvue = 2*pi*sqrt(rl(end)^3/mu_mars);
onplanet = linspace(0,P,1e5);%P/(1e5-1) increments;
fstay = onplanet/mars_day-floor(onplanet/mars_day);
thetaonplanet = w_mars*fstay*mars_day;
landersix = TSL(2)+theta_e+thetaonplanet+theta_a+pi;
nowat = landersix-floor(landersix/(2*pi))*2*pi;
for z = 1:13
    orbits = [0:12];
    orbits = orbits(z);
    lander_tof = TOFL(1)+dt_e+onplanet+dt_a+dt_hoh+(1-nowat/(2*pi))*crvue+orbits*crvue;
    [v r] = min(abs(1.5-lander_tof./P));
    value(z) = v;
    row(z) = r*(1/(1e5-1));
    figure (1999+z)
    plot(onplanet/P,lander_tof./P)
    xlabel('T_s (Stay time on martian surface in Martian days)')
    ylabel('T (Total time to rendezvous in Martian days)')
    title('Rendezvous Opportunities')
end
row
```

42.2 Fuel Cell Sizing for the MLV

Author: Matt Verbeke

42.2.1 Powering the MLV with Fuel Cells

Within this section of the appendix we explain the sizing of the Mars Lander Vehicle fuel cell. We wrote a code early in the project to make future iterations quick and easy. The inputs to the code are the total power required in Watts, the duration for which we need power in seconds, and an efficiency factor. With these inputs, the code outputs a total mass of reactants, liquid hydrogen and oxygen, as well as the total mass of water produced in kilograms.

42.2.2 Theory and Assumptions

After much research we choose an alkaline fuel cell. We base our decision mainly on historical precedence and the fact that the Space Shuttle's power comes from this type of fuel cell. Also the alkaline fuel cell produces drinkable water as a byproduct. Since the fuel cell produces excess water we use some of it as a heat sink for cooling the MLV.

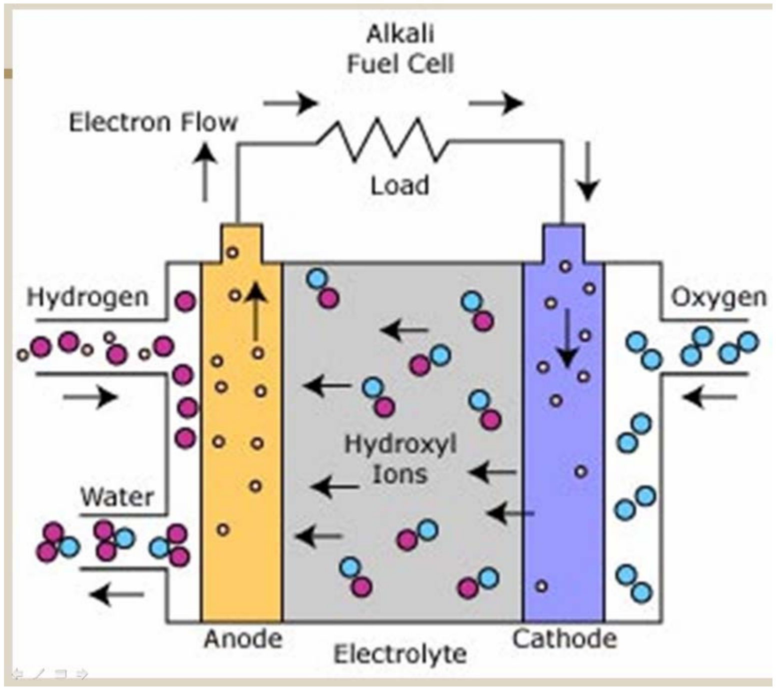


Figure 42-4 Generic Fuel Cell (fctec.com)

An alkaline fuel cell consists of three main components, the anode, the cathode and the electrolyte. The anode and cathode sandwich the electrolyte much like in Figure 42-4. Figure 42-4 also depicts the flow of the reactants and electricity into and out of the fuel cell.

The fuel cell sizing code mainly follows the assumptions of Fuel Cell Systems Explained by Larmine and Dicks [1]. As outlined in their book, we manipulate chemical formulas to size the overall fuel cell. Two main reactions occur inside of the fuel cell. First at the anode hydrogen combines with hydroxyl ions to form water and electrons. When given a path, the electrons can flow through a load and return to the cathode where they combine with oxygen and water to reform hydroxyl ions.

42.2.3 Analysis and Results

We base the mass of the fuel cell and its accessories on the Space Shuttle's fuel cell [2]. The total mass of the system is 105 kilograms. This mass includes 95 kilograms for the fuel cell stack and accessories and 10 kilograms for the reactant lines and voltage conversion hardware. The fuel cell stack is actually two sets of 32 individual cells connected in series which each produce 6 kilowatts of power. Of the 6 kilowatts produced 900 Watts are unusable due to the power requirements of the voltage conversion hardware. We should note that each of the 32 cell substacks outputs 28 volts direct current and 6 kW of power, though only one runs at a time. Having two substacks provides redundancy and also enables us to share the load which helps maintain a constant voltage output. Just like on the Shuttle, we purge our fuel cells twice a day to reduce contaminant buildup on the anode and cathode when active.

The fuel cell also requires 42 kilograms of liquid hydrogen (LH) and 167 kilograms of liquid oxygen (LOX). These reactants add little to the mass of the overall vehicle since we also use LOX and LH as the main propulsive reactants. Storage tanks for the consumables are also accounted for by propulsion and a water storage tank is provided by the thermal subsystem.

[1] Larmine, J., Dicks, A. Fuel Cell Systems Explained – Second Edition

Chichester, West Sussex : J. Wiley, c2003.

[2] Petty, John. Electrical Power System. 7 Apr. 2002. National Aeronautics and Space Administration. <http://spaceflight.nasa.gov/shuttle/reference/shutref/orbiter/eps>

42.2.4 Computer Code

```
close all; clear all; clc;

%%%%%%%%%%%%%%%%%%Constants%%%%%%%%%%%%%%%%%%
f = 96485.3415; %[c]

omw = 32e-3; % O2 [kg/mole]
hmw = 2.02e-3; % H2 [kg/mole]
watermw = 18.02e-3; % H2O [kg/mole]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%Inputs%%%%%%%%%%%%%%%%%%%%%
pt = 6000; %[W] Power required
trip = 74*60*60; %[sec] duration
eff = .875/1.48; %efficiency
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
vc = 1.48*eff; %Voltage of a single cell
omdot = pt/(4*f*vc)*omw; %[kg/s]
hmdot = pt/(2*f*vc)*hmw; %[kg/s]
watermdot = pt/(2*f*vc)*watermw; %[kg/s]

o_used = omdot*trip*1.1 % 10% Margin
h_used = hmdot*trip*1.1 % 10% Margin
prop_m = o_used+h_used
waste_heat = pt*(1.48/vc-1)

%For increasing the volume of the tanks
%Must make sure that they can still withstand
%the pressure for a given propellant amount.
%
%Note : This part of code has become nearly obsolete!
%
al_den = 2840; %kg/m^3
in_den = 0; %kg/m^3

p_wtank = 1.04e5;
p_htank = 2025679.6066; %[N/m^2] or 293.8 psia
p_otank = 6791335.645; %[N/m^2 or 985 psia

wpropm_store = prop_m; %Store 3.5 days of water
wrho_store = 1000; %[kg/m^3]
wtank_vol = wpropm_store/wrho_store;
wtank_i = (3*wtank_vol/(4*pi))^(1/3);
wt = 0.01; %Initial thickness guess

% otank_o = 0.93472; %H2 tank outer shell [m] dry mass 75.021586 kg Aluminum 2219
% otank_i = 0.849122; %At -285F holds 291.501785 kg O2  Inconel 718
orho_store = 0;
otank_vol = o_used/orho_store;
otank_i = (3*otank_vol/(4*pi))^(1/3);
ot = 0.01;  %Initial thickness guess

% htank_o = 1.15316; %At -420F holds 34.338238 kg H2 dry mass 80.620212 kg Aluminum 2219
% htank_i = 1.054354;
hrho_store = 0;
htank_vol = h_used/hrho_store;
htank_i = (3*htank_vol/(4*pi))^(1/3);
ht = 0.01;  %Initial thickness guess

w_stress = 80e10;
h_stress = 0;
o_stress = 0;
hyield = 0;    %Yield stress of Al 2219 @ -420F
oyield = 0;    %Yield stress of Al 2219 @ -285F
wyield = 75.8e6*1.2; %Yield stress of Al 2219 @ 72F
```

```
    while w_stress > wyield
        wt = wt + 0.001;
        w_stress = p_wtank*wtank_i/(2*wt);
    end

    while h_stress > hyield
        ht = ht + 0.01;
        h_stress = p_htank*htank_i/(2*ht);
    end

    while o_stress > oyield
        ot = ot + 0.01;
        o_stress = p_otank*otank_i/(2*ot);
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    wtmass = al_den*4/3*pi*((wtank_i+wt)^3-wtank_i^3)
    htmass = al_den*4/3*pi*((htank_i+ht)^3-htank_i^3);
    otmass = al_den*4/3*pi*((otank_i+ot)^3-otank_i^3);
```

42.3 Original Rendezvous Scheme

Author: Matt Verbeke

42.3.1 Elliptical Rendezvous Scheme

In the original rendezvous scheme, we launch the MLV into a 120 km altitude orbit and perform a maneuver to place it on an elliptic orbit with a period equal to that of the CTV. We attempt to find a solution where the rendezvous occurs near the apoapsis of the CTV orbit. We make the period of the CTV orbit and MLV orbit equal to allow multiple chances for a rendezvous. If possible, this scheme results in great mass savings in terms of fuel. As it turns out, a rendezvous can not occur near the apoapsis, but instead can only take place near the periapsis. Rendezvous at the periapsis, coupled with the large delta v to initially get on the transfer ellipse, causes this rendezvous scheme to be impractical. If we can change the CTV parking orbit period, a more practical solution may be found, however fewer launch opportunities exist.

42.3.2 Theory and Assumptions

We make the following assumptions in our analysis which affect the solution to the original rendezvous problem:

8. We place the CTV in an elliptic orbit with a 1-Martian day period and periapsis altitude of 350 km.
9. The crew separates the MLV from the CTV at apoapsis and performs a burn to target the Martian atmosphere.
10. Atmospheric entry occurs when the MLVs flight path angle is -7 deg and the altitude is 100 km.
11. After ascending from the Martian surface, the crew achieves a 120 km altitude orbit. We immediately perform a maneuver which places the CTV on an elliptic transfer orbit with a one Martian day period.

Figure **42-1** depicts the general solution to this problem.

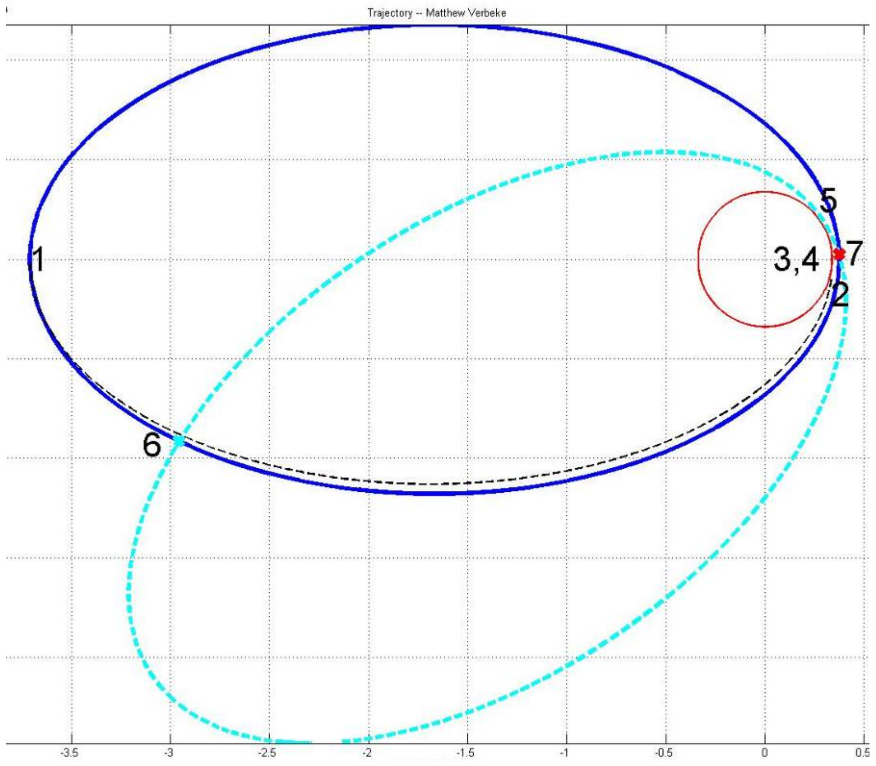


Figure 42-5 General Rendezvous Geometry

In Figure **42-1** the numbers represent the position of the MLV at a significant event in the rendezvous scheme. Table **42-1** specifies the significance of each event. Also the CTV's parking orbit is the large solid trajectory and the MLV rendezvous ellipse is the large dashed trajectory. Note that the intersections of the ellipses indicate possible points of rendezvous.

Table 42-4 Significant Events During Rendezvous

Position	Event During Rendezvous
1	Separation
2	Atmospheric Interface
3	Soft Landing on Martian Surface
4	Launch Site
5	Maneuver Point
6	1st Rendezvous Opportunity
7	2nd Rendezvous Opportunity

In this case the stay time gives a geometry where the first possible rendezvous point lies far from the apoapsis of the CTV orbit and the second opportunity lies near the periapsis. These encounters require large delta v's to accomplish the rendezvous and are therefore unfavorable. In this analysis changing the surface stay time changes the orientation of the MLV's rendezvous ellipse. Essentially the MLV transfer orbit's periapsis will precess about Mars. Figure 42-6 helps visualize the effect by illustrating three different surface stay times.

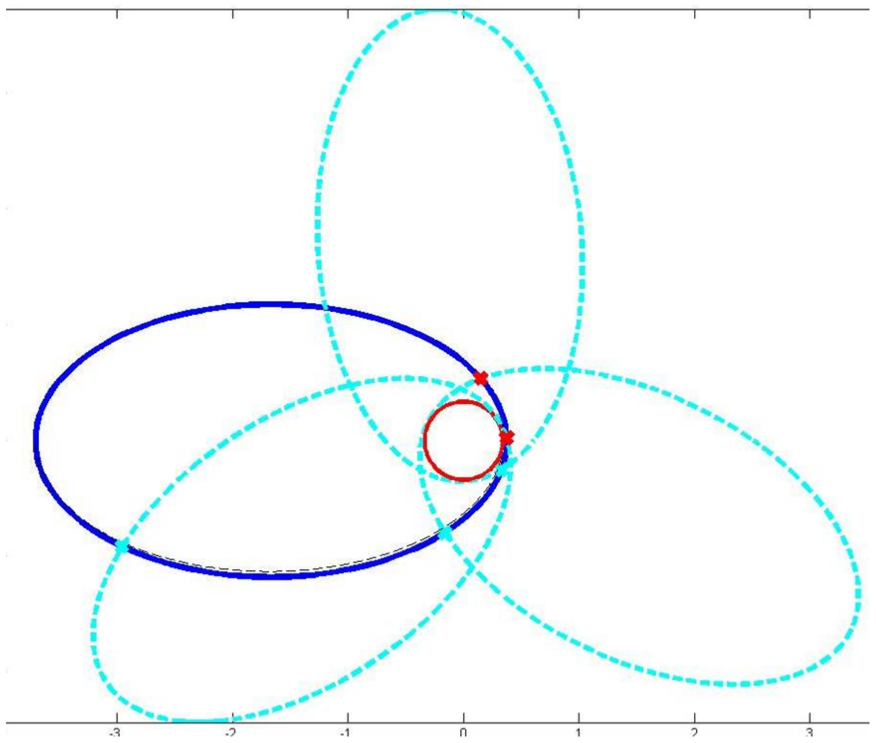


Figure 42-6 Affect of Stay time of Rendezvous Ellipse

42.3.3 Future Analysis

We do not present results for the original rendezvous scheme since a flaw exists in the computer code. Instead we describe the bug so that it may be dealt with in the future. To determine if a rendezvous occurs we keep track of the position of each vehicle and the time each vehicle takes to get to each position. The time of flight (TOF) between segments and the position of the MLV in the various trajectories are known ahead of time. We use the known MLV time of flights, and the shape of the CTV's orbit, to determine the CTV's position up to the point where the MLV performs its large delta v maneuver. From Figure 42-1 we can see that the rendezvous true anomaly values for the CTV and MLV are fixed for a given surface stay time. We also know the location of both vehicles before the possible rendezvous (at step 5). From this information we can find the time it takes each vehicle to get to the rendezvous point. If the times match a rendezvous occurs.

The code depends greatly on the ability to calculate the CTV's position for a given time of flight. Kepler's equation cannot be solved algebraically for the position of a body after some elapsed time. Therefore we iterate to converge on the future true anomaly. In some cases the iteration does not converge which gives inaccurate results. We did not discover this bug until the current rendezvous scheme was completed. We present the code below for reference.

42.3.4 Computer Code

```
close all; clear all; clc;

gamma_entry = -7;alt_entry = 100;entry_interface;
mars_day = 24.6229*3600; %[s]
w_mars = 2*pi/mars_day; %[rads/s]

theta_e = 35*pi/180;
theta_a = 14.4637*pi/180;

z = 0;
check = 0;
lmnop = 0;
miss_time_min = mars_day;

%while check == 0;
for trxinc = 0;%Increments the period for the rendezvous ellipse
    for day_inc = 0;%Increments the stay on the planet
        stay = day_inc*mars_day; %[s]
        fstay = stay/mars_day-floor(stay/mars_day);
        theta_stotal = w_mars*stay;
        theta_s = w_mars*fstay*mars_day; %Angular displacement [rad]

        dt_e = 1511.9;  %TOF entry [s]
        dt_a = 754.6;   %TOF ascent [s]

        rl = [r_a_e alt_entry+R_mars 0 0 0];
        rc = [r_a 0 0 0 0 0];

        TSC = [pi 0 0 0 0];
        EC = [0 0 0 0 0];

        TSL = [pi 0 0 0 0];
        EL = [0 0 0 0 0];

        TOFL = [0 0 0 0 0];
        TOFC = [0 0 0 0 0];

        %Periapsis Cond.
        EL(1) = acos((a_e-rl(1))/(a_e*e_e));
        EC(1) = acos((a_po-rc(1))/(a_po*e_po));

        %Lander @ Atm. Interface
        TSL(2) = 2*pi-acos(1/e_e*(p_e/rl(2)-1)); %Descending for entry
        EL(2) = 2*pi-acos((a_e-rl(2))/(a_e*e_e));
        TOFL(1) = sqrt(a_e^3/mu_mars)*(EL(2)-EL(1)+e_e*(sin(EL(1))-sin(EL(2))));

        %Lander @ Landing
        TSL(3) = TSL(2)+theta_e - floor((TSL(2)+theta_e)/(2*pi))*2*pi;
        TSL(4) = TSL(3)+theta_s - floor((TSL(3)+theta_s)/(2*pi))*2*pi;
        TSL(5) = TSL(4)+theta_a - floor((TSL(4)+theta_a)/(2*pi))*2*pi;

        TOFL = [TOFL(1) dt_e stay dt_a]; %TOFs prior to rendevous
        time = [TOFL(1)+P/2 0 0 0]; %Absolute time from periapsis prior to seperation

        %Find Position of CTV throughout time sequence
        for i=1:length(TOFL);
            if i > 1
                time(i) = time(i-1)+TOFL(i);
            end
            tmtp = time(i)-floor(time(i)/P)*P; %Time since periapsis
            M = sqrt(mu_mars/a_po^3)*tmtp;
            EC_2 = 0;
            %Finds future value of E based on time since periapsis at
            %future point
            for k = 1:100
                EC_2 = EC_2-(EC_2-e_po*sin(EC_2)-M)/(1-e_po*(cos(EC_2)));
            end
            EC(i+1) = EC_2;

            if EC(i) < 0
                if EC(i+1) > 0 | EC(i) < EC(i+1)
                    EC(i) = EC(i)+2*pi;
                    EC(i+1) = EC(i+1)+2*pi;
                elseif EC(i+1) < 0 & EC(i+1) < EC(i)
                    EC(i) = EC(i)+2*pi;
                    EC(i+1) = EC(i+1)+4*pi;
                end
            elseif EC(i) > 0
                if EC(i+1) < 0 | EC(i+1) < EC(i)
                    EC(i+1) = EC(i+1)+2*pi;
                end
            end

            TSC(i+1) = 2*atan(((1+e_po)/(1-e_po))^.5*tan(EC(i+1)/2));
            TOFC(i) = sqrt(a_po^3/mu_mars)*(EC(i+1)-EC(i)+e_po*(sin(EC(i))-sin(EC(i+1))));
        end
        rsc = (a_po*(1-e_po^2))./(1+e_po*cos(TSC));
        rsl = p_e./(1+e_e*cos(TSL));
        rsl(end) = 120+R_mars;

