EMTG Tutorial: Config Files

Tim Sullivan [*]

August 8, 2023

Revision Date	**Author**	**Description of Change**
December 2, 2022	Tim Sullivan	Initial revision.
June 30, 2023	Joseph Hauerstein	Conversion to LaTeX.
August 4, 2023	Joseph Hauerstein	Addition of Known Issues section.

[*]Aerospace Engineer, The Aerospace Corporation

Contents

List of Known Issues

List of Acronyms

EMTG Evolutionary Mission Trajectory Generator

GUI Graphical User Interface

AU Astronomical Unit

NASA National Aeronautics and Space Administration

NLP Nonlinear Program

GMAT General Mission Analysis Toolkit

1 Introduction

Evolutionary Mission Trajectory Generator (EMTG) can model performance characteristics of spacecraft and launch vehicles using text files separate from the main EMTG options file. These text files can be configuration controlled and shared across missions. So far, you've only briefly mentioned the configuration of the launch vehicle and spacecraft. This tutorial will dive deeper into the methods for modeling vehicle performance in EMTG though explaining the different options in these configuration files.

There are three basic ways to set up spacecraft hardware in EMTG, which can be seen on the "Spacecraft Options" tab of the PyEMTG GUI under the "Spacecraft model type" label: "0: Assemble from libraries", "1: Read .emtg_spacecraftoptions file", and "2: Assemble from missionoptions object". In the (Low/O)SIRIS-REx tutorials you used the "2: Assemble from missionoptions object" method by setting spacecraft values in PyEMTG. Method "1: Read .emtg_spacecraftoptions file" uses a single file to set all spacecraft mass, power, and propulsion options. Method "0: Assemble from libraries" uses separate propulsion and power system library files. The system to use is then selected in the "Spacecraft Options" tab. This tutorial will focus on using Method 1, the Spacecraft Options file.

Launch vehicles must be configured using the Launch Vehicle Options file.

More information can be found in the General Mission Analysis Toolkit (GMAT) Optimal Control Specification, Chapter 6: EMTG Spacecraft File Specification and Section 4.8.5: Format of EMTG Spacecraft File, Chapter 5 of the EMTG Software Design Document, and the example configuration files provided with this tutorial.

2 Spacecraft Options

In the LowSIRIS-REx tutorial, you specified the thrust and power systems of the spacecraft directly in PyEMTG. This section will go over how to do this using the spacecraft options file.

The spacecraft options file layout consists of sections of data called "blocks". The first block contains some global data and is identified by the line `#BeginSpacecraftBlock`. The spacecraft block is followed by one or more stage blocks, each of which is identified by the line `#BeginStageBlock`. (In EMTG, "staging" means to change the spacecraft hardware characteristics. Staging events are controlled by check boxes in the upper right of the "Journey Options" tab in PyEMTG. Creating a staging event moves EMTG from the current stage block to the next stage block.) Each stage block is independent of data in other stage blocks and contains some unblocked data, power library data identified by the line `#BeginStagePowerLibraryBlock`, and propulsion library data identified by the line `#BeginStagePropulsionLibraryBlock`. Thus, a two-stage spacecraft options file would be setup as follows:

- Spacecraft block data
- Stage 1 block
  - Unblocked data for stage 1
  - Power library blcok data for stage 1
  - Propulsion library block data for stage 1
- Stage 2 block
  - Unblocked data for stage 2
  - Power library block data for stage 2
  - Propulsion library block data for stage 2

2.1 Spacecraft Block

The spacecraft block data section sets the name of the spacecraft and enables or disables maximum values on the spacecraft electrical and chemical tanks, as well as the spacecraft dry mass. The spacecraft block also provides the required minimum/maximum values for constraints that are enabled. The lines and expected data types are shown in Table 1.

```
#BeginSpacecraftBlock
name example_spacecraft
EnableGlobalElectricPropellantTankConstraint 1
EnableGlobalChemicalPropellantTankConstraint 0
GlobalElectricPropellantTankCapacity 1000.0
GlobalFuelTankCapacity 100.0
GlobalOxidizerTankCapacity 0
```

Figure 1: Example Spacecraft Block Section.