        %Rendevous Ellipse
        rp_r = rsl(end);
        %trxinc = 3600*-.2;
        Pr = P+trxinc;
        a_r = ((Pr/(2*pi))^2*mu_mars)^(1/3);
        e_r = 1-rp_r/a_r;
        p_r = a_r*(1-e_r^2);

        %Not necessarily rendevousing at apoapsis. Need to find EC(6)
        %first! -- left was assuming I was meeting at apoapsis which is
        %EC(1)

        %left = sqrt(a_po^3/mu_mars)*(EC(1)-EC(5)+e_po*(sin(EC(5))-sin(EC(1))));
        %vector = [rsl(end)*cos(TSL(end)) rsl(end)*sin(TSL(end)) 0;r_a*cos(pi) r_a*sin(pi) 0];
        %TOF_OL = left;
        %mu_body = mu_mars;
        %[sma e p] = lambert(vector,TOF_OL,mu_body)
        %[a_rt e_rt p_rt]=lambert(vector,TOF_OL,mu_body);

        t = linspace(0,2*pi,40000);
        trl = linspace(-TSL(end),-TSL(end)+2*pi,40000); %Starts rr vector in the same inertial
position as r vector
        t1 = linspace(TSL(1),TSL(2),1000);

        r = (a_po*(1-e_po^2))./(1+e_po*cos(t)); %r for whole CTV orbit
        rlander = p_e./(1+e_e*cos(t1)); %r for the incoming descent traj
        rr = (p_r)./(1+e_r*cos(trl)); %r for the rendevous ellipse
%%%%%% Useful visualization plots %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         figure(1)
%         plot(R_mars*cos(t),R_mars*sin(t),'r-');
%         axis equal;
%         grid on;
%         xlabel('Y - Distance [km]')
%         ylabel('X - Distance [km]')
%         title('Trajectory -- Matthew Verbeke')
%         hold on
%
%         plot(r.*cos(t),r.*sin(t),'b-')
%         plot(rlander.*cos(t1),rlander.*sin(t1),'k--')
%         text(rsc(1)*cos(TSC(1)),rsc(1)*sin(TSC(1)),'1')
%         text(rsc(2)*cos(TSC(2)),rsc(2)*sin(TSC(2)),'2')
%         text(rsc(3)*cos(TSC(3)),rsc(3)*sin(TSC(3)),'3')
%         text(rsc(4)*cos(TSC(4)),rsc(4)*sin(TSC(4)),'4')
%         text(rsc(5)*cos(TSC(5)),rsc(5)*sin(TSC(5)),'5')
%
%         text(rsl(1).*cos(TSL(1)),rsl(1).*sin(TSL(1)),'Seperation')
%         text(rsl(2).*cos(TSL(2)),rsl(2).*sin(TSL(2)),'Atm. Int.')
%         text(R_mars*cos(TSL(3)),R_mars.*sin(TSL(3)),'Landing')
%         text(R_mars*cos(TSL(4)),R_mars.*sin(TSL(4)),'Launch')
%         text(rsl(5).*cos(TSL(5)),rsl(5).*sin(TSL(5)),'Man. Pt.')
%
%         plot(rr.*cos(t),rr.*sin(t),'c--')

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Explore 1st intersection point
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        [dr ind] = min(abs(r(1:length(r)*.6)-rr(1:length(r)*.6)));
        if ind > 1
            diff1 = rr(ind-1)-r(ind-1);
        else
            diff1 = 100;
        end
        diff2 = rr(ind)-r(ind);

        if abs(diff1) < abs(diff2)
            col = ind-1;
        else
            col = ind;
        end

        %plot(rr(col)*cos(t(col)),rr(col)*sin(t(col)),'rx','LineWidth',3)%Red x at 1st intersection
        if col == 1
            SL = 1;
            SC = 1;
        else
            SL = sign(rr(col)-rr(col-1)); %Determines ascending or descending
            SC = sign(r(col)-r(col-1));
        end

        vintl = sqrt(2*mu_mars/rr(col)-mu_mars/a_r);
        vintc = sqrt(2*mu_mars/r(col)-mu_mars/a_po);
        gammal = SL*acos(sqrt(mu_mars*p_r)/(rr(col)*vintl));
        gammac = SC*acos(sqrt(mu_mars*a_po*(1-e_po^2))/(r(col)*vintc));

        if sign(gammal)==sign(gammac)
            dgamma = abs(gammal-gammac);
        else
            dgamma = abs(gammal)+abs(gammac);
        end

        dvint = sqrt(vintl^2+vintc^2-2*vintl*vintc*cos(dgamma));

        tstar_rl = SL*acos(1/e_r*((p_r/rr(col))-1));
        TSL(6) = tstar_rl;
        E_rl = 2*atan(((1+e_r)/(1-e_r))^-.5*tan((tstar_rl)/2));

        tstar_rc = SC*acos(1/e_po*((a_po*(1-e_po^2)/r(col))-1));
        TSC(6) = tstar_rc;
        E_rc = 2*atan(((1+e_po)/(1-e_po))^-.5*tan(tstar_rc/2));

        if E_rl < 0
            E_rl = E_rl+2*pi;
        end

        if E_rc < 0 & E_rc < EC(end)
            E_rc = E_rc+floor(EC(end)/(2*pi))*6*pi;
        elseif E_rc < 0 | E_rc < EC(end)
            E_rc = E_rc+floor(EC(end)/(2*pi))*4*pi;
        end

        tof_rl = sqrt(a_po^3/mu_mars)*(E_rl-EL(end)+e_r*(sin(EL(end))-sin(E_rl)));
        tof_rc = sqrt(a_po^3/mu_mars)*(E_rc-EC(end)+e_po*(sin(EC(end))-sin(E_rc)));
        if tof_rl > Pr
            tof_rl = tof_rl-Pr;
        elseif tof_rc > Pr
            tof_rc = tof_rc-Pr;
        end

        miss_time = tof_rl - tof_rc;
        check = (abs(tof_rl - tof_rc) < 60*5);
        z = 1;
        if abs(miss_time) < miss_time_min
            porm = sign(miss_time);
            miss_time_min = abs(miss_time);
            d_min = day_inc;
            z_min = z;
        end
        if check == 1 & sign(miss_time)<0 %Within 5 mins & Lander ahead of CTV
            lmnop = lmnop+1;
            opps(lmnop) = day_inc;
            zopps(lmnop) = z;
            dvopps(lmnop) = dvint;
            trx(lmnop) = trxinc;
        end
        if tof_rl<0|tof_rc<0
            check = 1;
            display('Something is wrong')
            day_inc;
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Explore 2nd intersection point
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if check == 0
            ahead = 5e3;
            skip = ind+ahead;
            [dr2 ind2] = min(abs(r(skip:end)-rr(skip:end)));
            begin = skip;
            if skip+ind2 > length(r);
                disp('In loop')
                begin = .4*length(r);
                finish = .8*length(r);
                [dr2 ind2] = min(abs(r(begin:finish)-rr(begin:finish)));
            end

            diff3 = rr(ind2+begin)-r(ind2+begin);
            diff4 = rr(ind2+begin-1)-r(ind2+begin-1);
            if abs(diff3) < abs(diff4)
                col = ind2+begin;
            else
                col = ind2+begin-1;
            end

            %plot(rr(col)*cos(t(col)),rr(col)*sin(t(col)),'cx','LineWidth',3)%Cyan x at 2nd
intersection point
            if col == 1
                SL = 1;
                SC = 1;
            else
                SL = sign(rr(col)-rr(col-1)); %Determines ascending or descending
                SC = sign(r(col)-r(col-1));
            end

            vintl = sqrt(2*mu_mars/rr(col)-mu_mars/a_r);
            vintc = sqrt(2*mu_mars/r(col)-mu_mars/a_po);
            gammal = SL*acos(sqrt(mu_mars*p_r)/(rr(col)*vintl));
            gammac = SC*acos(sqrt(mu_mars*a_po*(1-e_po^2))/(r(col)*vintc));

            if sign(gammal)==sign(gammac)
                dgamma = abs(gammal-gammac);
            else
                dgamma = abs(gammal)+abs(gammac);
            end

            dvint = sqrt(vintl^2+vintc^2-2*vintl*vintc*cos(dgamma));

            tstar_rl = SL*acos(1/e_r*((p_r/rr(col))-1))+TSL(5);
            E_rl = 2*atan(((1+e_r)/(1-e_r))^-.5*tan(tstar_rl/2));

            tstar_rc = SC*acos(1/e_po*((a_po*(1-e_po^2)/r(col))-1));
            E_rc = 2*atan(((1+e_po)/(1-e_po))^-.5*tan(tstar_rc/2));

            if E_rl < 0
                E_rl = E_rl+2*pi;
            end

            if E_rc < 0 & E_rc < EC(end)
                E_rc = E_rc+floor(EC(end)/(2*pi))*6*pi;
            elseif E_rc < 0 | E_rc < EC(end)
                E_rc = E_rc+floor(EC(end)/(2*pi))*4*pi;
            end

            tof_rl = sqrt(a_po^3/mu_mars)*(E_rl-EL(end)+e_r*(sin(EL(end))-sin(E_rl)));
            tof_rc = sqrt(a_po^3/mu_mars)*(E_rc-EC(end)+e_po*(sin(EC(end))-sin(E_rc)));

            if tof_rl > Pr
                tof_rl = tof_rl-Pr;
            elseif tof_rc > Pr
                tof_rc = tof_rc-Pr;
            end

            miss_time = tof_rl - tof_rc;
            check = (abs(tof_rl - tof_rc) < 60*5);
            z = 2;
            if abs(miss_time) < miss_time_min
                porm = sign(miss_time);
                miss_time_min = abs(miss_time);
                d_min = day_inc;
                z_min = z;
            end

            if check == 1 & sign(miss_time)<0 %Within 5 mins & Lander ahead of CTV
            lmnop = lmnop+1;
            opps(lmnop) = day_inc;
            zopps(lmnop) = z;
            dvopps(lmnop) = dvint;
            trx(lmnop) = trxinc;
            end
        end

        hold off
        if day_inc == day_inc(end)
            check = 1;
        end
        if tof_rl<0|tof_rc<0
            check = 1;
            display('Something is wrong')
            day_inc;
        end
    end
end
```

43 Walker, Scott

43.1.1.1 CTV thermal system sizing code

Author: Scott Walker

MLI_cold.m

```
clc
close all
clear all

% This program calculates the thickness of MLI necessary to keep the CTV at
% 300 K without an active thermal control system.  After several
% iterations a power generation of 15.7 kW was chosen as a reasonable
% amount of power to passively radiate away.  This value gives a sensible
% MLI thickness that won't force the crew to run massive amounts of heaters
% to warm the interior of the ship, but it will also allow enough power to
% radiate passively for the crew to survive if the active thermal control
% system fails.

%CTV info
radius = 4.25; %radius of crew compartment
hieght = 4.26; %hieght of crew compartment
steph = 5.67*10^-8; %Stephan-Boltzmann constant
inner_temp = 300; %nominal temp the crew can handle for mission duration
space_temp = 10; %average temperature of deep space
MLI_em = 0.82; %emissivity of MLI
Q = 15.7*1000; %power generation in crew area (W)
G = 191.103; %Irradiance/m^2 from sun (near earth)

surf_area = radius*2*pi*hieght; %approximation of radiative surface area
surf_temp_hot = ((steph*space_temp^4*MLI_em + G)*surf_area +
Q)^.25/(MLI_em^.25*surf_area^.25*steph^.25)
surf_temp_cold = ((steph*space_temp^4*MLI_em)*surf_area + Q)^.25/(MLI_em^.25*surf_area^.25*steph^.25)

%MLI
MLI_k = 0.05; %MLI conductivity

%Aluminum
AL_thick = 0.01; %thickness  of aluminum
AL_k = 237; %aluminum conductance
AL_R = AL_thick/(AL_k*surf_area); %total resistance of aluminum layer

%poly
Poly_thick = 0.04; %polyethlene thickness
Poly_k = 0.29;%0.29 to 0.5 - polyethlene conductance
Poly_R = Poly_thick/(Poly_k*surf_area); %total resistance of polyethlene layer

%radiation
hr_cold = MLI_em*steph*(surf_temp_cold + space_temp)*(surf_temp_cold^2 + space_temp^2);
Rad_R_cold = 1/(hr_cold*surf_area); %radition resistance in thermal circuit given cold conditions
hr_hot = MLI_em*steph*(surf_temp_hot + space_temp)*(surf_temp_hot^2 + space_temp^2);
Rad_R_hot = 1/(hr_hot*surf_area); %radition resistance in thermal circuit given hot conditions

%%%%%%%%%%%%%%

MLI_thick_cold = surf_area*MLI_k*((inner_temp - space_temp)/Q - AL_R - Poly_R - Rad_R_cold)
%calculates the thickness of the MLI given cold conditions
MLI_thick_hot = surf_area*MLI_k*((inner_temp - space_temp)/Q - AL_R - Poly_R - Rad_R_hot) %calculates
the thickness of the MLI given hot conditions
```

```
Q_cold_cond_hot_thick = (surf_area*MLI_k*(inner_temp - space_temp))/(surf_area*MLI_k*Rad_R_cold +
MLI_thick_hot + surf_area*MLI_k*(AL_R + Poly_R)) %amount of power radiated away using the hot
thickness in cold conditions
Q_cold_cond_cold_thick = (surf_area*MLI_k*(inner_temp - space_temp))/(surf_area*MLI_k*Rad_R_cold +
MLI_thick_cold + surf_area*MLI_k*(AL_R + Poly_R)) %amount of power radiated away using the cold
thickness in cold conditions
Q_hot_cond_cold_thick = (surf_area*MLI_k*(inner_temp - space_temp))/(surf_area*MLI_k*Rad_R_hot +
MLI_thick_cold + surf_area*MLI_k*(AL_R + Poly_R)) %amount of power radiated away using the cold
thickness in hot conditions
Q_hot_cond_hot_thick = (surf_area*MLI_k*(inner_temp - space_temp))/(surf_area*MLI_k*Rad_R_hot +
MLI_thick_hot + surf_area*MLI_k*(AL_R + Poly_R)) %amount of power radiated away using the hot
thickness in hot conditions
```

RESULTS