Line number	Line name	Data type
1	Name	String
2	EnableGlobalElectricPropellant-TankConstraint	Boolean integer
3	EnableGlobalChemicalPropella-ntTankConstraint	Boolean integer
4	EnableGlobalDryMassConstrai-nt	Boolean integer
5	GlobalElectricPropellantTankC-apacity	Real (kg)
6	GlobalFuelTankCapacity	Real (kg)
7	GlobalOxidizerTankCapacity	Real (kg)
8	GlobalDryMassLowerBound	Real (kg $\geq 0 \leq$ GlobalDryMassBounds[1])
9	GlobalDryMassUpperBound	Real (kg $\geq$ GlovalDryMassBounds[0])

Table 1: Spacecraft Block Data.

The bounds values are only considered if the corresponding constraint is active. For example, the spacecraft block in Figure 1 would only constrain the electric propellant tank since the constraint is set to “1”. Even though the chemical fuel tank capacity has a value of 100 kg, that constraint would not be active because the corresponding option to enable the chemical tank constraint is set to “0”.

2.2 Stage Unblocked Data

After the Spacecraft Block section, there are one or more “Stage Block” sections. Each Stage Block begins with “Unblocked” information. This section is similar to the Spacecraft Block section, specifying stage masses, constraints, and maximum masses for the fuel tanks. There are also lines for number of thrusters and throttle sharpness. The line items and expected data types are shown in Table 2.

The Stage Block section has a dry mass constraint Boolean option but no entry for the bounds like in the Spacecraft Block section. If `EnableDryMassConstraint == 1`, a constraint is created specifying that the final mass at the end of the stage must be greater than the required mass at the end of the stage. The required mass is the sum of the stage BaseDryMass, stage electric propulsion system mass, stage chemical propulsion system mass, stage power system mass, electric propellent

Line number	Line name	Data type
1	Name	String
2	BaseDryMass	Real (kg)
3	AdapterMass	Real (kg)
4	EnableElectricPropellantTankConstraint	Boolean integer
5	EnableChemicalPropellantTankConstraint	Boolean integer
6	EnableDryMassConstraint	Boolean integer
7	ElectricPropellantTankCapacity	Real (kg)
8	ChemicalFuelTankCapacity	Real (kg)
9	ChemicalOxidizerTankCapacity	Real (kg)
10	ThrottleLogic	Integer(1: min number of thrusters, 2: max number of thrusters
11	ThrottleSharpness	Real (kg)
12	PowerSystem	String (name of power system)
13	ElectricPropulsionSystem	String (name of propulsion system)
14	ChemicalPropulsionSystem	String (name of propulsion system)

Table 2: Stage Unblocked Data.

mass margin, chemical fuel mass margin, chemical oxidizer mass margin. The fractional margins are set in PyEMTG in the “Spacecraft Options” tab under the “Margins” header.

Throttle sharpness is how quickly the thruster transitions between different settings. A larger value indicates a steeper curve and faster transition. The throttle sharpness model uses an approximation of the Heaviside step function where k is the throttle sharpness and x is a power parameter. A k value of 100 is usually a good rule of thumb. Examples of throttle sharpness curves are shown in Figure 2. See Section 5.6.2.4 of the EMTG Software Design Document for a more detailed explanation of how throttle sharpness is implemented and used in EMTG.

$$H = \frac{1}{1 - e^{-2kx}}$$

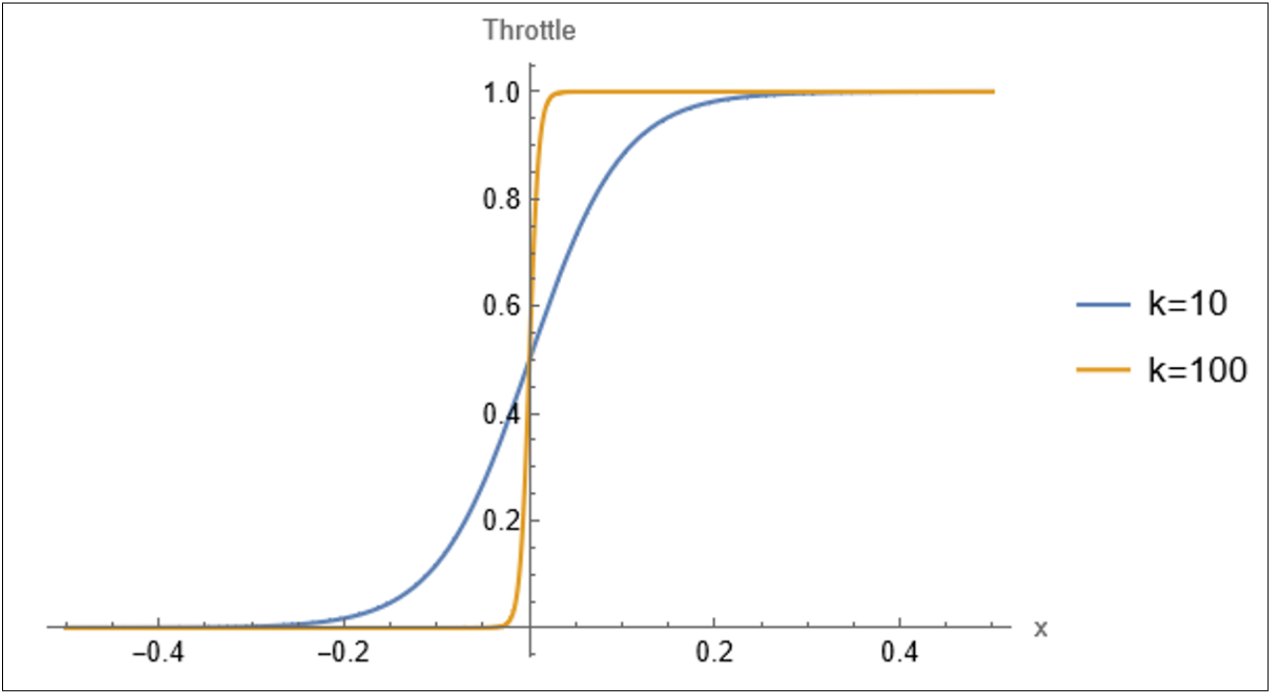


Figure 2: Throttle Sharpness.

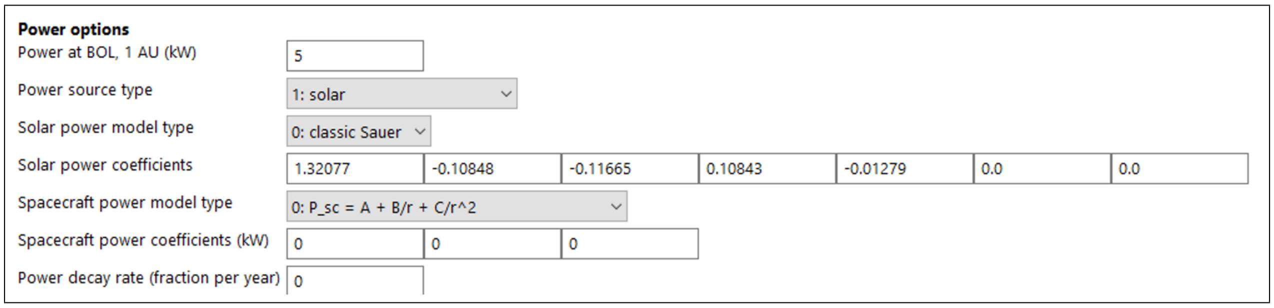
The last lines of the unblocked stage section specify the names of the power, electrical, and chemical propulsion systems being used for this stage. These must correspond to names available in the power and propulsion library blocks for this stage. The library of available power and propulsion systems for the stage are detailed in the next blocks. An example of all stage unblocked data options is shown in Figure 3.

```
#BeginStageBlock
name Example_Stage
BaseDryMass 0
AdapterMass 0
EnableElectricPropellantTankConstraint 0
EnableChemicalPropellantTankConstraint 0
EnableDryMassConstraint 0
ElectricPropellantTankCapacity 1000.0
ChemicalFuelTankCapacity 100.0
ChemicalOxidizerTankCapacity 0
ThrottleLogic 1
ThrottleSharpness 10000
PowerSystem Big_Solar_Arrays
ElectricPropulsionSystem HERMeSx3
ChemicalPropulsionSystem RCS
```

Figure 3: Example Stage Unblocked Data.

2.3 Power Library Block

The Power Library Block specifies in text form the set of power options. This is the information that would be set in the PyEMTG "Power options" section if "Spacecraft model type" were set to "2: Assemble from missionoptions object" as in Figure 4.



Power options

Power at BOL, 1 AU (kW)	5					
Power source type	1: solar					
Solar power model type	0: classic Sauer					
Solar power coefficients	1.32077	-0.10848	-0.11665	0.10843	-0.01279	0.0	0.0
Spacecraft power model type	0: P_sc = A + B/r + C/r^2					
Spacecraft power coefficients (kW)	0	0	0			
Power decay rate (fraction per year)	0					

Figure 4: PyEMTG Power Options.

Power systems are specified by a name followed by a series of comma or space-delimited numbers all on one line. A stage can have multiple power systems specified in the Power Library Block by having multiple lines of data, but only the one specified in the Unblocked Stage data as the `PowerSystem` will be used for that stage. Table 3 specifies the parameters and data types for each Power System.