```
surf_temp_hot = 290.0603

surf_temp_cold = 233.4164

MLI_thick_cold = 0.0172

MLI_thick_hot = 0.0556

Q_cold_cond_hot_thick = 1.1498e+004

Q_cold_cond_cold_thick = 1.5700e+004

Q_hot_cond_cold_thick = 2.4741e+004

Q_hot_cond_hot_thick = 1.5700e+004
```

43.1.1.2 MHV Fault Tree

Author: Scott Walker

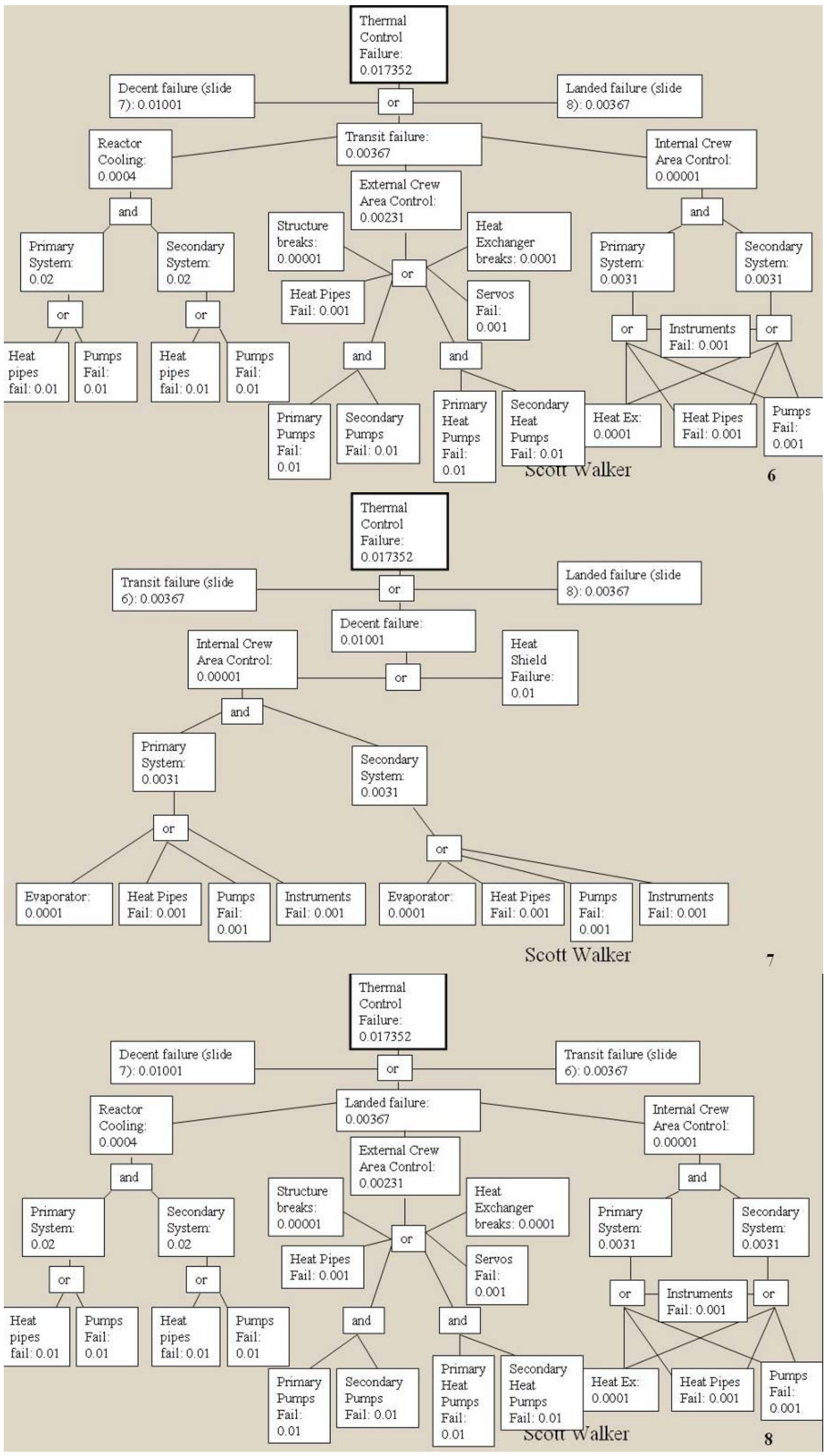


Figure 1-1 MHV fault tree

43.1.1.3 MHV Thermal System Analysis

Author: Scott Walker

Contributor: Sam Rodkey

MHV External thermal analysis

We have an analytical model to determine how much heat is passively given off by the MHV. The analysis is performed on both a hot ($T_{enviroment} = 295^oK$) day and a cold ($T_{enviroment} = 160^oK$) night. The first step of the process determines the Rayleigh number (R_a) and the Nusselt numbers (N_{uD}, N_{uL}) from environmental and material constants. N_{uD} is the Nusselt number for free convection from the length of the MHV's cylindrical body. N_{uL} is the Nusselt number for free convection from the circular ends of the cylinder.

$$\beta = \frac{1}{T_{film}} = \frac{1}{(T_{surface} + T_{enviroment})/2}$$

$$\alpha = \frac{k}{\rho C_p}$$

$$R_a = \frac{g\beta(T_{surface} - T_{enviroment})D^3}{\upsilon\alpha}$$

$$N_{uD} = \left\{0.6 + \frac{.387R_a^{1/6}}{[1+(0.559/P_r)^{9/16}]^{8/27}}\right\}^2$$

$$N_{uL} = \left\{0.825 + \frac{.387R_a^{1/6}}{[1+(0.492/P_r)^{9/16}]^{8/27}}\right\}^2$$

From these numbers a thermal resistance can be calculated (h_d, h_l). The thermal resistances are used to determine how much heat ($q_{convect\_cylinder}$, $q_{convect\_endplates}$) is freely convecting away from the body of MHV.

$$h_d = \frac{k}{D}N_{uD}$$

$$h_L = \frac{k}{L}N_{uL}$$

$$q_{convect\_cyclinder} = h_D \Pi DL(T_s - T_\infty)$$

$$q_{convect\_endplates} = 2h_L A(T_s - T_\infty)$$

A simple calculation of the heat loss rate due to body radiation ($q_{radiation}$) is based on environment temperature (T_∞), MHV surface temperature (T_s), surface area (A) and material constants.

$$q_{radiation} = \varepsilon \prod A\sigma(T_s^{\ 4} - T_\infty^{\ 4})$$

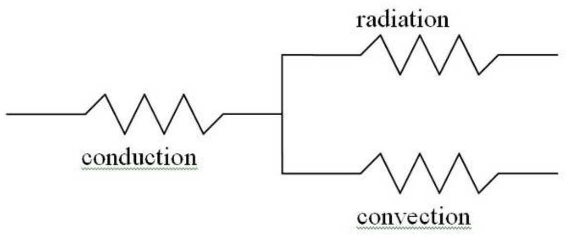
These heat fluxes are added together to get the total passive heat loss rate (q_{total}).

$$q_{total} = q_{convect\_cylinder} + q_{convect\_endplates} + q_{radiation}$$

Next the model uses different thermal circuits to represent the possible heat paths for hot days and cold nights. From this we can determine how large the radiators will be and how thick the Multi-Layer Insulation (MLI) will be.

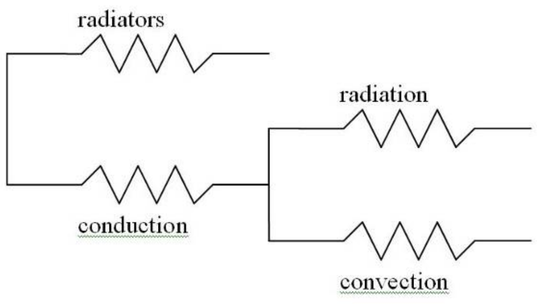
Thermal circuit – cold conditions

It is assumed that when the environment is at its coldest the radiators will not be used. Therefore, heat can only conduct out through the MLI, and then radiate from the MHV surface or convect into the atmosphere.



Thermal circuit – hot conditions

When the atmosphere is at its peak temperature the radiators will be running at full power. In this case, heat can either follow the cold circuit path or follow the parallel path to the radiators.



We now simply solve each thermal circuit like an electrical circuit; were resistors in series are added directly, and the reciprocal of the sum of the reciprocals of resistors in parallel is taken.

$$R_{conduction} = \frac{l}{kA}$$

$$R_{convection} = \frac{h_d A_{cylinder} h_l A_{endplates}}{h_d A_{cylinder} + h_l A_{endplates}}$$

$$h_{radiation} = \frac{q_{radiation}}{A_{total}(T_{surface} - T_{enviroment})}$$

$$R_{radiation} = \frac{1}{h_{radiation} A_{total}}$$

$$R_{total\_cold} = R_{conduction} + \frac{R_{convection} R_{radiation}}{R_{convection} + R_{radiation}}$$

$$h_{radiators} = \frac{q_{radiators}}{A_{radiators}(T_{radiators} - T_{enviroment})}$$

$$R_{radiators} = \frac{1}{h_{radiators} A_{radiators}}$$

$$R_{1\_hot} = R_{conduction} + \frac{R_{convection} R_{radiation}}{R_{convection} + R_{radiation}}$$

$$R_{total\_hot} = \frac{R_{1\_hot} R_{radiators}}{R_{1\_hot} + R_{radiators}}$$

From these resistances (R_{total_cold}, R_{total_hot}) we can back calculate a thickness for MLI and a radiator size. We do this by equating the total heat flow rate (Q_{total}) to the amount of power (determined by electronic equipment and human metabolism inside MHV) that must leave the MHV for it to stay at

300^{o}K. This allows us to find both *l* (the thickness of the MLI) and A$_{radiators}$ (the total necessary radiator surface area)

$$Q_{total} = \frac{T_{surface} - T_{enviroment}}{R_{total}}$$

CO$_2$ (Primary gas in Martian Atmosphere) data

	CO$_2$ 280^{o}K	CO$_2$ 300^{o}K
P (kg/m^3)	1.9002	1.7730
C$_p$ (KJ/kg-K)	.830	.851
M (N-s/m^2)	140E-7	1.49E-7
N (m^2/s)	7.36E-6	8.4E-6
ά (m^2/s)	9.63E-6	11E-6
P$_r$	.765	.766
K (W/m-K)	15.2E-3	16.55E-3

Table 1-1 CO$_2$ thermal properties [2]

Other constants

T$_{surface}$ - 300^{o}K

T$_{enviroment}$ – 160^{o}K (cold)

295^{o}K (hot)

g – 3.711 m/s^2

D – 6.5 m

L – 14 m

t – 0.0055 m

T$_{radiators}$ – 350^{o}K

k$_{MLI}$ – 0.05 W/m-K

Important results

MHV passively radiates 47.2 kW in the cold case. Only 10.0 kW are radiated passively in the hot case. The thickness of the MLI that provides these results is 0.045 m. The area of the radiators is 45 m^2. Because the MHV is capable of using up to 50.9 kW the radiators may need to operate continuously.

References:

[1] Larson, Wiley, and Linda Pranke. Human Spaceflight. New Yorkr: The McGraw-Hill Companies, Inc, 2001

[2] Incropera, Frank, and David Dewitt. Fundamentals of Heat and Mass Transfer. Hoboken: John Wiley & Sons, 2002

43.1.1.4 Active Thermal Control System Sizing

Author: Scott Walker

The thermal subsystem are sized using the equations found in Human Spaceflight [1].

	Mass (kg)	Power (kW)	Volume (m^3)
Heat exchangers	17 + 0.25 X capacity in kW	0	0.016 + 0.0012 X capacity in kW
Cold plates	12 X capacity in kW	0	0.028 X capacity in kW
Pumps with accumulator	4.8 X loop capacity in kW	.023 X loop capacity in kW	0.017 X loop capacity in kW
Plumbing & valves	Add 15% to active system	Negligible	Negligible
Instruments and controls	Add 5% to active system	Negligible	Negligible
Fluids	Add 5% to active system	0	Negligible
Heat pumps	8 X capacity in kW	Varies	Negligible
Radiators (deployable)	8.5 per m^2	Negligible	0.06 per m^2
Heat pipes	0.000294 X capacity in watts X (length in m)^2	0	2.03 X10^{-7} X capacity in watts X (length in m)^2

Table 43-1 Sizing Parameter equations from Human Spaceflight [1]

The results are as follows

	Number or size (m or m^2)	Mass (kg)	Power (kW)	Volume (m^3)
Heat exchangers	2	52	0	0.12
Cold plates	20	240	0	0.56
Pumps with accumulators	2	344	0.823	1.22
Plumbing and valves		192		
Instruments and controls		64		
Fluids		64		
Heat pumps	2	80	1.000	
MLI	250	250		2.5
Radiators	26	220	1.000	1.56
Heat pipes	30	316	0	0.22

Total		1822	2.823	6.17

Table 43-2 CTV thermal parameters

	Number or size (m or m^2)	Mass (kg)	Power (kW)	Volume (m^3)
Heat exchangers	2	54	0	0.13
Cold plates	20	240	0	0.56
Pumps with accumulators	2	390	0.94	1.39
Plumbing and valves		200		
Instruments and controls		66		
Fluids		66		
Heat pumps	2	80	1.000	
MLI	205	205		2.50
Radiators	45	380	1.000	2.71
Heat pipes	30	360	0	0.24
Total		2041	2.94	7.53

Table 43-3 MHV thermal parameters

	Number or size (m or m^2)	Mass (kg)	Power (kW)	Volume (m^3)
Heat exchangers	0	0	0	0
Cold plates	6	72	0	0.17
Pumps with accumulators	1	250	1.19	0.88
Plumbing and valves		74		
Instruments and controls		25		
Fluids		25		
Heat pumps	0	0	0	
MLI	20	20		0.2
Radiators	50	430	1.000	3.04
Heat pipes	10	150	0	0.10
Total		1048	2.19	4.4

Table 43-4 HRS thermal parameters

	Number or size (m or m^2)	Mass (kg)	Power (kW)	Volume (m^3)

Heat exchangers	0	0	0	0
Cold plates	6	72	0	0.17
Pumps with accumulators	1	107	0.51	0.38
Plumbing and valves		40		
Instruments and controls		13		
Fluids		13		
Heat pumps	0	0	0	
MLI	20	20		0.2
Radiators	22	186	1.000	1.31
Heat pipes	10	66	0	0.05
Total		520	1.51	2.1

Table 43-5 MORS thermal parameters

References:

[1] Larson, Wiley, and Linda Pranke. Human Spaceflight. New Yorkr: The McGraw-Hill Companies, Inc, 2001

44 Wright, Ray

44.1 Appendix: Artificial Gravity

Author: Ray Wright

44.1.1 The Need for Artificial Gravity

One of the mission critical designs for the Crew Transport Vehicle (CTV) is creating Artificial Gravity. The astronauts are in space for 1100 days at the maximum, and microgravity is known to have some serious health risks. The purpose of this section is explaining the process that we use in determining the minimal radius that the CTV will spin about.

44.1.2 Theory and Assumptions

Some assumptions that we use are first, the amount of gravity that the astronaut will feel at their feet is 9.81 m/s^2. Second, the CTV must have a constant spin. Third, during spin up and spin down, the astronauts will not experience more than a 25 percent gradient between their feet and their head. Finally the angular momentum pointing pulses will not change the spin rate.

From *Artificial Gravity and the Architecture of Orbital Habitats* [1], the total acceleration felt by the astronauts is defined as:

$$A = \Omega^2 r \pm 2\Omega + \frac{1}{r}$$

44-1

where Ω is the angular velocity of the CTV and r is the radius arm of the CTV. However, the 2Ω and the 1/r apply to movement parallel to the centripetal acceleration vector at a speed greater than 1 meter per second. We will assume that the astronauts only go up and down the ladder at a slow pace, so we can negate the effects of the last terms, turning Equation 44-1 into Equation 44-2.

$$A = \Omega^2 r$$

44-2

The absolute gradient is the difference in acceleration that an astronaut feels from his head to his feet. We use the standard 25 percent maximum gradient.

44.1.3 Analysis

We did two analyses with the code. First, we found the required radius arm from the center of mass to the floor of the crew quarters by rearranging Equation 44-2 to Equation 44-3.

$$r = \frac{g}{\Omega^2}$$

44-3

where g is the acceleration of gravity, and *Ω* is the angular velocity. Varying the angular velocities, we can find the radiuses required for a 1 G acceleration which is seen in Table 44-1:

Table 44-1: Angular Velocities and Radiuses

Angular Velocities (Ω)(rev/min)	Radius r (m)
2.0 (Optimal Velocity)	224
4.0	56
5.06	35 (Largest cm radius)
5.37	31 (Smallest cm radius)
5.44	30
6.0 (Max Velocity)	25

As we can see from this table, as the angular velocity increases, the radius decreases. The optimal solution is defined in [1], where people who have not gone through Air Force and Astronaut Training, can adjust properly. However, the radius of the truss is roughly the size of two football fields which is unacceptable.

We also note that the max angular velocity has the shortest moment arm. We choose 60 m as our truss length. Code written by Brendan Eash outputs the center of mass (cm) during the trip. The cm will change over time, which can be seen in Table 44-1. We account for the changing cm by changing the ranging the angular velocity of the CTV between 5.06 and 5.37 rev per min to keep a 1 G spin.

The code also finds the gravity that the astronauts experience along the truss and in the crew quarters.

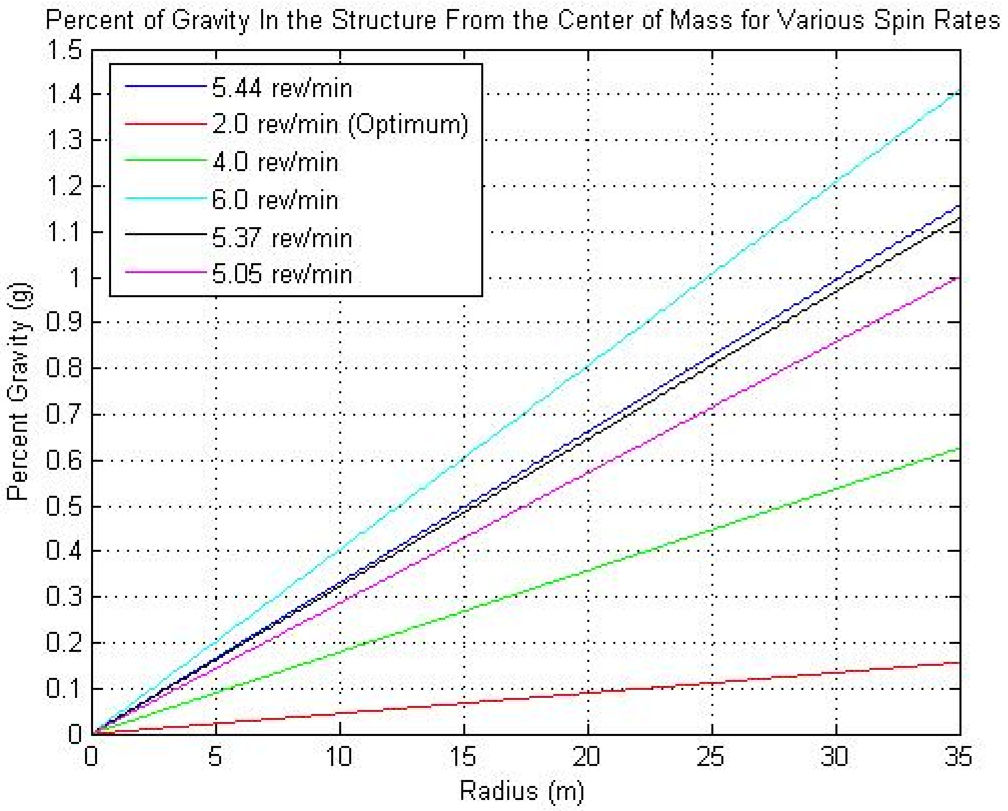


Figure 44-1: Shows the Gravity Along the Truss to the Center of Mass

Figure 44-1 shows the gravity along the truss from the center of mass to the edge of the crew quarters. Notice that total height of the crew quarters is 5 meters. The figure also shows that the minimal amount of gravity that the astronauts will experience during the trip is 0.9 g's. Therefore, we have designed for the astronauts to experience minimal amounts of microgravity, reducing the worry for the health effects during the trip.

44.1.4 Code

```
% Ray Wright
% AAE 450
% Artificial Gravity Pro Rating over the Structure
% 1/24/2005

clear all
close all
clc

% Current Radius Design
r = 0:0.1:35;

% Spin Rates
w = 5.44*2*pi/60;       % Current Design
w1 = 2*2*pi/60;         % Optimum Design
w2 = 4*2*pi/60;         % Intermitent Design
w3 = 6*2*pi/60;         % Max Design
g = 9.81;

% Finds the Radius Needed to Have 1 g based on Spin Rates
r1 = g/w^2;
r2 = g/w1^2;
r3 = g/w2^2;
r4 = g/w3^2;
w4 = sqrt(g/31);
w5 = sqrt(g/35);
% For a Spin Rate of 5.44 (rev/min)
a = w^2*r;              % Centripital Acceleration
p = a/g;                % Percent G in # of G's

% For a Spin Rate of 2 (rev/min) (Optimum)
a1 = w1^2*r;            % Centripital Acceleration
p1 = a1/g;              % Percent G in # of G's

% For a Spin Rate of 4 (rev/min)
a2 = w2^2*r;            % Centripital Acceleration
p2 = a2/g;              % Percent G in # of G's

% For a Spin Rate of 4 (rev/min)
a3 = w3^2*r;            % Centripital Acceleration
p3 = a3/g;              % Percent G in # of G's

% For a Spin Rate of 4 (rev/min)
a4 = w4^2*r;            % Centripital Acceleration
p4 = a4/g;              % Percent G in # of G's

% For a Spin Rate of 4 (rev/min)
a5 = w5^2*r;            % Centripital Acceleration
p5 = a5/g;              % Percent G in # of G's

fprintf('|------ omega (rev/min) ------|--------- r (m) ---------|\n')
fprintf('|          %f           |         %f       |\n',w*60/(2*pi),r1)
fprintf('|          %f           |         %f      |\n',w1*60/(2*pi),r2)
fprintf('|          %f           |         %f       |\n',w2*60/(2*pi),r3)
fprintf('|          %f           |         %f       |\n',w3*60/(2*pi),r4)
fprintf('|          %f           |         %f       |\n',w4*60/(2*pi),31)
fprintf('|          %f           |         %f       |\n\n\n\n\n',w5*60/(2*pi),35)

figure(1)
plot(r,p)
hold on
plot(r,p1,'r')
plot(r,p2,'g')
plot(r,p3,'c')
plot(r,p4,'k')
plot(r,p5,'m')
xlabel('Radius (m)')
```

```
ylabel('Percent Gravity (g)')
title('Percent of Gravity In the Structure From the Center of Mass for Various Spin Rates')
legend('5.44 rev/min','2.0 rev/min (Optimum)','4.0 rev/min','6.0 rev/min','5.37 rev/min','5.05
rev/min','location','northwest')
grid
```

44.1.5 Reference

[1]. Hall, Theodore W. *Artificial Gravity and the Architecture of Orbital Habitats*. 20 March 1997. http://www.spacefuture.com/archive_artificial_gravity_and_the_architecture_of_orbital_habitats.shtml

44.2 Appendix: Auxiliary Tank Configuration Code

Author: Ray Wright

44.2.1 Description

For any of the theory, please refer to section 2.3.6.5, written by Ray Wright. This section outlines the inputs and outputs of the code for the analysis done in section 2.3.6.5.

44.2.2 Inputs and Outputs

As described in Section 2.3.6.5, the auxiliary tank sizing is a simple process. This code is very flexible. It will take any fuel mass, and find a tank size that will be in a spherical shape. The pressure in Pa and temperature in Celsius are also inputs that can be changed. The molecular weight of the fuel is also an input.

From the inputs, the code first determines the volume of the material and the radius of the sphere needed to enclose the fuel. The code also outputs, thickness in meters, and the actual tank mass in kg.

This code was modified to find the tank mass of the ARV for the Helium required to fill the ballute upon reentry. The results can be seen in section 2.2.2.2.

44.2.3 Code

```
% Ray Wright
% AAE 450
% 3/24/2005
% Xenon Gas Tank Sizing

close all
clear all
clc
format long g

% constants
m(1) = 1600;            % kg; Mass for Spin Up/Down and Angular Momentum
                        % pointing

m(2) = 1800;            % kg; Mass for Station Keeping

R = 8.3144;             % J/(mol*K; Universal Gas Constant
T = -110;               % Celcius liquid
p = 100;                  % (atm); pressure of storage
Xe = 131;               % g/mol; Molecular Weight of Xenon
yield_stress = 1.60e9/4;   % Carbon Epoxy
safety_factor = 1.5;    % Includes MLI
ii = 0;
for ii = 1:2            % Finds the Number of Mole (mol)
    n(ii) = m(ii)*1000/Xe;
end
```

```
T = T+273.15;           % Converts from Celcius to Kelvin
p = p * 101325;         % Converts from atm to N/m^2

for ii = 1:2;           % Finds the Volume needed for the Storage
    V(ii) = n(ii)*R*T/p;
end


for ii = 1:2;           % Finds the Radius of a Sphere for the Storage Container
    r(ii) = (3*V(ii)/(4*pi))^(1/3);
end
for ii = 1:2;
    t(ii) = sqrt(3)*p*r(ii)/yield_stress;
end

for ii = 1:2;
    mass_tank(ii) = 4/3*pi*((r(ii)+t(ii))^3-r(ii)^3)*1000;
end

fprintf('|                   |  Spin/H pointing  |  Station Keeping |\n')
fprintf('|-------------------|-------------------|------------------|\n')
fprintf('| Fuel  m (kg)      |   %f     |   %f    |\n',m(1),m(2)')
fprintf('|       n (mol)     |   %f    |     %f |\n',n(1),n(2)')
fprintf('|       V (m^3)     |   %f        |    %f      |\n',V(1),V(2)')
fprintf('|       r (m)       |   %f        |     %f     |\n',r(1),r(2)')
fprintf('|       p (Pa)      |   %f |  %f |\n',p,p')
fprintf('|       T (K)       |   %f      |     %f   |\n',T,T')
fprintf('|       t (m)       |   %f        |     %f     |\n',t(1),t(2)')
fprintf('|tank mass (kg)     |   %f      |     %f   |\n',mass_tank(1),mass_tank(2)')
fprintf('| Other m (kg)      |   %f     |     %f   |\n',2000,700')
fprintf('|-------------------|-------------------|------------------|\n\n\n\n\n')
```

44.3 Appendix: Initial Wall Thickness Calculations

Author: Ray Wright

44.3.1 Description

The code in this section describes how we considered a multitude of materials and wall thickness for the crew quarters and fuel tanks.

44.3.2 Theory and Assumptions

Some assumptions that we have are the following, first, all material is elastic, meaning that when the material is deformed, it will go back to its original shape without any damage. Second, the material has a uniform density. Finally, the internal volume is 205 m^3 with a height of 5 meters.

The four major factors that we consider when we design for the pressure vessel are hoop stress, longitudinal stress, yield stress and weight. Hoop stress is defined as:

$$\sigma_H = \frac{pr}{t}$$

44-4

where p is the internal pressure in Pa, r is the radius, and t is the thickness of the pressure vessel. Longitudinal stress is defined as:

$$\sigma_L = \frac{pr}{2t}$$

44-5

where p, r and t are defined above.

Using the initial volume, and varying the thicknesses, the volume can be found by assuming a height of five meters.

$$V = \pi\left((r_i + t)^2 - r_i^2\right)h$$

44-6

where r_i is the internal radius, t is the thickness, and h is the height of the cylinder. We then multiply the volume by the materials density, to find the mass of the material that we will use to build the structure.

44.3.3 Analysis

From [1], we know that the max stress in a pressurized cylinder is the hoop stress. Setting the hoop stresses to the yield stress of the desired material, we solve for the minimal thickness needed for the material, which can be seen in Table 44-2.

Table 44-2: Shows the Minimum Thickness based on Material Yield Stresses

Material	Yield Stress (MPa)	Min Thickness (m)
Aluminum	490	0.0007
Magnesium	280	0.001
Steel	700	0.0005
Titanium	1000	0.0004
Carbon Epoxy	156	0.0025

From Table 44-2, we see that if we use the Carbon Epoxy, we are going to use the most material. Using Equations 44-4 and 44-5, we can find the stresses exerted on the structure which can be seen in Figure 44-2.

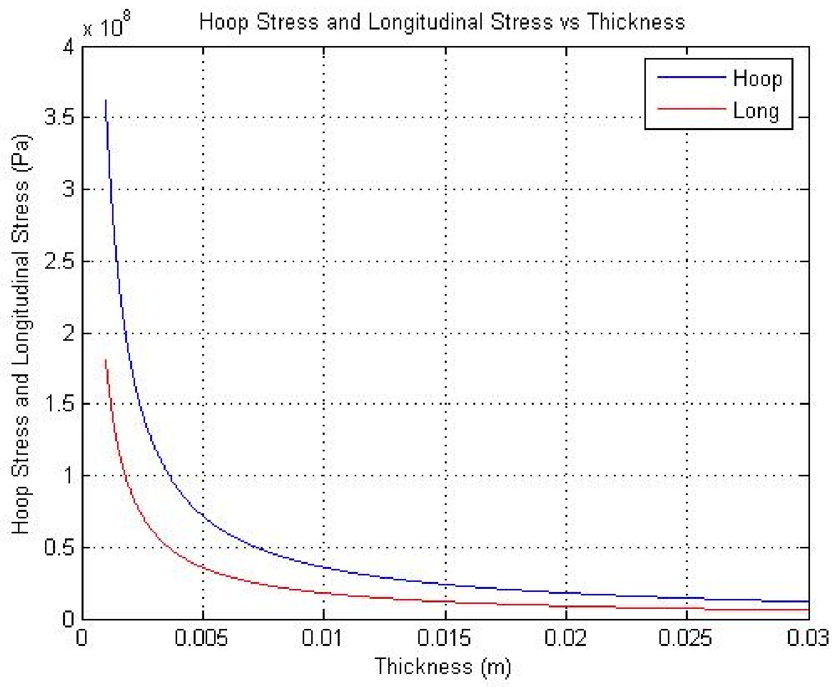


Figure 44-2: Hoop and Longitudinal Stress with Varying Thicknesses

From Figure 44-2, we see that the stresses start to level out at around two cm. We will use this value and assume that the stress savings is not worth the mass increase if the thickness is increased. Using a thickness of two cm, we find the mass of the material used for the crew quarters in Table 44-3.

Table 44-3: Mass of Material Used based on a Thickness of 2 cm

Material	Thickness (m)	Mass (kg)
Aluminum	0.02	6295
Magnesium	0.02	3980
Steel	0.02	18481
Titanium	0.02	10095
Carbon Epoxy	0.02	3890

We can see from Table 44-3 using Carbon Epoxy as the material for the outer walls will save the most mass. This can also be seen in Figure 44-3.

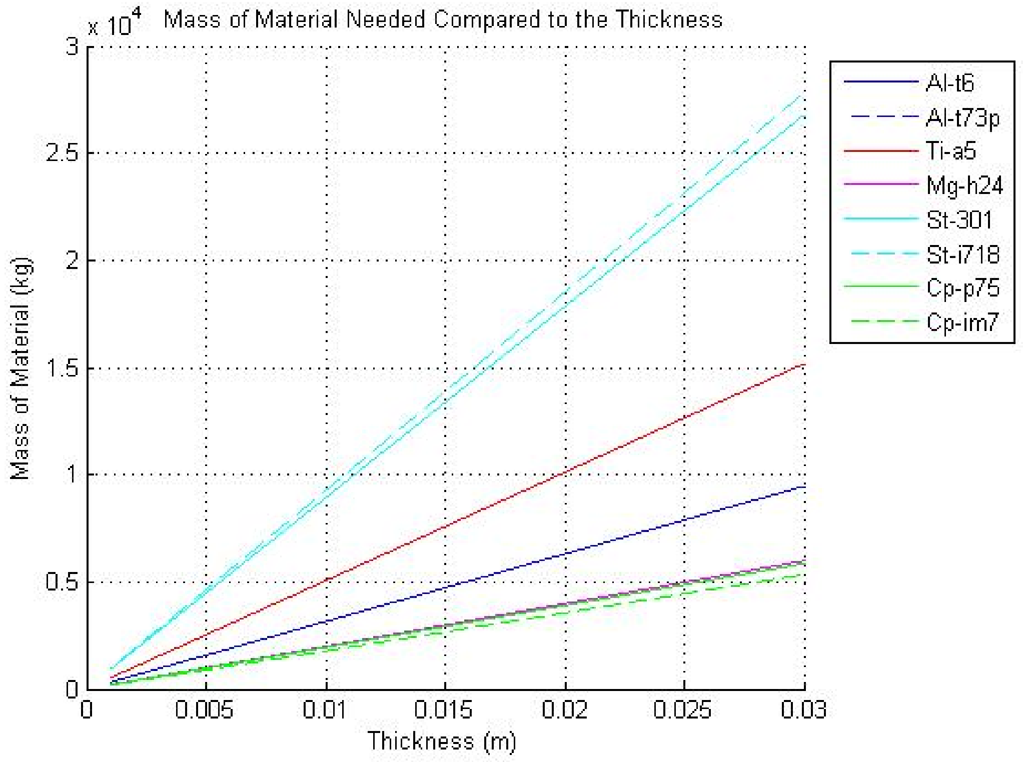


Figure 44-3: Masses Of Materials Dependent on a Varying Thickness

We see that from the picture, as the thickness is increased, the masses increase linearly. Thus Carbon Epoxy should be used to for the walls of the crew quarters. The material will provide ample protection and keep the mass down.

44.3.4 Code

```
% Ray Wright
% AAE 450
% Structures
% Crew Area
%
% This code is a simple model of types of stresses that the vehilce will undertake
% Assumptions: Atmospheric Pressure, 7ft allowance for the crew, 1 structure

close all;
clear all;
clc;

% Constants for the ship
p = 101325;                 % Pressure (Pa = N/m^2)
Vi = 200;                    % Inner Volume (m^3)
h = 5;                      % Height (m)

% Figuring out the inner/outer radius and thickness of the Vehicle Material
r1 = sqrt(Vi/(pi*h))         % Inner Radius
ti = linspace(0.001,0.03,2^12);
I = pi*r1^3*ti;
A = pi*((r1+ti).^2-r1^2);
i = size(ti);
L = 30;
% choose E: Al = 79e9; Mg = 45e9; Steel = 210e9; Ti = 120e9; 177e9;
E = [79e9 45e9 210e9 120e9 177e9];
E = E(5);
sig_crit = pi^2*E*I./(A*L^2);
% Finds the hoop and longitudinal stresses based on different thickness
for ii = 1:i(2);
    r2(ii) = ti(ii) + r1;
    Vo(ii) = pi*((r2(ii))^2 - r1^2)*h;
    hoop(ii) = p*r1/ti(ii);
    long(ii) = p*r1/(2*ti(ii));
    tmax(ii) = p*r1/(2*ti(ii));
end

% Min Thickness
% choose E: Al = 490e6; Mg = 280e6; Steel = 700e6; Ti = 1000e6; 156e6;
yield_stress = [490e6 280e6 700e6 1000e6 156e6];
for i = 1:length(yield_stress)
    tmin(i) = p*r1/yield_stress(i)
end

rho = [2.8 1.77 8.22 4.49 1.73]*100^3/1000;
Vm = pi*((r1+0.02)^2 - r1^2)*h;
for i = 1:length(rho);
    mass(i) = Vm*rho(i);
end


% Aluminum Alloys
t6p = 2.8*100^3/1000;       % density (kg/m^3)
t73p = 2.8*100^3/1000;      % density (kg/m^3)

% Titanium
a5 = 4.49*100^3/1000;       % density (kg/m^3)

% Magnesium
h24 = 1.77*100^3/1000;       % density (kg/m^3)

% Steel
s301 = 7.92*100^3/1000;       % density (kg/m^3)
i718 = 8.22*100^3/1000;       % density (kg/m^3)

% Composites
p75 = 1.73*100^3/1000;       % density (kg/m^3)
```

```
im7 = 1.58*100^3/1000;       % density (kg/m^3)

fprintf('|------ Material  ------|------ Yield Stress  ------|---- Thickness (m) -----|\n')
fprintf('|      Aluminum         |          %f       |         %f
|\n',yield_stress(1)/1e6,tmin(1))
fprintf('|      Magnesium        |          %f       |         %f
|\n',yield_stress(2)/1e6,tmin(2))
fprintf('|      Steel            |          %f       |         %f
|\n',yield_stress(3)/1e6,tmin(3))
fprintf('|      Titanium         |          %f      |         %f
|\n',yield_stress(4)/1e6,tmin(4))
fprintf('|      Carbon Epoxy     |          %f       |         %f
|\n',yield_stress(5)/1e6,tmin(5))
fprintf('\n\n\n\n\n')

fprintf('|------ Material  ------|-------- Mass (kg) --------|---- Thickness (m) -----|\n')
fprintf('|      Aluminum         |          %f       |         %f       |\n',mass(1),0.02)
fprintf('|      Magnesium        |          %f       |         %f       |\n',mass(2),0.02)
fprintf('|      Steel            |          %f      |         %f       |\n',mass(3),0.02)
fprintf('|      Titanium         |          %f      |         %f       |\n',mass(4),0.02)
fprintf('|      Carbon Epoxy     |          %f       |         %f       |\n',mass(5),0.02)
fprintf('\n\n\n\n\n')

figure(1)
plot(ti,hoop)
hold on
plot(ti,long,'r')
xlabel('Thickness (m)')
ylabel('Hoop Stress and Longitudinal Stress (Pa)')
title('Hoop Stress and Longitudinal Stress vs Thickness')
legend('Hoop','Long')
grid
hold off

figure(2)
plot(ti,Vo)
xlabel('Thickness (m)')
ylabel('Volume of Material (m^3)')
title('Volume of Material Needed Compared to the Thickness')
grid

% Plots the Masses of the Material
figure(3)
hold on
plot(ti,Vo*t6p)
plot(ti,Vo*t73p,'--')
plot(ti,Vo*a5,'r')
plot(ti,Vo*h24,'m')
plot(ti,Vo*s301,'c')
plot(ti,Vo*i718,'c--')
plot(ti,Vo*p75,'g')
plot(ti,Vo*im7,'g--')
xlabel('Thickness (m)')

ylabel('Mass of Material (kg)')
title('Mass of Material Needed Compared to the Thickness')
legend('Al-t6','Al-t73p','Ti-a5','Mg-h24','St-301','St-i718','Cp-p75','Cp-
im7','location','NorthEastOutside')
grid
hold off
```

44.3.5 Reference

[1] Gere, James M. Mechanics of Materials. Pacific Grove CA: Brooks/Cole, 2001.

44.3.6 Biliography

www.matweb.com

44.4 Appendix: Initial Calculation of the Mass of the CTV

Author: Ray Wright

Contributor: Matt Harrigan

44.4.1 System Description

In the initial design phase, we need to know an initial structural mass estimate of the Crew Transport Vehicle (CTV). The estimate provides a starting place for the Propulsion Group in determining what type of engines to use, and the size of the engines needed to propel the CTV to Mars and back.

44.4.2 Theory and Assumptions

This code determines an approximation of the mass of the crew quarters of the CTV using Carbon Epoxy as the shell. It also determines the mass of the truss on the CTV using Aluminum as the material. We assume that all of the material has a constant density, with a uniform distribution. When we design for the crew quarters, we must make sure that the thickness of the outside walls can support the stresses created from the internal pressure of 1 atmosphere. Two main stresses are considered when dealing with a pressure vessel, they are hoop and longitudinal. Hoop Stress is defined as:

$$\sigma_H = \frac{pr}{t}$$

44-7

where *p* is the internal pressure, *r* is the radius of the cylinder, and *t* is the thickness of the wall. Longitudinal stress is defined as:

$$\sigma_L = \frac{pr}{2t}$$

44-8

where *p*, *r*, and *t* are defined previously. The are several advantages with Carbon Epoxy. First, the material has a low density, which implies lower masses. Secondly, carbon epoxy is extremely strong, and withstands large loads and stresses, thus providing ample protection for the astronauts. One thing that we must keep in mind is that the hoop and longitudinal stresses must not be greater than the yield stress. Since we know the radius, and the pressure, and we know the yield stress we can solve for the minimum thickness needed to have a safe structure. Using Von Mises Yield Criterion we can relate hoop and longitudinal stress to the yield stress by using 44-9:

$$\frac{1}{3}\left(\sigma_1^2 - \sigma_1\sigma_2 + \sigma_2^2\right) = \frac{1}{3}\sigma_Y^2$$

44-9

where σ_1 is the hoop stress, σ_2 is the longitudinal stress, and σ_Y is the yield stress of the material, we can solve for the thickness of the material, which we see in the following derivation.

$$\frac{1}{3}\left(\frac{p^2r^2}{t^2} - \frac{p^2r^2}{2t^2} + \frac{p^2r^2}{4t^2}\right) = \frac{1}{3}\sigma_Y^2$$

44-10

Here we take Equation 44-7 and Equation 44-8 and insert them into Equation 44-9.

$$\frac{3p^2r^2}{4t^2} = \sigma_Y^2$$

44-11

We reduce Equation 44-10 into Equation 44-11. Then we solve for the thickness and get Equation 44-12.

$$t = \frac{\sqrt{3}pr}{2\sigma_Y}$$

44-12

Equation 44-12 solves for the minimum thickness of the walls so that the hoop and longitudinal stress will not exceed the yield stress. We then find the volume of the material, multiply that by the density to find the mass of the structure.

To find the masses of the main columns of the truss, we are going to consider a preliminary analysis based on yield criterion and buckling. The code outputs the thicknesses based on these criteria. Note: This analysis was done on the original concept of the four truss system. We assume that the main columns of the truss are un-pressurized thin cylinders. We also assume that there are cables that take all of the bending loads. Equation 44-13 is the critical stress associated with the critical load.

$$\sigma_{cr} = \frac{P_{cr}}{A}$$

44-13

where P_{cr} is the load applied on the truss and A is the cross-sectional area of the truss. T. The P_{cr} is defined in Equation 44-14.

$$P_{cr} = \frac{\pi^2 EI}{L^2}$$

44-14

Where *E* is the Young's Modulus, *I* is the moment of Inertia, and *L* is the length of the rod.

$$\sigma_{cr} = \frac{\pi^2 EI}{AL^2}$$

44-15

Where *E*, *I*, *L*, and *A* are defined previously. From the cross sectional area, we can find the thickness.

44.4.3 Analysis

Using Equation 44-12, we can find the minimum thickness of the crew quarters and the mass. This can be seen in Table 44-4.

Table 44-4: Wall and Cap Thickness and Mass for the Crew Quarters

Crew Quarters	Thickness (m)	Mass (kg)
Outside Walls	0.004	459
End Caps	0.004	381
Floors and Inner Walls	-----	7560
Total Mass of Crew Structure	-----	8400

The table shows the minimum thickness needed for the crew walls and caps. The thickness can be reduced if stiffeners and stringers are used. However this is only a preliminary study. The code also assumes that the floors, inner walls, joists for the floors and support for the freezer will be roughly ten times the mass of the crew quarter shell, to give an overall mass of 8400 kg.

For the truss, Equations 44-14 and 44-15 are used to find the thickness of the truss. Assuming an outside diameter of 10 centimeters and a length of 20 m, the thickness of the truss beams can be found and seen in Table 44-5.

Table 44-5: Thickness Based on Failure Criteria and Mass based on Largest Thickness

Failure Mode	Thickness (m)	Mass (kg)
Compression	0.0001	------
Tension	0.001	------
Mass of Truss	-------	11274

This table shows that the tension in the truss is the driving factor. We then calculate the mass of the truss by assuming a that there are going to be three beams the make up the truss. Thus the mass for one truss is 5637 kg, and the mass for two trusses is seen in the table.

From further calculations of bending, compression loads, tension loads, and buckling loads, the hollow truss beam was abandoned. The design radically changed, and from these initial calculations it was obvious that a solid beam would greatly increase the rigidity of the structure and eliminate some complicated failure modes such as buckling.

44.4.4 Code

```
% Ray Wright
% Matt Harrigan

close all
clear all
clc
format long g

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Crew Area
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Inputs
inner_rad = 3.6;
height_crew_area = 5;
yield_stress = 156e6;
rho_alum = 1000;
p = 101325
safety_factor = 2;
mass_frac_crew_quarters = 10;

 % This Value is Anything that is NOT related to the Structure of the Crew
 % Quarters, for example the fridge, computers, etc.  For Structure Purpose
 % Set = 0
mass_non_struc_crew_quarters = 0;

% Thickness of Crew Living Area Wall
t_crew_wall = sqrt(3)*p*inner_rad/(2*yield_stress)*safety_factor
t_crew_cap = p*inner_rad/yield_stress*safety_factor

V_struct = pi*((inner_rad+t_crew_wall)^2 - inner_rad^2)*height_crew_area +
2*pi*(inner_rad)^2*t_crew_cap  % Volume of Material for skin and the two caps which make the pressure
Vessel (m^3).
mass_crew_shell = V_struct*rho_alum

% Stuctural Mass of Crew Quarters
mass_crew_struct = mass_crew_shell*mass_frac_crew_quarters
mass_crew_final = mass_crew_struct + mass_non_struc_crew_quarters

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perpendicular to Spin Axis TRUSS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yield_stress = 490e6;
rho_alum = 2800;
truss_length = 20;
truss_diameter = 0.15;
load_on_truss = 9.8*mass_crew_final
thickness_ten_truss = load_on_truss/(4*yield_stress*pi*truss_diameter)*safety_factor*10
mass_truss1 = pi*truss_diameter*thickness_ten_truss*4*truss_length*rho_alum*10


% Buckling
youngs_mod = 70e9;
accel_ship = 10;
theta_cable =60*pi/180;
col_buckling_coeff = 1;
col_rad_centroid = 1;

ten_in_cable = mass_crew_final*accel_ship/cos(theta_cable)
comp_in_truss = sin(theta_cable)*ten_in_cable
thickness_buckling1_truss = comp_in_truss *
truss_length/(pi^3*youngs_mod*truss_diameter*4*col_rad_centroid)*100

t = 0.0005:0.00001:0.01;
phi_buck = 1/16*sqrt((truss_diameter*.5)./t);
gam_buck = 1-.901*(1-exp(-phi_buck));
pcr = 0.6*gam_buck*youngs_mod.*t./(truss_diameter*.5);
```

```
%plot(t,pcr)
[x,y] = min(pcr - safety_factor/4*comp_in_truss)
```

44.4.5 Bibliography

Gere, James M. Mechanics of Materials. Pacific Grove CA: Brooks/Cole, 2001.

Sun, C.T. Mechanics of Aircraft Structures. New York NY: John Wiley and Sons, 1998

44.5 Appendix: Micrometeorite Shielding Code

Author: Ray Wright

44.5.1 Description

For any of the theory, please refer to section 2.3.6.2, written by Ray Wright. This section outlines the inputs and outputs of the code for the analysis done in section 2.3.6.2.

44.5.2 Inputs and Outputs

The code has only a few inputs. It needs the density of the micrometeorite in g/cm^3 and largest dimension of the micrometeorite assuming a sphere in cm. For the trip, we need to know the heliocentric radius for the starting and end destination in Astronomical Units (AU). The relative velocity of the micrometeorites are already set, however, the range can be changed. Also, the mission duration can be varied. One major input is the surface area of the vehicle in m^2.

The code outputs a number of things. First, the code calculates the number of times that the CTV is expected to get hit depending on the mass ranges of the micrometeorites. Next, the probability of getting hit on a free and non-free return graph appears on the screen, followed by a recommended bumper or back up wall thickness. Finally the code outputs the probability of catastrophic failure during the mission.

44.5.3 Code