Index	Variable name	Data type
1	Name	String
2	SpacecraftPowerSupplyType	Integer (0: Constant, 1: Solar)
3	SpacecraftPowerSupplyCurveType	(0: Sauer, 1: Polynomial)
4	SpacecraftBusPowerType	Integer (0: TypeA_Quadratic, 1: TypeB_Conditional)
5	P0	Real (kW)
6	MassPerkW	Real (kg)
7	DecayRate	Real (1/years)
8-14	Gamma[0:6]	Real
15-17	BusPower[0:2]	Real

Table 3: Power System Variables.

2.3.1 Power Supply Type and Curve

Items 2, 3, and 5-14 specify the power produced for the propulsion system, while items 4 and 15-17 specify the spacecraft power use for purposes other than electric propulsion. (I.e., a non-zero bus power means that there is some amount of power produced by the spacecraft that cannot be used by the electric propulsion system.) Several options require coefficients of r_s, which is the spacecraft distance from the Sun, measured in AU. Additional information, including more detailed equations of how various power quantities are calculated, is available is Section 5.5 of the EMTG Software Design Document.

Spacecraft_Power_Supply_Type: This option is an integer specifying either a constant power supply (0) or one dependent on solar power produced by the spacecraft (1).

Spacecraft_Power_Supply_Curve_Type: If the power supply type is set to solar, this option specifies the type of solar power curve, either Sauer or polynomial. The coefficients, γ_i, for both types are specified in indices 8-14 for polynomial curve or 8-11 for Sauer. $P_{Generated}$ is in kW.

0: Sauer

$$P_{Generated} = \frac{P_0}{r_s^2}\left(\frac{\gamma_0 + \frac{\gamma_1}{r_s} + \frac{\gamma_2}{r_s^2}}{1 + \gamma_3 r_s + \gamma_4 r_s^2}\right)$$

1: Polynomial

$$P_{Generated} = P_0 \cdot \sum_{i=0}^{6} \gamma_i r_s^{i-2}$$

P0: This is the power in kW delivered at 1 AU at the start of the mission.

mass_per_kW: This is the spacecraft stage mass increase per kW power delivered.

decay_rate: Decay rate of the power supplied by the power supply curve or constant power source. The units are 1/years and the change in supplied power is given by the following equation where t is in years,

$$P_{Generated} = P_{Generated,nominal} e^{-\text{decay\_rate}\cdot(t - t_{ref})}$$

The epoch t_{ref} is set in PyEMTG on the "Spacecraft Options" tab in the "Power Source Decay Reference Epoch" text box.

gamma[0:6]: Coefficients for the power supply curves, as described above.

2.3.2 Bus Power

The bus power options specify the power used by the spacecraft, which is *not* available to be used by the electric propulsion system.

Spacecraft_Bus_Power_Type: Like power supply type, there are two different options for modeling the spacecraft bus power type. The coefficients for each are specified by the values in indices 15-17.

0: quadratic

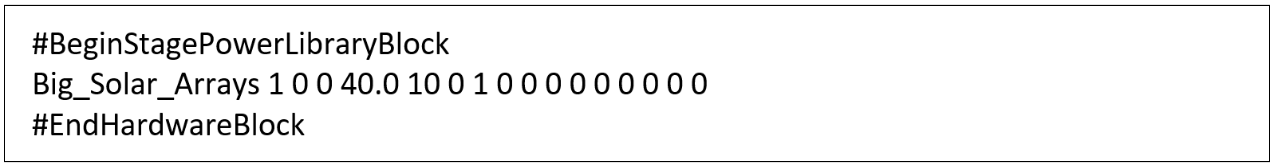
$$P_{Bus} = \sum_{i=0}^{2} \text{bus\_power}_i / r_s^i$$

1: conditional

$$\begin{cases} P_{Bus} = \text{bus\_power}[0], P_{Provided} > \text{bus\_power}[0] \\ P_{Bus} = \text{bus\_power}[0] + \text{bus\_power}[1] \cdot (\text{bus\_power}[2] - P_{Provided}), \text{otherwise} \end{cases}$$

BusPower[0:2]: The coefficients for either bus power type equation.