```
% Ray Wright
% AAE 450
% Structures
% Micrometeorites and Probabilities

close all
clear all
clc
format long g


% Constants
k = 0.54;                          % Aluminum Constant
rho_m = 7.86;                      % Meteorite Density (g/cm^3)
vol = 4/3*0.01^3;                  % Volume of Micro   (cm^3)
v = linspace(10,75,2^12);          % Velocity of Micro (km/s)
r1 = 1;                            % Initial Heliocentric Radius
r2 = 1.52;                         % Final Heliocentric Radius
m_micro = rho_m*vol;               % Mass of Micro     (g)

% Threshold Penetration Thickness
t = k*rho_m^(1/6)*m_micro^(0.352).*v.^(0.875);

% Number of Impacts On The Journey
surface_area_ctv= 351.275;                  % (m^2)
t1 = 1100*24*3600;                       % Free Return Travel in (s)
t2 = 151*24*3600;                        % Nominal to Mars in (s)
```

```
t3 = 211*24*3600;                       % Nomianl Return to Earth in (s)

m1 = linspace(10^-6,10^0,2^12);         % Micrometeorite Distribution 0f mass (g)
m2 = linspace(10^-12,10^-6,2^12);       % Micrometeorite Distribution 0f mass (g)

R = linspace(r1,r2,2^12);               % Distance from the sun (AU)

Sc1 = 10.^(-18.173 - 1.213*log10(m1) - 1.5*log10(R));
Sc2 = 10.^(-18.142 - 1.584*log10(m2) - 0.063*(log10(m2)).^2 - 1.5*log10(R));
Fc1 = 10.^(-14.37-1.213*log10(m1));                       % Meteoroid Flux Based on m1
Fc2 = 10.^(-14.339-1.584*log10(m2)-0.063*(log10(m2)).^2);  % Meteoroid Flux Based on m2
Fc1_avg = mean(Fc1);
Fc2_avg = mean(Fc2);

% Number of Impacts From Earth To Mars and Back Based On Distribution
N_Fc1_t1 = Fc1_avg*surface_area_ctv*t1
N_Fc1_t2 = Fc1_avg*surface_area_ctv*t2
N_Fc1_t3 = Fc1_avg*surface_area_ctv*t3
N_Fc2_t1 = Fc2_avg*surface_area_ctv*t1
N_Fc2_t2 = Fc2_avg*surface_area_ctv*t2
N_Fc2_t3 = Fc2_avg*surface_area_ctv*t3

trip_time = linspace(4,1100,2^12);

% Probabilty Constants
spd_leo = linspace(1e-6*1/1000^3,0.001*1/1000^3,2^12);       % 1/m^3
spd_geo = linspace(1e-8*1/1000^3,1e-6*1/1000^3,2^12);        % 1/m^3
ac_crew = 5.2*4.75*2;%pi*((r_crew+t_crew)^2);        % m^2
ac_tank = 19.5*9.5;%pi*((r_tank+t_tank)^2);        % m^2

t1 = linspace(0,1100*24*3600,2^12);
t2 = linspace(0,151*24*3600,2^12);
t3 = linspace(0,211*24*3600,2^12);

% Probabilty for m1 Free, Outbound, Inbound
PC_leo_crew = 1-exp(-Sc1.*ac_crew.*t1.*v);
PC_leo_tank = 1-exp(-Sc1.*ac_tank.*t1.*v);

PC_leo_crew2 = 1-exp(-Sc1.*ac_crew.*t2.*v);
PC_leo_tank2 = 1-exp(-Sc1.*ac_tank.*t2.*v);

PC_leo_crew3 = 1-exp(-Sc1.*ac_crew.*t3.*v);
PC_leo_tank3 = 1-exp(-Sc1.*ac_tank.*t3.*v);

% Probabilty for m1 Free, Outbound, Inbound
PC_geo_crew = 1-exp(-Sc2.*ac_crew.*t1.*v);
PC_geo_tank = 1-exp(-Sc2.*ac_tank.*t1.*v);

PC_geo_crew2 = 1-exp(-Sc2.*ac_crew.*t2.*v);
PC_geo_tank2 = 1-exp(-Sc2.*ac_tank.*t2.*v);

PC_geo_crew3 = 1-exp(-Sc2.*ac_crew.*t3.*v);
PC_geo_tank3 = 1-exp(-Sc2.*ac_tank.*t3.*v);

% Probability of Catastrophic Failure
Alum_out_wall = 0.05;
Poly_in_wall = 0.05;
Comp_pressure_wall = 0.05;
stringers = 0.01;
prob_catastrophic_failure = Alum_out_wall*Poly_in_wall*Comp_pressure_wall*stringers*100 % (%)
% Plots
figure(1)
plot(v,t)
title('Penetration Thickness Based on the Velocity of the Micrometeorite')
xlabel('Velocity (km/s)')
ylabel('Thickness (cm)')

figure(2)
hold on
plot(t1*1/(24*3600),PC_leo_crew)
plot(t1*1/(24*3600),PC_leo_tank,'r')
```

```
xlabel('Trip Time (days)')
ylabel('Probability')
title('Probability of Impact vs Velocity for a Free Return based on 10^-^6< m <10^0')
legend('Crew Wall','Fuel Tank')

figure(3)
subplot(211)
hold on
plot(t2*1/(24*3600),PC_leo_crew2)
plot(t2*1/(24*3600),PC_leo_tank2,'g')
hold off
xlabel('Trip Time (days)')
ylabel('Probability')
title('Probability of Impact vs Velocity for the Outbound Leg of the non-Free Return based on 10^-^6<
m <10^0')
legend('Crew Wall','Fuel Tank')
subplot(212)
hold on
plot(t3*1/(24*3600),PC_leo_crew3)
plot(t3*1/(24*3600),PC_leo_tank3,'c')
hold off
xlabel('Trip Time (days)')
ylabel('Probability')
title('Probability of Impact vs Velocity for the Inbound Leg of the non-Free Return based on 10^-^6< m
<10^0')
legend('Crew Wall','Fuel Tank')

figure(4)
hold on
plot(t1*1/(24*3600),PC_geo_crew)
plot(t1*1/(24*3600),PC_geo_tank,'r')
xlabel('Trip Time (days)')
ylabel('Probability')
title('Probability of Impact vs Velocity for a Free Return based on 10^-^1^2< m <10^-^6')
legend('Crew Wall','Fuel Tank')

figure(5)
subplot(211)
hold on
plot(t2*1/(24*3600),PC_geo_crew2)
plot(t2*1/(24*3600),PC_geo_tank2,'g')
hold off
xlabel('Trip Time (days)')
ylabel('Probability')
title('Probability of Impact vs Velocity for the Outbound Leg of the non-Free Return based on 10^-
^1^2< m <10^-^6')
legend('Crew Wall','Fuel Tank')
subplot(212)
hold on
plot(t3*1/(24*3600),PC_geo_crew3)
plot(t3*1/(24*3600),PC_geo_tank3,'c')
hold off
xlabel('Trip Time (days)')
ylabel('Probability')
title('Probability of Impact vs Velocity for the Inbound Leg of the non-Free Return based on 10^-^1^2<
m <10^-^6')
legend('Crew Wall','Fuel Tank')
```

45 Yochum, Robert A.

45.1 MHV transit Appendix

Author(s): Robert Yochum

This section is broken up according to Programs with their functions. Main programs are designated with a header block as seen below in the first main program(TNRE). Most programs have comments in them or have variables that are easy to understand when compared to the chapter section.

45.1.1 TNRE.m Main Program

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Robert Yochum      Created: Feb 10 2005         Edited: Mar 09,'2005   %
%                                                                        %
% Engine performance program, Strictly designed for the developement of  %
% a nuclear rocket. Test bed for performance is the Nerva Project from   %
% the 1970's.                                                            %
%                                                                        %
% SubPrograms needed                                                     %
%     GamaSet                                                            %
%     Mach                                                               %
%     P_core                                                             %
%     V_core                                                             %
%     FigureT                                                            %
%     OutPutTable                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
clc

go=9.80656;
Ru=8314.41;
Tc=3050;
Pc=10000000;

ghost2=input('\nChoose a propellant\n\nH2  = 1\nCH4 = 2\n\nCO2 = 3\nH2O = 4\n\nChoice:   ');

Gamaset  %%finds gama at a given temperature
Points=50;
EpsX=linspace(1,110,Points);
a=sqrt(gama*Tc*Ru/Mass);
C_Star=sqrt(Ru/Mass*Tc/gama)*((gama+1)/2)^((gama+1)/(2*(gama-1)));

for i=1:Points
   M(i)=Mach(gama,EpsX(i)); %single error check
   PE(i)=Pc/((1+(gama-1)/2*M(i)^2)^(gama/(gama-1)));
   Pr(i)=PE(i)/Pc;
   Cf(i)=sqrt(2*gama^2/(gama-1)*(2/(gama+1))^((gama+1)/(gama-1))*(1-Pr(i)^((gama-
1)/gama)))+EpsX(i)*(Pr(i));
   Isp(i)=(Cf(i)*C_Star/go);
end

bat=1;
%while bat==1
   %User Input Section

Thrust=21070.58;%input('Input desired thrust of engines:   ');
tb=31434.56;%input('Input total burn time of Engines (seconds):  ');
Pd=2000;%input('Input power density (Nerva:1570 MW/m^3):  ');
%%finds Core power and number of engines for a given thrust
%%Core(i,j)=[power of core, number of engines, m_dot of engines, Volume per core, Energy per core,
Pipe Area, Core Mass]

%%m_dot calculation
```

```
Core(1,3)=Thrust/max(Isp)/go;
P_core

V_core

%ThrustCurve
At=Core(1,3)*sqrt(gama*Tc*Ru/Mass)/max(Pc)/(gama*(2/(gama+1))^((gama+1)/(2*(gama-1))));
Rt=sqrt(At/pi);
ChaMass=((pi*(R_core+.02)^2*H_core-pi*(R_core)^2*H_core)+(4/3*pi*(R_core+.02)^3-
4/3*pi*(R_core)^3))*(19.17/1000*1000000);

for i=1:Points
   Aet(i)=EpsX(i)*At;
   Ret(i)=sqrt(Aet(i)/pi);
   NozLength(i)=(Ret(i)-Rt)/tan(15*pi/180);
   NozMass(i)=(-
1/3*pi*NozLength(i)*Ret(i)^2+1/3*NozLength(i)*pi*(Ret(i)+.02)^2)*(19.17/1000*1000000);%Density of
Tungsten 19.17 g/cm3
   TotMass(i)=NozMass(i)+ChaMass+Core(1,7);
end
TotalLength=max(NozLength)+H_core+2*R_core;
FuelMass=Core(1,3)*tb+.05*(Core(1,3))*45000*4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FissionDensity=Core(1,1)/3.2e-11;
S=2.42*FissionDensity;
r=linspace(0,2500,Points);
O_gama_r=S./2*log(1+R_core^2./r.^2);               %assuming disk source
FNF=.8*(1/(1-2/3)-1)*O_gama_r;
Radiation=FNF+O_gama_r;
Rems=min(Radiation)*3600/1000;



OutPutTable
%end
%%%%%%%%%%%%%%%%%%%
FigureT
```

45.1.1.1 Subprogram GamaSet (for TNRE.m)

```
Mass1=[2.016,16.043,44.01,18.015];
%%conversion for cp equations
PsyT=Tc/100;
% Cp Calculation for Various Gases %
Cp_H2 =1/Mass1(1)*(56.505  - 702.74*(PsyT.^(-.75)) + 1165  *(PsyT.^(-1 )) - 560.7  *(PsyT.^(-1.5)));
Cp_CH4=1/Mass1(2)*(-672.87 + 439.74*(PsyT.^(.25 )) - 24.875*(PsyT.^(.75)) + 323.88 *(PsyT.^(-.5 )));
Cp_CO2=1/Mass1(3)*(-3.7357 + 30.529*(PsyT.^(.5  )) - 4.1034*(PsyT.^(1  )) + .024198*(PsyT.^(2   )));
Cp_H2O=1/Mass1(4)*(143.05  - 183.54*(PsyT.^(.25 )) + 82.751*(PsyT.^(.5 )) - 3.6989 *(PsyT.^(1   )));
%Gama Calculations for Various Gases%
Gama_H2 =Cp_H2./(Cp_H2 -(8.314/Mass1(1)));
Gama_CH4=Cp_CH4./(Cp_CH4-(8.314/Mass1(2)));
Gama_CO2=Cp_CO2./(Cp_CO2-(8.314/Mass1(3)));
Gama_H2O=Cp_H2O./(Cp_H2O-(8.314/Mass1(4)));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ghost2==1
   gama=Gama_H2;
   Mass=Mass1(1);
elseif ghost2==2
   gama=Gama_CH4;
   Mass=Mass1(2);
elseif ghost2==3
   gama=Gama_CO2;
   Mass=Mass1(3);
elseif ghost2==4
   gama=Gama_H2O;
   Mass=Mass1(4);
else
   fprintf('\n\tYou are stupid.')
```

```
end
```

45.1.1.2 Function for numerical determination of Expansion ratio (for TNRE.m)

```
function M=Mach(gama,EpsX)
%EpsE=max(geom(:,5))/min(geom(:,5));
MH=6;
error=1;
Step=.0001;
while error >=.001
   MH=MH-Step;
   EpsH=1/MH*(((2/(gama+1))*(1+(gama-1)/2*MH^2))^((gama+1)/((gama-1))))^.5;
   error=(EpsH-(EpsX))/EpsX;
end
M=(MH);
```

45.1.1.3 Subprogram P_core (for TNRE.m)

```
%%for ghost1 main program
%%Calculates Number of Cores and Power of each core
%%Core(i,j)=[power of core, number of engines, m_dot of engines, Volume per core, Energy per core,
Pipe Area, Core Mass]

Mass1=[2.016,16.043,44.01,18.015];
%%%%%%%%%%%%%%%%%%%%%%
clc
if ghost2==1
   Core(1,1)=(.018061*Tc-5.715417)*Core(1,3);
   Mass=Mass1(1);
elseif ghost2==2
   Core(1,1)=(.008264*Tc-4.374854)*Core(1,3);
   Mass=Mass1(2);
elseif ghost2==3
   Core(1,1)=(.002246*Tc-.670951)*Core(1,3);
   Mass=Mass1(3);
elseif ghost2==4
   Core(1,1)=(.003039*Tc-.162990)*Core(1,3);
   Mass=Mass1(4);
else
   fprintf('\n\tYou are stupid.')
end
Core(1,2)=ceil(Core(1,1)/2000);
```

45.1.1.4 Subprogram V_Pcore (for TNRE.m)

```
%%Calculates Volume of core and radius and height of core
%%Core(i,j)=[power of core, number of engines, m_dot of engines, Volume per core, Energy per core,
Pipe Area, Core Mass]
f_refl=1.2;

Core(1,4)=Core(1,1)/Core(1,2)*(6.4e-19*tb+1/Pd);   %Volume per Core
Core(1,5)=Core(1,1)/Core(1,2)*tb;                  %Energy Per Core
%Neutrons, Mass, Volume Consumed per core
CoreCons(1,1)=Core(1,5)/3.206e-11;
CoreCons(1,2)=.235*CoreCons(1,1)/.6023e24;
CoreCons(1,3)=CoreCons(1,2)/19600;

%Rcore=linspace(.1,5,50);
%for R_core=1:50
 %  B(R_core)=pi^2*((R_core/10)^4+(2.405*Core(1,4)/(f_refl*pi^2*(R_core/10))^2))^.5/Core(1,4);
 %  Hcore(R_core)=pi*(B(R_core)^2-(2.405/f_refl/(R_core/10))^2)^(-.5);
%end


%Dimensions of per core
R_core=(Core(1,4)/Core(1,2)/2/pi)^(1/3);%.35;
H_core=R_core*2;%Core(1,4)/Core(1,2)/pi/R_core^2;

%Cross Sectional Area of Propellant
rho=max(Pc)*Mass/(Ru*Tc);
Core(1,6)=Core(1,3)/Core(1,2)/(.2/rho*sqrt(gama*8314/Mass*Tc))/500;
Radius=sqrt(Core(1,6)/pi);
```

```
%Mass of Engines
density=2300; %kg/m^3
Core(1,7)=Core(1,4)*density*Core(1,2);
```

45.1.1.5 Figure subprogram (for TNRE.m)

```
%%plots figures for use
figure(1)
subplot(221)
plot(EpsX,Isp)
xlabel('Eps')
ylabel('Isp')
grid

subplot(222)
plot(EpsX,TotMass)
xlabel('Eps')
ylabel('Mass')
grid

subplot(223)
plot(r,O_gama_r)
hold on
plot(r,FNF)
plot(r,Radiation)
hold off
xlabel('Radius (cm)')
ylabel('Gama Flux')
grid
```

45.1.1.6 Output table (for TNRE.m)

```
%%Core(i,j)=[power of core, number of engines, m_dot of engines, Volume per core, Energy per core,
Pipe Area, Core Mass]
clc
if ghost2==1
   fprintf('\nFuel type is --------- =     H_2')
elseif ghost2==2
   fprintf('\nFuel type is --------- =    CH_4')
elseif ghost2==3
   fprintf('\nFuel type is --------- =    CO_2')
elseif ghost2==4
   fprintf('\nFuel type is --------- =    H_2O')
else
   fprintf('\n\tYou are stupid.')
end
fprintf('\nMax Isp for engines -- = %7.0f sec',max(Isp))
fprintf('\nTotal Thurst --------- = %7.3f kN',Thrust/1000)
fprintf('\nExpansion ratio ------ = %7.0f ',max(EpsX))
fprintf('\nTotal m_dot of engines = %7.2f kg/sec',Core(1,3))
fprintf('\n')
fprintf('\nChamber Pressure ----- = %7.1f Mpa',Pc/1000000)
fprintf('\nChamber Temperature -- = %7.0f K',Tc)
fprintf('\nNumber of engines ---- = %7.0f',Core(1,2))
fprintf('\n')
fprintf('\nTotal burn time ------ = %7.0f sec',tb)
fprintf('\nTotal mass of cores -- = %7.2f kg',Core(1,7))
fprintf('\nTotal Energy of cores  = %7.0f MW-sec',Core(1,5))
fprintf('\nTotal Power Output --- = %7.0f MW',Core(1,1))
fprintf('\nRems produced by TNRE  = %7.0f x 10^3 rem',Rems/1000)
fprintf('\n')
fprintf('\nTotal Mass Per Engine  = %7.2f kg',max(TotMass))
fprintf('\nm_dot Per engine ----- = %7.2f kg/sec',Core(1,3)/Core(1,2))
fprintf('\nVolume per engine ---- = %7.3f m^3',Core(1,4))
fprintf('\n')
fprintf('\nArea Throat ---------- = %7.3f m^2',At)
fprintf('\nArea Exit ------------ = %7.3f m^2',max(Aet))
fprintf('\nRadius Throat -------- = %7.3f m',Rt)
fprintf('\nRadius exit ---------- = %7.3f m',max(Ret))
fprintf('\nNozzle length -------- = %7.4f m',max(NozLength))
fprintf('\nTotal length --------- = %7.4f m',TotalLength)
fprintf('\n')
fprintf('\nFuel Mass ------------ = %7.0f kg',FuelMass)
fprintf('\n\n')
```

45.1.1.7 Research done for TNRE

This section is taken from www.ucar.edu/eo/staff/dward/sao/fit/nuclear.htm. It shows different possible propulsion systems.

Rocket Type	Isp	Thrust range
Solid Core Nuclear Thermal Rockets	850-1,000	High
Nuclear-powered Electric Rockets	2,000-3,000	Low
Gas Core Nuclear Thermal Rockets	1,500-3,000	High
Nuclear Pulse Rockets	>10,000	Extremely High
Nuclear Salt Water Rockets	5,000-100,000	Very High
Nuclear Fission-fragment Rockets	1,000,000	Moderate

This is a list of links used in the Transport analysis.

<http://www.fas.org/nuke/space/c04rover.htm>

<http://www.marsacademy.com/propul5.htm>

<http://www.astronautix.com/props/nucarlh2.htm>

<http://www.astronautix.com/articles/sovermal.htm>

<http://sfwrg.org/n003.html>

<http://www.astrodigital.org/space/nuclear.html>

<http://www.atomicinsights.com/sep95/rocket_programs.html>

<http://www.beyondearthorbit.org/rocketsnuclear.html>

<http://www.inspi.ufl.edu/tutorial/Nuclear_Propulsion/>

<http://www.nuclearspace.com/A_PWrussview_FINX.htm>

<http://internet.cybermesa.com/~mrpbar/rocket.html>

<https://engineering.purdue.edu/AAE/Courses/CoursePages/aae450/2005/spring/>

45.2 MHV entry analysis

Author(s): Robert Yochum

This analysis was performed separately to determine entry of the MHV. A program was thus written. It is based off Aircraft dynamics.

45.2.1 MHV.m Program: A program for entry of MHV.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Robert Yochum      Created: Feb 10 2005         Edited: Mar 09,'2005   %
%                                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
clc

%%variables - definition
LDRatio=1;
S=75;
Cd=.7;
Cl=Cd*LDRatio;
Lc=.1;
M=.1;
N=.1;
m=100000;
R=192;

J11=10;
J22=10;
J33=10;
J13=5;
angle=8; %entry angle
alpha=15;  %angle of attack
Vel=3500;
%----------------inputs----------------%
p1=0;
p2=0;
h=100000;
thi=0;
theda=alpha*pi/180;
psy=0;
u=Vel*cos(angle*pi/180);
v=0;
w=Vel*sin(angle*pi/180);
p=0;
q=0;
r=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


x0=[p1;p2;h;thi;theda;psy;u;v;w;p;q;r];

[t,x]=ode45('EOMFlight',[0 1500],x0);

Tc=210;
Mass1=[2.016,16.043,44.01,18.015];
PsyT=Tc/100;
Cp_CO2=1/Mass1(3)*(-3.7357 + 30.529*(PsyT.^(.5  )) - 4.1034*(PsyT.^(1  )) + .024198*(PsyT.^(2   )));
Gama=Cp_CO2./(Cp_CO2-(8.31441/Mass1(3)));
V=2*sqrt(Gama*R*Tc);


figure(1)
subplot(211)
hold on
plot(x(:,1),x(:,3),'g')
%plot(x(:,2),x(:,3))
%plot(sqrt(x(:,1).^2+x(:,2).^2),x(:,3),'r')%x velocity
title('range vs altitude')
xlabel('range')
ylabel('altitude')
hold off
grid
```

```
subplot(212)
hold on
plot(sqrt(x(:,1).^2+x(:,2).^2),x(:,7),'g')%x velocity
plot(sqrt(x(:,1).^2+x(:,2).^2),x(:,8),'k')
plot(sqrt(x(:,1).^2+x(:,2).^2),x(:,9))    %vertical velocity
plot(sqrt(x(:,1).^2+x(:,2).^2),sqrt(x(:,7).^2+x(:,9).^2),'r')
plot(sqrt(x(:,1).^2+x(:,2).^2),V)
xlabel('range')
ylabel('velocity')
grid
```

45.2.1.1 EOM function for MHV.m program The function is based off EOMs of aircraft. Moments are assumed to be zero.

```
function dx=EOMFlight(t,x)
%%Definitions
variables;

dx=zeros(12,1);

%%position
dx(1)=cos(x(6))*cos(x(5))*x(7)  + (-sin(x(6))*cos(x(4))+cos(x(6))*sin(x(5))*sin(x(4)))*x(8)  + (
sin(x(6))*sin(x(4))+cos(x(6))*sin(x(5))*cos(x(4)))*x(9);

dx(2)=sin(x(6))*cos(x(5))*x(7)  + ( cos(x(6))*cos(x(4))+sin(x(6))*sin(x(5))*sin(x(4)))*x(8) + (-
cos(x(6))*sin(x(4))+sin(x(6))*sin(x(5))*cos(x(4)))*x(9);

dx(3)= sin(x(5))*x(7)-cos(x(5))*sin(x(4))*x(8)-cos(x(5))*cos(x(4))*x(9);

%%roll rates

dx(4)=x(10)+tan(x(5))*sin(x(4))*x(11)+tan(x(5))*cos(x(4))*x(12);
dx(5)=              cos(x(4))*x(11)     -     sin(x(4))*x(12);
dx(6)=    (sin(x(4))/cos(x(5)))*x(11)+(cos(x(4))/cos(x(5)))*x(12);

%%velocity

dx(7)= x(12)*x(8)-x(11)*x(9)-(3.72*(3375000/(3375000+x(3))))*sin(x(5)) +        (-Cd*S*.5*((((228+(-
.003*x(3)))/228)^(-(3.72*(3375000/(3375000+x(3)))/(-.003*R))))*774)/(R*(228+(-
.003*x(3))))*(sqrt(x(7)^2+x(8)^2+x(9)^2))^2*cos(x(5))+Cl*S*.5*((((228+(-.003*x(3)))/228)^(-
(3.72*(3375000/(3375000+x(3)))/(-.003*R))))*774)/(R*(228+(-
.003*x(3))))*(sqrt(x(7)^2+x(8)^2+x(9)^2))^2*sin(x(5)))/m;

dx(8)=-x(12)*x(7)+x(10)*x(9)+(3.72*(3375000/(3375000+x(3))))*cos(x(5))*sin(x(4));

dx(9)= x(11)*x(7)-x(10)*x(8)+(3.72*(3375000/(3375000+x(3))))*cos(x(5))*cos(x(4))+(-Cd*S*.5*((((228+(-
.003*x(3)))/228)^(-(3.72*(3375000/(3375000+x(3)))/(-.003*R))))*774)/(R*(228+(-
.003*x(3))))*(sqrt(x(7)^2+x(8)^2+x(9)^2))^2*sin(x(5))-Cl*S*.5*((((228+(-.003*x(3)))/228)^(-
(3.72*(3375000/(3375000+x(3)))/(-.003*R))))*774)/(R*(228+(-
.003*x(3))))*(sqrt(x(7)^2+x(8)^2+x(9)^2))^2*cos(x(5)))/m;
%%Moments
dx(10)=0;%(J22-J33)*x(11)*x(12)/J11+J13*x(10)*x(11)/J11+Lc/J11+J13*dx(12)/J11;
dx(11)=0;%(J33-J11)*x(10)*x(12)/J22+J13/J22(x(12)^2-x(10)^2)+M/J22;
dx(12)=0;%(J11-J22)*x(10)*x(11)/J33-J13*x(11)*x(12)/J33+N/J33+J13/J33*dx(10);

return
```

46 Yoke, Jeff

46.1 Appendix

Author: Jeffrey Yoke

Contributors: Chris Cunha, Aaron Hauser

46.1.1 Evolution of Mars Lander Vehicle (MLV) Design

The overall design of the MLV underwent a large number of changes throughout the duration of the design process. It began as a very crude drawing based off of the Lunar Module. This representation can be seen in Figure 46-1.

46.1.1.1 Initial Representation – Based off of Lunar Module design [1]

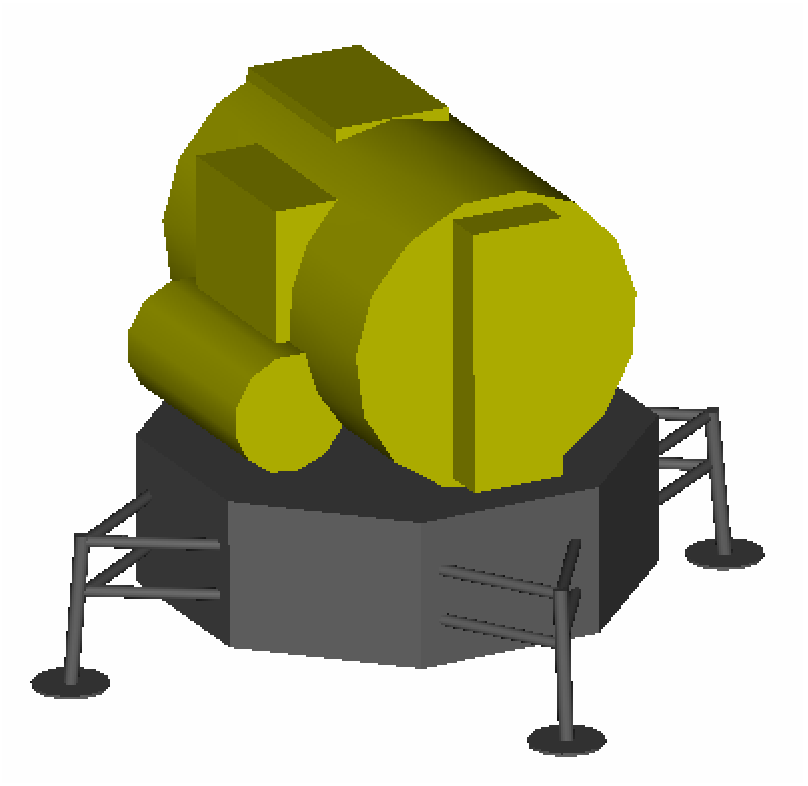


Figure 46-1 Initial MLV Design

- Crew Compartment – Gold; 22 m^3 pressurized volume
- Descent engine housing – Black
- Total Launch Mass = 58,000 kg
- Skin is made from milled aluminum and titanium fasteners

46.1.1.2 Evolution of the Crew Compartment

We, as a class, realized that this mass was much too high for the mission to be a success. This caused us to find ways to reduce the mass of the MLV without putting the crew at an even greater risk. The first step we took was to change the material that the MLV was being made of. By changing the skin from aluminum to a carbon composite, we were able to save a large amount of mass. This is due to the density of the carbon composite being 1550 kg/m^3 as compared to 2750 kg/m^3 for the aluminum alloy being used. Since the composite material behaves very well under the same conditions as the aluminum there was not a drastic drop off in material properties and thus the thickness of the skin was able to remain about the same.

The next step was to find a way to better use the interior volume of the cylindrical crew compartment. As can be seen in Figure 46-1, the Lunar Module had a crew compartment that consisted mainly of a large pressurized cylinder. We found it to be more conducive to fitting all the astronauts and their equipment comfortably to use a vertical cylinder. The initial thought was to solely use a vertical cylinder, but this led to aerodynamic problems when returning to the Crew Transport Vehicle (CTV). This led us to the decision to place a rounded top on the cylinder. Whether it is a cone or a blunter object, there needed to be some sort of aerodynamic body on the top of the crew compartment. It made sense to include this in the pressurized volume so to not have any excess structural mass. An overall structural mass of a crew compartment with a pressurized volume of 22m^3 can be seen in Figure 46-2 and was produced using the following code:

```
% Jeffrey Yoke
% AAE450
% Lander mass

clear all
close all
clc

rho_air=1.225;             % Density of air in lander, kg/m^3
sigma_yield=240;           % Yield stress of aluminum, GPa
rho_alum=2750;             % Density of aluminum, kg/m^3
h_cyl=1.417;               % Height of cylindrical portion of lander, m
radius=2;                  % Radius of cylinder and cone, m
h_cone=1;                  % Height of conical portion of lander, m
l_leg=1;                   % Length of Leg, m
P_tank=1;                  % Tank pressure, atm
launch_g=10;               % Max g load during booster launch
safety_factor=2;           % Factor of safety

vol_total=22;                           % Total pressurized volume, m^3
vol_cyl=vol_total-pi*radius^2*h_cone/3  % m^3
vol_cone=vol_total-pi*radius^2*h_cyl    % m^3
air_mass=rho_air*vol_total              % Mass of air in crew cabin, kg
P_max=air_mass*9.8*launch_g/(pi*radius^2)+P_tank*101325   % Pressure during launch, Pa

cyl_thick=safety_factor*sqrt(.75*P_max^2*radius^2/((sigma_yield*10^6)^2))  % m
```

```
% Minimum thickness and mass for lander (however, 5mm was chosen for ascent to ensure safety)
mass_desc=(pi*(radius+.01)^2*(1.02)-pi*(radius)^2*(1))*rho_alum
mass_desc_composite=(pi*(radius+.01)^2*(1.02)-pi*(radius)^2*(1))*1550
% Therefore, thickness initially set at 5 mm
thick=.005
cyl_mass_5mm=(pi*(radius+thick)^2*(h_cyl+thick)-vol_cyl)*rho_alum
cone_mass_5mm=((pi*(radius+thick)^2*(h_cone+thick)/3)-vol_cone)*rho_alum
leg_mass=100;
total_mass_5mm=cyl_mass_5mm+cone_mass_5mm+mass_desc+leg_mass  %Structural mass if thickness of ascent
is 5mm
total_mass_5mm_comp=(cyl_mass_5mm+cone_mass_5mm+mass_desc+leg_mass)/rho_alum*1550
% Tested with thickness of 1cm for comparison
thick2=.01
cyl_mass_1cm=(pi*(radius+thick2)^2*(h_cyl+thick2)-vol_cyl)*rho_alum
cone_mass_1cm=((pi*(radius+thick2)^2*(h_cone+thick2)/3)-vol_cone)*rho_alum
total_mass_1cm=cyl_mass_1cm+cone_mass_1cm+mass_desc+leg_mass  %Structural mass if thickness of ascent
is 1cm
total_mass_1cm_comp=(cyl_mass_1cm+cone_mass_1cm+mass_desc+leg_mass)/rho_alum*1550
% Tested at minimum of 1mm
cyl_mass_1mm=(pi*(radius+cyl_thick)^2*(h_cyl+cyl_thick)-vol_cyl)*rho_alum
cone_mass_1mm=((pi*(radius+cyl_thick)^2*(h_cone+cyl_thick)/3)-vol_cone)*rho_alum
total_mass_1mm=cyl_mass_1mm+cone_mass_1mm+mass_desc+leg_mass  %Structural mass if thickness of ascent
is 1mm
total_mass_1mm_comp=(cyl_mass_1mm+cone_mass_1mm+mass_desc+leg_mass)/rho_alum*1550
x=[1,5,9]; % Thicknesses, mm
mass=[total_mass_1mm,total_mass_5mm,total_mass_1cm]; % Total Masses for set thickness, kg
bar(x,mass)
title('Mars Lander Vehicle Mass v. Thickness for 1mm,5mm,10mm')
ylabel('Mass (kg)')
hold
mass_comp=[total_mass_1mm_comp,total_mass_5mm_comp,total_mass_1cm_comp];
x_comp=[.5,4.5,8.5];
bar(x_comp,mass_comp,'r')
legend('Aluminum','Carbon Composite',2)
```

This code uses the following inputs: ρ_{air} (density of air, kg/m^3), σ_y (yield strength of material, GPa), $\rho_{aluminum}$ (density of aluminum, kg/m^3), $h_{cylnder}$ (height of the cylinder, m), h_{cone} (height of the cone, m), r (radius of the cylinder and cone, m), P_{tank} (pressure in the compartment, atm), l_{leg} (length of the legs, m). It should be noted that these parameters were the values being used at this stage of the design and do not apply to the final design. The code outputs the total mass of the crew compartment for both materials when skin thicknesses of 1mm, 5mm, and 10mm are looked at. These were chosen to show the effect of increasing the thickness above the minimum allowable.

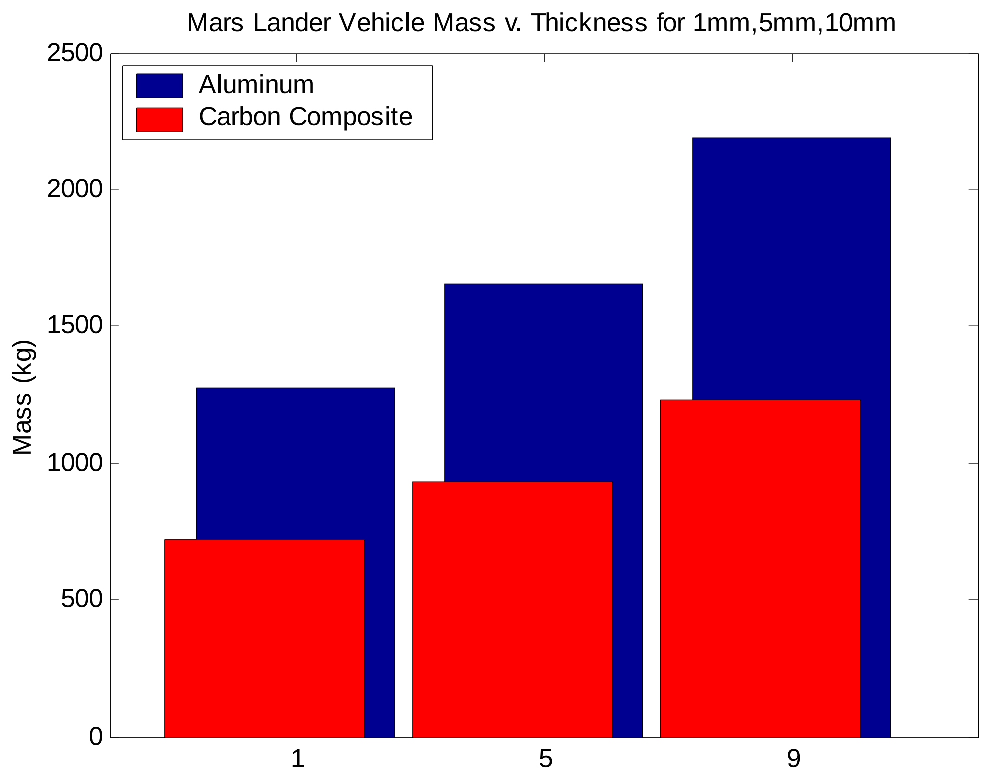


Figure 46-2 MLV Mass vs. Skin Thickness

The next step in reducing the mass was to reduce the overall volume of the crew compartment. A study was conducted to show the effect that increasing the skin thickness had on pressurized volumes of 15m^3 and 22m^3. As can be seen in Figure 46-3, the rate at which the mass increases is greater for larger volumes with respect to thickness. This fact aids in the decision to make the crew compartment as small as feasibly possible. The parameters in Table 46-1 were used for the two cases looked at.

Table 46-1 Trade Study Parameters

Volume (m^3)	Radius (m)	$h_{cylinder}$ (m)	h_{cone} (m)
15	1.50	1.78	1.03
22	2.00	1.42	1.00

By using these parameters as well as the following code, we are able to show the aforementioned relationship. The MATLAB code outputs Figure 46-3 that shows the mass relationship for the crew compartment for skin thicknesses varying from 1mm to 15mm.

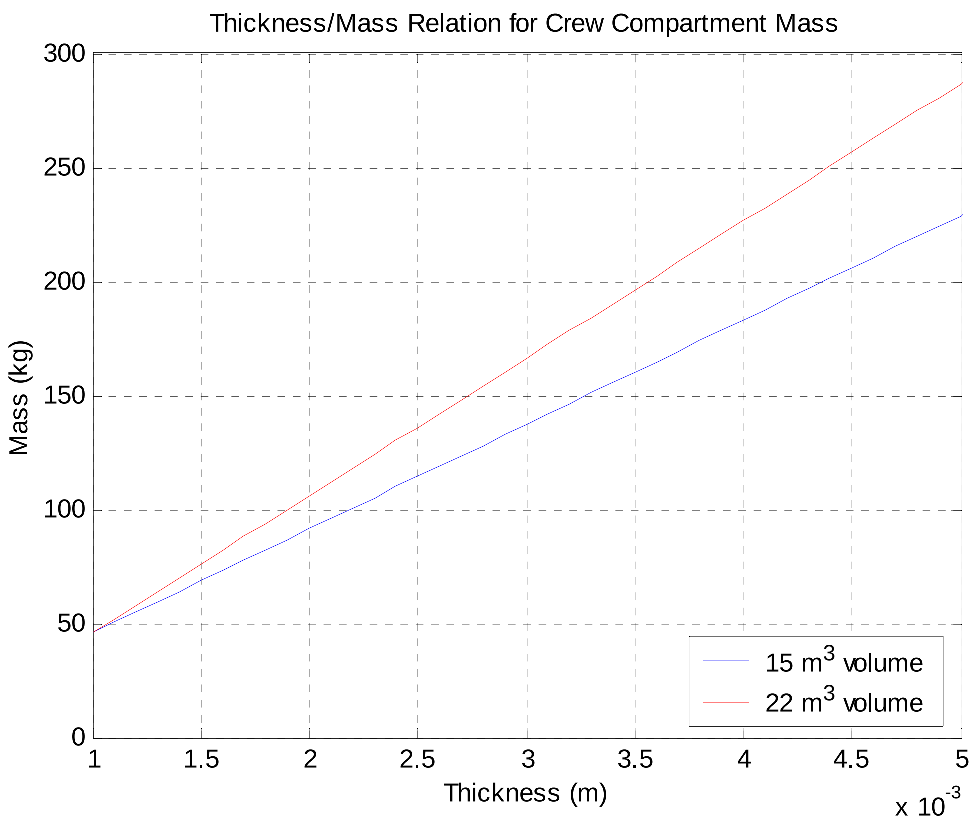


Figure 46-3 Thickness/Mass Relation for Crew Compartment

```
% Jeffrey Yoke
% AAE450
% Lander mass

clear all
close all
clc

%%%%%%%%%%%%%%%%%%%%%%%%% 15 m^3 Volume%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rho_air=1.225;              % Density of air in lander, kg/m^3
sigma_yield=240;            % Yield stress of aluminum, GPa
rho_comp=1550;              % Density of carbon composite, kg/m^3
h_cyl=1.78;                 % Height of cylindrical portion of lander, m
radius=1.5;                 % Radius of cylinder and cone, m
h_cone=1.03;                % Height of conical portion of lander, m
l_leg=1;                    % Length of Leg, m
P_tank=1;                   % Tank pressure, atm
launch_g=6;                 % Max g load during booster launch
safety_factor=1.2;          % Factor of safety

vol_total=15;                        % Total pressurized volume, m^3
vol_cyl=pi*radius^2*h_cyl  ;         % m^3
vol_cone=pi*radius^2*h_cone/3;       % m^3
air_mass=rho_air*vol_total;          % Mass of air in crew cabin, kg
P_max=air_mass*9.8*launch_g/(pi*radius^2)+P_tank*101325   % Pressure during launch, Pa

% Minimum thickness and mass for lander (however, 5mm was chosen for ascent to ensure safety)
R_desc=2;
end_thick=safety_factor*sqrt(.25*P_max^2*R_desc^2/((sigma_yield*10^6)^2))  % m
cyl_thick=safety_factor*sqrt(3)*P_max*radius/(sigma_yield*10^6)            % m

r_desc=2;
R_desc=1.5;
```

```
h_desc=4;
mass_desc_frus=(pi*((r_desc+.0025)^2+(R_desc+.0025)*(r_desc+.0025)+(R_desc+.0025)^2)*(h_desc+.0025)/3-
pi*((r_desc)^2+(R_desc)*(r_desc)+(R_desc)^2)*(h_desc)/3)*rho_comp;
mass_desc_sph=(4/6*pi*(R_desc+.0025)^2-4/6*pi*R_desc^2)*rho_comp;
mass_desc=mass_desc_frus+mass_desc_sph
% Therefore, thickness initially set at 5 mm
thick=.0025;
cyl_mass_5mm=(pi*(radius+thick)^2*(h_cyl+2*thick)-vol_cyl)*rho_comp;              % kg
cone_mass_5mm=((pi*(radius+thick)^2*(h_cone+thick)/3)-vol_cone)*rho_comp          % kg
ascent_mass_5mm=cyl_mass_5mm%+cone_mass_5mm                                       % kg
leg_mass=50;                                                                      % kg
total_mass_5mm=cyl_mass_5mm+cone_mass_5mm+mass_desc+leg_mass  %Structural mass if thickness of ascent
stage is 5mm

% Tested with thickness of 1cm for comparison
thick2=.01;
cyl_mass_1cm=(pi*(radius+thick2)^2*(h_cyl+thick2)-vol_cyl)*rho_comp;              % kg
cone_mass_1cm=((pi*(radius+thick2)^2*(h_cone+thick2)/3)-vol_cone)*rho_comp;       % kg
ascent_mass_1cm=cyl_mass_1cm+cone_mass_1cm                                        % kg
total_mass_1cm=cyl_mass_1cm+cone_mass_1cm+mass_desc+leg_mass  %Structural mass if thickness of ascent
stage is 1cm

% Tested at minimum of 1mm
cyl_mass_1mm=(pi*(radius+cyl_thick)^2*(h_cyl+cyl_thick)-vol_cyl)*rho_comp;        % kg
cone_mass_1mm=((pi*(radius+cyl_thick)^2*(h_cone+cyl_thick)/3)-vol_cone)*rho_comp; % kg
ascent_mass_1mm=cyl_mass_1mm+cone_mass_1mm                                        % kg
total_mass_1mm=cyl_mass_1mm+cone_mass_1mm+mass_desc+leg_mass  %Structural mass if thickness of ascent
stage is 1mm

t=[.001:.0001:.015];
cyl_mass_vec=(pi.*(radius+t).^2.*(h_cyl+t)-vol_cyl).*rho_comp;          % kg
cone_mass_vec=((pi.*(radius+t).^2.*(h_cone+t)./3)-vol_cone).*rho_comp;
ascent_mass_vec=cyl_mass_vec+cone_mass_vec;
plot(t,ascent_mass_vec)
grid on

%%%%%%%%%%%%%%%%%%%%%%%%% 22 m^3 Volume%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('-------------22 m^3-------------')
h_cyl=1.417;              % Height of cylindrical portion of lander, m
radius=2;                 % Radius of cylinder and cone, m
h_cone=1;                 % Height of conical portion of lander, m

vol_total=22;                            % Total pressurized volume, m^3
vol_cyl=vol_total-pi*radius^2*h_cone/3;  % m^3
vol_cone=vol_total-pi*radius^2*h_cyl;    % m^3
air_mass=rho_air*vol_total;              % Mass of air in crew cabin, kg
P_max=air_mass*9.8*launch_g/(pi*radius^2)+P_tank*101325     Pressure during launch, Pa

cyl_thick=safety_factor*sqrt(.75*P_max^2*radius^2/((sigma_yield*10^6)^2))  % m

r_desc=3;
R_desc=2;
h_desc=4;
mass_desc_frus=(pi*((r_desc+.005)^2+(R_desc+.005)*(r_desc+.005)+(R_desc+.005)^2)*(h_desc+.005)/3-
pi*((r_desc)^2+(R_desc)*(r_desc)+(R_desc)^2)*(h_desc)/3)*rho_comp;
mass_desc_sph=(4/6*pi*(R_desc+.005)^2-4/6*pi*R_desc^2)*rho_comp;
mass_desc=mass_desc_frus+mass_desc_sph
% Therefore, thickness initially set at 5 mm
thick=.005;
cyl_mass_5mm=(pi*(radius+thick)^2*(h_cyl+thick)-vol_cyl)*rho_comp;                % kg
cone_mass_5mm=((pi*(radius+thick)^2*(h_cone+thick)/3)-vol_cone)*rho_comp;         % kg
ascent_mass_5mm=cyl_mass_5mm+cone_mass_5mm                                        % kg
leg_mass=100;                                                                     % kg
total_mass_5mm=cyl_mass_5mm+cone_mass_5mm+mass_desc+leg_mass  %Structural mass if thickness of ascent
is 5mm

% Tested with thickness of 1cm for comparison
thick2=.01;
cyl_mass_1cm=(pi*(radius+thick2)^2*(h_cyl+thick2)-vol_cyl)*rho_comp;              % kg
cone_mass_1cm=((pi*(radius+thick2)^2*(h_cone+thick2)/3)-vol_cone)*rho_comp;       % kg
ascent_mass_1cm=cyl_mass_1cm+cone_mass_1cm                                        % kg
```

```
total_mass_1cm=cyl_mass_1cm+cone_mass_1cm+mass_desc+leg_mass  %Structural mass if thickness of ascent
is 1cm

% Tested at minimum of 1mm
cyl_mass_1mm=(pi*(radius+cyl_thick)^2*(h_cyl+cyl_thick)-vol_cyl)*rho_comp;        % kg
cone_mass_1mm=((pi*(radius+cyl_thick)^2*(h_cone+cyl_thick)/3)-vol_cone)*rho_comp; % kg
ascent_mass_1mm=cyl_mass_1mm+cone_mass_1mm                                        % kg
total_mass_1mm=cyl_mass_1mm+cone_mass_1mm+mass_desc+leg_mass  %Structural mass if thickness of ascent
is 1mm

hold on
cyl_mass_vec=(pi.*(radius+t).^2.*(h_cyl+t)-vol_cyl).*rho_comp;          % kg
cone_mass_vec=((pi.*(radius+t).^2.*(h_cone+t)./3)-vol_cone).*rho_comp;
ascent_mass_vec=cyl_mass_vec+cone_mass_vec;
plot(t,ascent_mass_vec,'r')
xlabel('Thickness (m)')
ylabel('Mass (kg)')
title('Thickness/Mass Relation for Crew Compartment Mass')
legend('15 m^3 volume','22 m^3 volume',0)
axis([.001 .005 0 300])
```

The inputs for the code are ρ_{air} (density of air, kg/m^3), σ_y (yield strength of material, GPa), ρ_{comp} (density of carbon composite, kg/m^3), $h_{cylnder}$ (height of the cylinder, m), h_{cone} (height of the cone, m), r (radius of the cylinder and cone, m), P_{tank} (pressure in the compartment, atm), and l_{leg} (length of the legs, m). The code calculates the overall mass of the entire MLV by adding the masses for each section that it has calculated. The masses are calculated by finding the structural volume of each section and then multiplying that by the density of the carbon composite.

A visual representation of the MLV at this point in the design can be seen in Figure 46-4. In this figure, the lower portion of the vehicle represents the portion where both the descent and ascent engines are housed, while the upper half is the pressurized crew compartment with an aerodynamic body located on top of a cylinder.

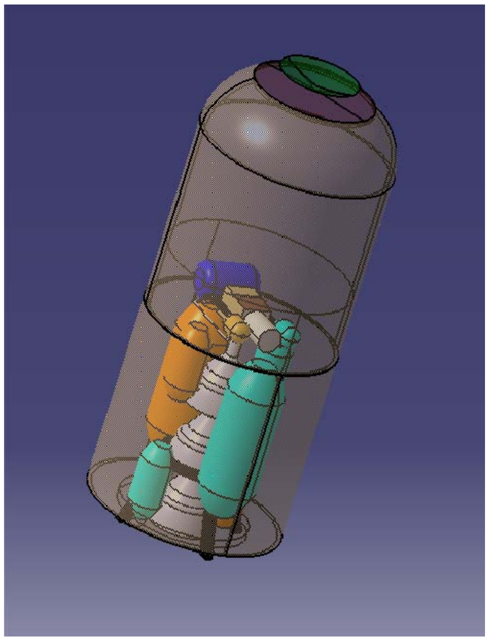


Figure 46-4 15m^3 MLV (Chris Cunha)

We continued to gradually decrease the pressurized volume of the crew compartment until it reached its final value of 9m^3. This was the minimum volume that could hold all of the necessary equipment as well as the astronauts and their suits.

It was at this point that we encountered the problem of damage to the structure from micrometeorites. A trade study was done to find a way to prevent any catastrophic damage from being created by these objects. Through this study, it was decided that the best way to counteract this possible damage was to have two thin-walled sections of skin separated by ribs. This allowed for a decrease in the thickness of the wall since the ribs would be able to carry the compressive loads that were before only being carried by the skin of the MLV.

The final structural design of the crew compartment consists of a tapered cylinder with a bottom radius of 1.315m and a top radius of 1.3m. There is a rounded top section above the cylinder giving the crew compartment an overall height of 1.7m. In the middle of the top there is a hatch to allow the astronauts easy movement to and from the MLV.

46.1.1.3 Evolution of the Engine Structure

As can be seen in Figure **46-1**, the engine structure began as two very separate portions for the ascent and descent stages. All of the tanks and hardware for the descent engine was originally housed in the octagonal black portion of the figure, while the ascent engine was located on one side of the gold section. As we changed the shape and size of the crew compartment, there needed to be changes made to the layout of the engine tanks and hardware as well. This led to the initial decision to place the crew compartment directly on top of the ascent engine structure and then place that on top of the descent engine structure. A representation of the first iteration of this design can be seen in Figure 46-5 below.

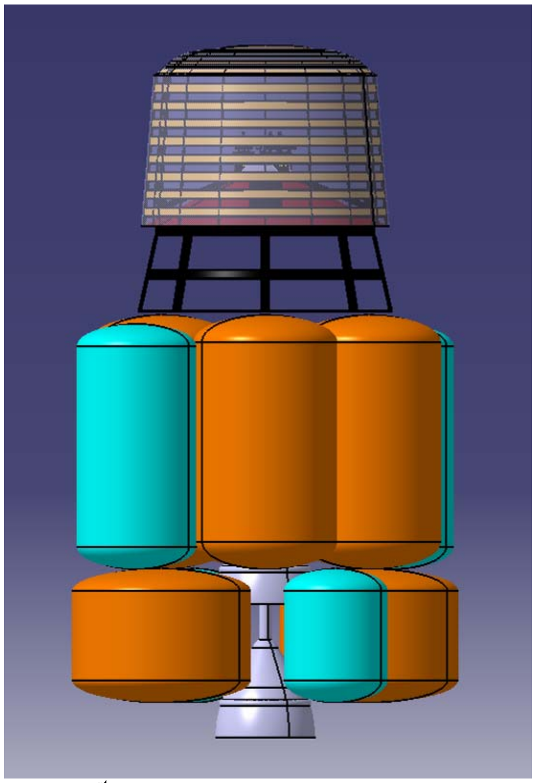


Figure 46-5 1st Iteration to Engine Layout (Chris Cunha)

As you can see, we made it so that the tanks to hold the oxidizer and fuel were the same height in the ascent stage as well as in the descent stage. Also, the propellants were split evenly between tanks so there were not two massive tanks per stage taking up very large amounts of volume. Table 46-2 shows the breakdown of volumes at this point in the design process.

Table 46-2 Tank Size Breakdown (Per Tank)

Stage	h_{LOX} (m)	r_{LOX} (m)	h_{LH} (m)	r_{LH} (m)

Ascent	2.75	0.72	2.75	0.867
Descent	1.5	0.59	1.5	1.015

The actual engine hardware is placed in the middle of the tanks to allow for a symmetrical arrangement of the tanks helping to keep the MLV balanced. The location of the actual engine hardware has remained the same since this point in the design.

As the overall mass of the MLV continued to increase, there became a need for more propellant and thus larger tanks. This created a problem in keeping the major and minor radii of the structure small enough to be able to fit in the descent aeroshell and then into the ELV. To fix this problem, we made the tanks taller as can be seen in Figure 46-6. As you can see, for the ascent stage, the oxidizer and fuel tanks are no longer the same height.

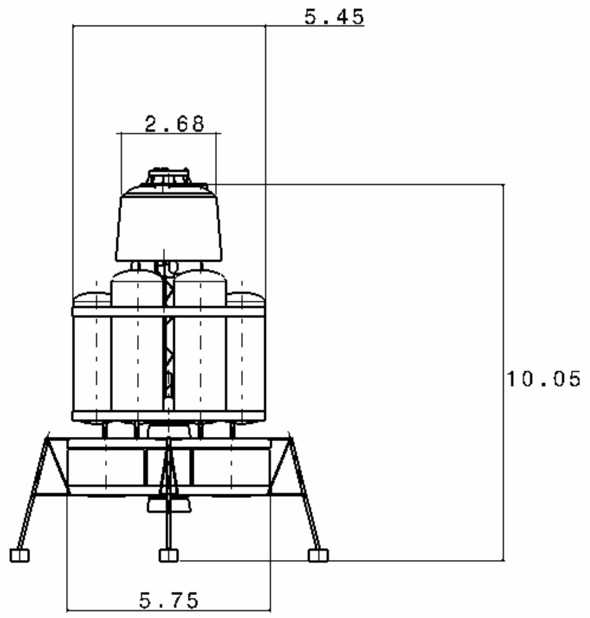


Figure 46-6 Final Engine Layout (Chris Cunha)

The taller tanks of the ascent stage are the four hydrogen tanks. In this picture, you can also see that there is a support structure now going around the outside of the tanks. The structures are the plates that are underneath each set of tanks and near or at the top of each set to hold the tanks in place. A breakdown of the new dimensions of the tanks can be seen in Table 46-3.

Table 46-3 Final Tank Dimensions

Stage	h_{LOX} (m)	r_{LOX} (m)	h_{LH} (m)	r_{LH} (m)
Ascent	3.5	0.2	4.1	0.375
Descent	1.5	0.63	1.5	1.03

Between the top plate of the descent stage and the bottom plate of the ascent stage, there are six support rods. These rods allow for ample separation between the engine hardware of each stage. When it is time to return to the CTV, there are small electro-explosive charges that go off breaking the connection between the supports and the ascent stage. This is similar to what was done during the Apollo missions to free the ascent stage [1].

46.1.1.4 Landing Gear

From the beginning, we planned to use some version of "spider" legs to support the MLV when it is on the surface of Mars. The main problem that was encountered during the design of the legs was how to cushion the landing so that the legs did not break off. This led to the decision to use some type of shock absorber. We initially looked at the possibility of using a hydraulic shock similar to what is used on everyday cars and trucks. However, this type of system created a major problem with safe and easy attachment. Since the legs need to be stowed during the trip to mars, the shock would need to be able to deform a large amount to account for the change in the distance between connection points. It also created a weight problem as the size of shock that we would need is of a substantial mass.

This led us to look into alternate methods of cushioning the impact, which, in turn, led to the use of a honeycomb within the footpad, primary strut, and secondary strut of each leg. The honeycomb material crushes upon impact with the Martian surface and allows for the MLV to self-level [3]. It also has a large mass savings over the use of hydraulic shock absorbers. For instance, shock absorbers are made from some metal which would cause the density of the material to be greater than 2700 kg/m^3 if it were made of any common metals. The density of the honeycomb used is 700 kg/m^3. The following code calculates the total mass of the landing gear system. It also outputs the total mass of the honeycomb and the total mass of the composite tubing.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code is used to calculate the mass of the landing gear used on the MLV. %
% Honeycomb portion of the code is a modified version of the MHV leg mass code %
% written by Aaron Hauser.  Composite tube portion of the code written by Jeff %
% Yoke.                                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all

% Honeycomb

mb = 30000;                   % kg mass of vehicle
```

```
v = [10];                      % m/s velocity of descent
g = 9.8;                       % m/s^2 gravtiy
gm = .38*g;                    % m/s^2 martian gravity
Wb = mb*gm;                    % N martian weight of craft craft
ho = 4;                        % m height of center of mass
Ev = 1/2*mb*v.^2+Wb*ho         % J total energy dissipated

Fc = 6894.757*8000             % Pa Ultimate compressive stress
dh_s = .75;                    % m usable srtoke
he_s = 1;                      % m footpad thickness
we =  1/62.428*1000*44         % density
Espec = Fc*dh_s/he_s/we        % m^2/s^2 specific energy
strut_mass = Ev/Espec/gm*4     % kg mass of honeycomb

Fc = 6894.757*8000             % Pa Ultimate compressive stress
dh_p = 2/3*.3;                 % m usable srtoke
he_p = .3;                     % m footpad thickness
we =  1/62.428*1000*44         % density
Espec_pad = Fc*dh_p/he_p/we    % m^2/s^2 specific energy
pad_mass = Ev/Espec_pad/gm*4   % kg mass of honeycomb

tot_honey=strut_mass+pad_mass % Total mass of honeycomb for 4 legs

% Composite Tube

l=1;                           % m Length of Primary Strut
rho=1550;                      % kg/m^3 Density of Carbon Composite
r_in=.04;                      % m Inner Radius
r_out=.045;                    % m Outer Radius
v=pi*r_out^2*l-pi*r_in^2*l;    % m^3 Volume of Primary Strut
mass_around_honey=v*rho*4;     % kg Mass of Primary Strut Case
r_in2=.013;                    % m Inner Radius of other struts
r_out2=.014;                   % m Outer Radius of other struts
v_else=pi*r_out2^2*.75*l-pi*r_in2^2*.75*l;  % m^3 Volume of 0.75m Strut
mass_else1=24*v_else*1550;                  % kg Mass of 0.75m Struts
v_else2=pi*r_out2^2*2*l-pi*r_in2^2*2*l;     % m^3 Volume of 2m Strut
mass_else2=8*v_else2*1550;                  % kg Mass of 2m Struts
v_else3=pi*r_out2^2*l-pi*r_in2^2*l;         % m^3 Volume of 1m Strut
mass_else3=8*v_else3*1550;                  % kg Mass of 1m Struts
v_long=pi*r_out^2*1.5*l-pi*r_in^2*1.5*l;    % m^3 Volume of Main Upper Strut
mass_long=4*v_long*1550;                    % kg Mass of Main Upper Struts
tot_nonhoney=mass_around_honey+mass_else1+mass_else2+mass_else3+mass_long % kg Total Mass of Comosite
Tubing

landing_gear_mass=tot_honey+mass_around_honey+mass_else1+mass_else2+mass_else3+mass_long % kg Total
Mass of Landing Gear
```

Within this code, the inputs are: m_b (mass of the vehicle, kg), v (velocity of the vehicle at impact, m/s), g (gravity on Earth, m/s^2), g_m (gravity on Mars, m/s^2), W_b (weight of the craft on Mars, N), h_0 (height of the center of mass, m), d_h (usable stroke, m), h_e (height of honeycomb, m), and the inner and outer radii for the composite tubing in meters (r_{in} and r_{out}). The mass of the honeycomb is determined by a ratio of the total energy dissipated (E_v, J) to the specific energy of the honeycomb (E_{spec}, J). This relationship, as well as the equations for E_v and E_{spec}, can be seen in Equation $m_h = \dfrac{E_v}{E_{spec} * g_{Mars}}$ 46-1 though Equation $E_{spec} = \dfrac{F_c * d_h}{h_e * w_e}$ 46-3 respectively.

$$m_h = \frac{E_v}{E_{spec} * g_{Mars}} \qquad 46\text{-}1$$

$$E_v = \frac{1}{2} * m_{lander} * v^2 + W_{lander} * h \qquad 46\text{-}2$$

$$E_{spec} = \frac{F_c * d_h}{h_e * w_e} \qquad 46\text{-}3$$

The code then calculates the total mass of the composite tubing that encases the honeycomb and makes up the other portions of the landing gear. Table 46-4 shows the mass breakdown of the landing gear.

Table 46-4 Mass Breakdown of Leg System

	Honeycomb	Composite Tubing	Downlock	Total
Mass (kg)	75.7	26.3	5	107

The downlock mechanism ensures that once the landing gear is deployed, there is no way for the legs to return to their stowed position. This mechanism can be seen in Figure 46-7.

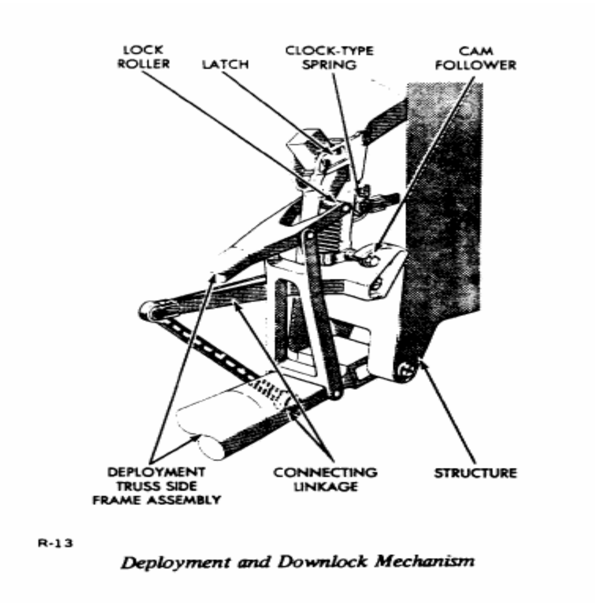


Figure 46-7 Downlock Mechanism [2]

46.1.1.5 Ladder System

For the astronauts to safely there needed to be a ladder system from the hatch on the crew compartment down to the Martian surface. On the crew compartment, this system is located on one side and consists of the two side supports and three rungs between these supports. The plates that are used to hold the propellant and oxidizer tanks in place can be used as steps as well. There are then two more side supports running from the top ascent engine plate to the bottom ascent engine plate. Due to the greater distance between these plates, there is a need for six rungs in this section. Once again, the tank housing plates can be used as steps. For the descent stage, there are once again the two side supports and three rungs just as with the crew compartment portion. The astronauts are then able to step down from the bottom descent plate to the surface of Mars due to the compression of the honeycomb and thus shortening of the legs. This ladder will also be used to climb back into the crew compartment when returning to the CTV. The following code was used to determine the overall mass of the ladder system as well as the volume that the system takes up.

```
% Jeffrey Yoke
% AAE450
% Ladder mass

clear all
close all
clc

rho=1550;   % Density of Composite Used, kg/m^3
rout=.034;  % Outer Radius of Tube, m
rin=.029;   % Inner Radius of Tube, m
L1=2.75;    % Length of Ascent Ladder Support, m
L2=1.7;     % Length of Crew Compartment Support, m
L3=.5;      % Length of Rung, m
L4=1.5;     % Length of Descent Support, m

m1=(pi*rout^2*L1-pi*rin^2*L1)*rho*2;  % Mass of Ascent Ladder Supports, kg
m2=(pi*rout^2*L2-pi*rin^2*L2)*rho*2;  % Mass of Crew Compartment Supports, kg
m3=(pi*rout^2*L3-pi*rin^2*L3)*rho*12; % Mass of Rungs, kg
m4=(pi*rout^2*L4-pi*rin^2*L4)*rho*2;  % Mass of Descent Supports, kg

mtot=m1+m2+m3+m4   % Total Mass of Ladder System, kg

v1=pi*rout^2*L1*2; % Volume of Ascent Ladder Supports, m^3
v2=pi*rout^2*L2*2; % Volume of Crew Compartment Supports, m^3
v3=pi*rout^2*L3*6; % Volume of Rungs, m^3
v4=pi*rout^2*L4*2; % Volume of Descent Supports, m^3

vtot=v1+v2+v3+v4   % Total Volume of Ladder System, m^3
```

The inputs of this code are ρ_{comp} (density of composite used, kg/m^3), $r_{out/in}$ (outer or inner radius of the tubes, m), and $L_{1\text{-}4}$ (lengths of the tubes, m). The results of the code can be seen in Table 46-5 below.

Table 46-5 Ladder Mass by Section (kg)

CC Supports	Ascent Supports	Descent Supports	Rungs	Total
5.2	8.4	4.4	9.2	27.2

46.1.1.6 Final Design

By using all of the steps mentioned in the previous sections of the appendix, we were able to come to a final design for the MLV. A visual representation of this system can be seen in Figure 46-8 below. As you can see from looking at our final design, there is very little in common with the initial design that was discussed. The only obvious connections are the design of the legs has not deferred a large amount from the initial concept as well as the use of a cylindrical body to house the crew, although we chose to stand the cylinder up instead of laying it on its side.

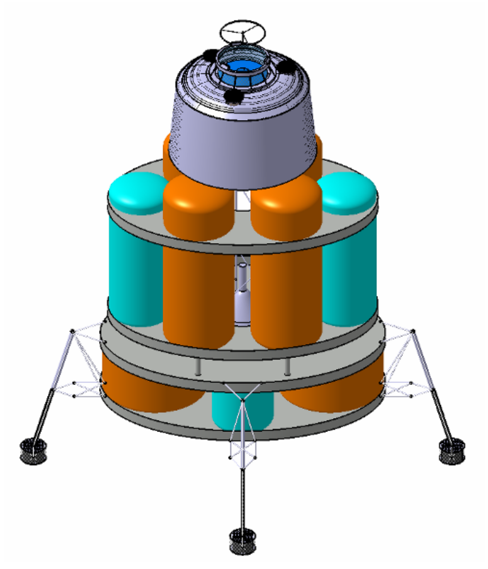


Figure 46-8 Final MLV Design (Chris Cunha)

References

[1] Pullo, Frank A. The Specs. Space/Craft Assembly and Test. February 3, 2005. <http://users.specdata.com/home/pullo/TWOCOL.HTM>.

[2] Duncan, John. The Lunar Module Descent Stage. March 1, 1998. Apollo/Saturn. February 20, 2005. <http://apollosaturn.com/Lmnr/descent.htm>.

[3] Fischer, Mark. Aerobraking and Impact Attenuation. Spring 1995. University of Texas. February 25, 2005. <http://www.tsgc.utexas.edu/archive/subsystems/aero.pdf>.

47 Zeszotek, Michelle

Author: Michelle Zeszotek

47.1.1 Ascent and Recovery Vehicle (ARV) – Phoenix

47.1.1.1 Conceptual Design

The Ascent and Recovery Vehicle (ARV) takes the crew to dock in a long period parking orbit with the Crew Transport Vehicle (CTV). The ARV also returns the crew through the Earth's atmosphere at the end of the mission. The vehicle remains docked with the CTV during the journey to Mars, and remains shut down during interplanetary transit.

Overall vehicle requirements state that the vehicle must be capable of supporting a four person crew for up to 30 hours. We launch the ARV on either the Earth Launch Vehicle (ELV) or a separate launch vehicle that is currently in use and has a sufficient payload capacity. Furthermore, the ARV must have the capability to maneuver, rendezvous, and dock with the CTV beyond Low Earth Orbit (LEO) with an expendable ascent propulsion stage. Thus, the vehicle cannot detach from the CTV for any reason except Earth re-entry. Moreover, it must function with a self contained propulsion, power, and life support systems for maneuvers during ascent and descent.

Upon receiving the requirements for the ARV, decisions were made in accordance to what type of vehicle or spacecraft we would need to design in order to accomplish the specified mission objectives. Many concepts were evaluated and each of these will be covered in the following sections.

47.1.1.1.1 Orbital Space Plane

The first concept evaluated was an Orbital Space Plane (OSP). In essence, the OSP would resemble a vehicle similar to that of the Space Shuttle. The space plane would not have its own main engines, but rather ride atop an expendable rocket, such as a Delta IV or an Atlas V [1]. Pilots would be a thing of the past, and maneuvering the craft in space would be small, automated thrusters. The plane would carry only tiny payloads, making room for them would require reducing the size of the crew and removing seats [1].

Compared to conventional space vehicles, the OSP is capable of not only launching humans into space, but also returning them from space. As many as five astronauts can fly in the space plane.

Furthermore, the OSP has the capability of sustaining the lives on board for up to four days. On some levels, it may appear that the OSP is just a lower-technology way to get people to and from space. The key here is that the OSP would allow for improvements in safety while in transit.

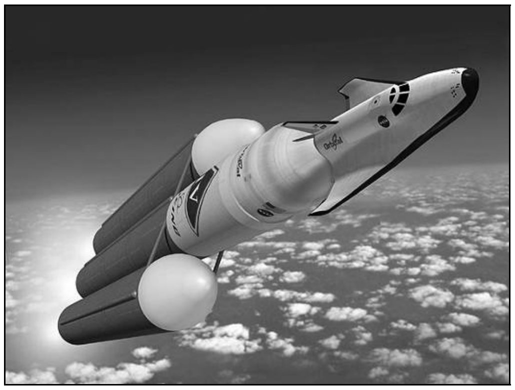


Figure 9: Artist rendition of the an Orbital Space Plane, image from www.spacetoday.org [2]

47.1.1.1.2 Soyuz Spacecraft

The second concept explored was to model the ARV after the existing Soyuz Spacecraft. Being that it is the longest serving manned spacecraft in the world; the Soyuz seems to be a viable option. The Soyuz itself has been improved and redesigned several times over its lifespan and is regarded as a proven and safe spacecraft [3].

The Soyuz launches aboard a Soyuz Rocket. It consists of an Orbital Module, a Descent Module and an Instrumentation/Propulsion Module [4]. Throughout its existence, the Soyuz has increased its safety especially in descent and landing. It has small, more efficient computers and improved displays. In addition, the Soyuz accommodates individuals as large as 1.9 meters. Two engines can reduce landing speeds and forces felt by crew members by 15 to 30 percent [5]. Furthermore, the Soyuz can spend up to one year in space.

47.1.1.1.3 Apollo Command Service Module (CSM)

The final concept studied to model the ARV after, was the Apollo Crew Service Module. Although the existing Apollo hardware would be unsuitable because of obsolescence, lack of traceability, and the inability to qualify these components for flight, an Apollo derived ARV, has sufficient merit. Scaling up the size of the Command Module (CM) by five to eight percent, could make the pre-existing capsule large enough to accommodate six or seven people, well beyond that of the required four crew members for Project Legend [6]. However, little beyond the general shape of the capsule would be retained.

Further studies of the Apollo allow us to recognize the similarities between the functions of the Apollo CM and the functions that need to be performed by the ARV. However, it is also clear that the Service Module on the Apollo is an unnecessary structure for the outlined mission being undertaken. The Apollo in essence, performs every function that the ARV needs to perform.

47.1.1.2 Design Selection

Upon study and exploration of the capabilities inherent in each of the three conceptual designs, we choose to proceed with designing an Apollo-derivative as the ARV.

The Orbital Space Plane was ruled out almost immediately. Due to the fact the Orbital Space Plane is not a proven technology and its technology readiness is unknown for the future, we decided that this was not a viable option. Thus, the safety of the astronauts is priority number one and unproven technology cannot ensure astronaut safety.

The idea of using an existing Soyuz to act as the ARV was contemplated by the group for some time, but ultimately rejected. While the Soyuz accomplishes every objective set forth in the mission specifications, it still requires resizing to accommodate another crew member. Furthermore, we decided that Soyuz is not the best solution, primarily due to the fact that it is a foreign technology. If we are going to land on Mars, it’s going to be an All-American effort.

The preceding explanations lead to the choice of using an Apollo-derived ARV. First and foremost, it is a proven technology. There is also great ease in modifying the Apollo capsule in order to accommodate the mission specifications set forth in this report for the ARV. Also, the existing Apollo capsules are of minimal size.

47.1.1.3 Design Process

Since using an Apollo-derivative was agreed upon, we now move forward with the actual design of the ARV. Upon first designing the ARV, it is most reasonable to scale up the existing Apollo capsule by five to eight percent, in accordance to Reference [6]. This naturally accommodates the one additional crew member for the missions of Project Legend. However, upon calculating the mass for the scaled structure, the ARV is about 32,500 kg. Furthermore, it has become more and more apparent that with a reduction in mass, the Service Module existent on the Apollo CSM would not be needed in the design of the ARV. As a result of the conclusions made upon first inspection of the ARV design, it is evident that the ARV needs to be a design built from the ground up, but based on the Apollo capsule.

Basing the design of the ARV on the Apollo Command Module leads to the basic shape of the ARV, a cone. From there, we determine that the ARV allows just enough room for the astronauts to remain seated, as the mission only requires them to be in the ARV for 30 hours, as opposed to the Apollo capsule mission time of 14 days. Furthermore, in order to ensure that the ARV is structurally sound, we feel that the ARV would best be designed if the crew cabin were modeled as a cylinder. Thus, modeling the crew cabin as a cylinder allows for ease of calculation, and a minimal amount of space is designed for. Since the cylinder would be pressurized in order to sustain the crew, the remaining conical structure is housing for all electronics and systems unnecessary for sustaining human life. As a result of this configuration, the complete structure sees a reduction in weight, being that only a portion of the ARV would need to be pressurized.

Once the basic configuration was decided upon, a more in depth design phase takes place. Sizing of the ARV is done through the use of a Matlab code. This Matlab code can be found at the end of this Appendix, in Section 47.1.1.6. The ARV is constructed out of a composite material. The composite is a carbon, with the fiber made up of plies layered in 0°, ±45°, and 90° orientations. When dividing the properties of the composite material by four, it becomes isotropic and a quality estimate of its weight, sizes, etc. can be obtained [7].

The following Table outlines all structural masses of the ARV.

Table 6: Key structural components of the Ascent and Recovery Vehicle

Group Classification	Item	Mass (kg)
Aerodynamics	*Ballute*	1819
	Parachutes	116.36
	Heat Shield	893
Communications	*Omni Antenna*	26
Dynamics & Controls	*Guidance System*	132.32
Human Factors	*Docking Port*	290
Propulsion	*RCS Thrusters*	742.22
Structures	*Pressurized Volume*	1315
	Helium Tank	4.4
	Structural Supports	500

The geometry of the ARV is seen in the following drawing.

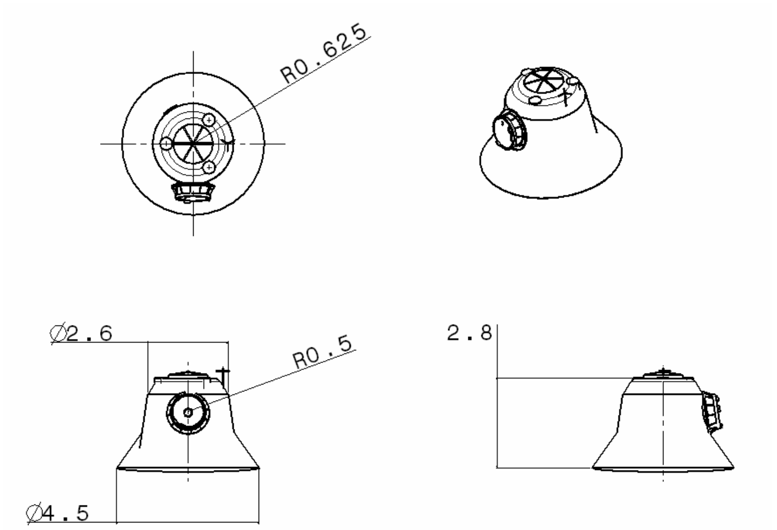


Figure 10: Final Ascent and Recovery Vehicle dimensions, drawing by Chris Cunha

47.1.1.4 Design Features

There are many features of the ARV that must be accounted for, proving that it can accomplish the specified mission. Refer to the ARV Overview for a discussion of these features.

47.1.1.5 Final Design

The final design of the ARV is reviewed in Section 1.5.2 of the document.

47.1.1.6 Matlab Code

The following Matlab code is used to calculate the mass and thicknesses of several structural components present on the ARV. By inputting the requested material properties and dimensions, masses and dimensions of ARV structural components are outputted. The following are the inputs requested by the code: yield stress, elastic modulus, pressure, density, dimensions specific to the ARV, and G's felt at launch. The outputs from the code can be seen in the following section. Equations used in this code can be found in References [7], [8], and [9].

```
clc
clear all

%Assumptions
%Crew compartment is pressurized cylinder
%Cylinder capped with hemi-spheres

yield_stress = 1.56e9/4;                %pa
E = 140e9/4;                            %pa
pressure = 101000;                      %pa
radius = 1.25;                          %m
length = 1.7;                           %m
safety_factor = 2;
shell_safety_factor = 3
rho = 1550                              %kg/m3
top_mass = 2000;                        %total mass supported by top of cylinder
launch_g = 10;
%Cylinder
thick_hoop = shell_safety_factor*pressure*radius/yield_stress   %m
%Cap
thick_cap = shell_safety_factor*pressure*radius/yield_stress    %m

%Mass
shell_mass = (2*pi*radius*thick_hoop*length+2*pi*radius^2*thick_cap*1.25)*rho*2              %kg
      %double walled
shell_area = 2*pi*radius*length + 2*pi*radius^2*1.25;   %m2
shielding_density = shell_mass/shell_area*.1           %g/cm2
%Stiffeners
%Resist axial compression during launch
%Assume worst case load throughout length of cylinder
long_num = 25;                                 %number of axial stiffeners
ring_num = 5;                                  %number of ring stiffeners
stiff_width = .02;                             %radial dimension of axial stiffeners
stiff_length = length/(ring_num+1);     %effective buckling length of axial stiffeners
end_width = .005;                              %length of end of the C channel
spacing = 2*pi*radius/long_num;         %m, distance between stiffeners
 %Failure due to column buckling, pinned ends
stiff_thick_1 = safety_factor*stiff_load*12*stiff_length^2/(pi^2*E*(stiff_width)^3)   %m
%Failure due to yielding
stiff_thick_2 = safety_factor*stiff_load/((stiff_width+2*end_width)*yield_stress)     %m
%Find bigger of thicknesses
stiff_thick = max([stiff_thick_1 stiff_thick_2])                                     %m
stiff_mass = stiff_thick*(2*end_width+stiff_width)*length*rho*long_num        %kg
ring_mass = 2*radius*pi*stiff_width*stiff_thick*rho*ring_num                  %kg
%Floor
floor_load = 60000;                     %N
thick_floor = .05;
sheet_moment = floor_load*radius
thick_sheet = safety_factor*sheet_moment/(thick_floor*yield_stress)
mass_floor = pi*radius^2*thick_sheet*4*rho
struc_mass = (ring_mass + stiff_mass + shell_mass + mass_floor)*4
```

47.1.1.7 Matlab Results

Table 7: Results outputted from Matlab code

Variable	Output
Structural Shell Safety Factor (shell_safety_factor)	3
Material Density (rho)	1550 kg/m^3
Hoop Thickness (thick_hoop)	9.7115e-004 m
Cap Thickness (thick_cap)	9.7115e-004 m
Structural Shell Mass (shell_mass)	77.1419 kg
Shielding Density (shielding_density)	0.3011 g/cm^2
Failure Due to Buckling – Resulting Stiffener Thickness (stiff_thick_1)	0.0055 m
Failure Due to Yielding – Resulting Stiffener Thickness (stiff_thick_2)	0.0013 m
Floor Thickness (thick_sheet)	0.0077 m
Floor Mass (mass_floor)	234.1091 m
Structural Mass (struc_mass)	1.3148e+003 kg

47.1.1.8 References

[1] Foust, Jeff. “Orbital Space Plane: Back to Apollo?" Online. 5 May 2003. Available http://www.thespacereview.com/article/19/1.

[2] “Orbital Space Plane." Online. 17 January 2005. Available http://www.spacetoday.org/SpcShtls/SpacePlane.html.

[3] “Soyuz TMA." Online. 17 January 2005. Available http://offearthnet.rdays.com/manned/main.html.

[4] “Soyuz Spacecraft." Online. 17 January 2005. Available http://www.russianspaceweb.com/soyuz.html.

[5] Darling, David. “The Complete Book of Spaceflight.” John Wiley & Sons, Inc., New Jersey, 2003.

[6] Brown, Irene. “NASA Looks to Apollo for Future Ship." Online. 9 May 2003. Available http://dsc.discovery.com/news/briefs/20030505/apollo.html.

[7] Sun, C.T., “Mechanics of Aircraft Structures.” John Wiley and Sons, Inc., New York, 1998.

[8] Gere, James M., “Mechanics of Materials.” Brooks/Cole, Inc., California, 2001.

[9] Grandt, A.F., AAE 454 Class Notes, Fall 2004.