In practice, early designs frequently make use of simple power models (i.e., lots of zeros in the power system specification line). However, as the overall design increases in fidelity, moving to a more accurately modeled power system helps maintain realism in the trajectory design. An example of a simple power model is shown in Figure 5.



```
#BeginStagePowerLibraryBlock
Big_Solar_Arrays 1 0 0 40.0 10 0 1 0 0 0 0 0 0 0 0 0
#EndHardwareBlock
```

Figure 5: Power Library Block Example for a Simple Power Model.

2.4 Propulsion Library Block

The Propulsion Library Block specifies in text form the set of propulsion system options. These are the options that would be set in the "Propulsion options" section of the PyEMTG GUI when "Spacecraft model type" is set to "2: Assemble from missionoptions object" as shown in Figure 6. There many more types of thrusters than there are power systems, and it's worth selecting different types using the "Engine type" option in PyEMTG to see the options which are valid for each thruster type. Additional information is available is Section 5.6 of the EMTG Software Design Document.

Like the Power Library Block, the Propulsion Library Block contains one or more rows with the name of the propulsion system and a series of parameters that specify its performance. These are specified in Table 4.

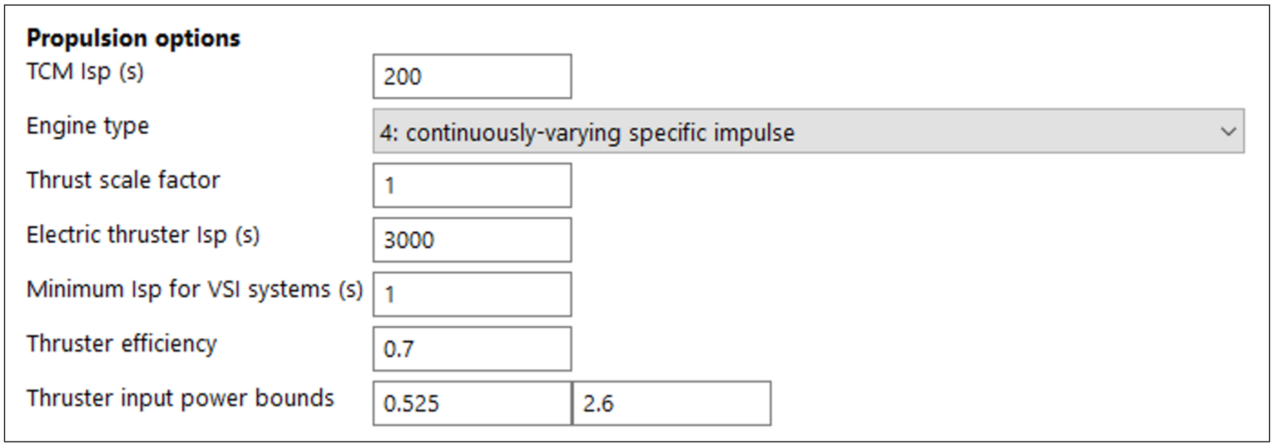

Propulsion options	
TCM Isp (s)	200
Engine type	4: continuously-varying specific impulse
Thrust scale factor	1
Electric thruster Isp (s)	3000
Minimum Isp for VSI systems (s)	1
Thruster efficiency	0.7
Thruster input power bounds	0.525 / 2.6

Figure 6: PyEMTG Propulsion Options for Continuously-varying Isp.

Index	Variable name	Data type
1	Name	String
2	ThrusterMode	Integer (0: ConstantThrustIsp, 1: FixedEfficiencyCSI, 2: FixedEfficiencyVSI, 3: Poly1D, 4: SteppedHThrust1D, 5: SteppedLMdot1D, 6: SteppedHIsp1D, 7: SteppedHefficiency1D, 8: SteppedFullSet1D,, 9: Stepped2D, 10: Poly2D)
3	ThrottleTableFile	String ("none" or path to external file)
4	MassPerString	Real (kg)
5	NumberOfStrings	Integer
6	Pmin	Real (kW)
7	Pmax	Real (kW)
8	ConstantThrust	Real (N)
9	ConstantIsp	Real (s)
10	MinimumIsp/MonopropIsp	Real (s)
11	FixedEfficiency	Real [0,1]
12	MixtureRatio	Real [0,1]
13	ThrustScaleFactor	Real ($\geq$ 0
14-20	ThrustCoefficients[0:6]	Real (mN)
21-27	MassFlowCoefficients[0:6]	Real (mg/s)

Table 4: Propulsion System Variables.

ThrusterMode: Integer specifying type of thruster. Not all variables are relevant for each type of thruster.

ThrottleTableFile: EMTG can read in a file specifying thruster throttle levels and parameters at each level. This capability is beyond the scope of this tutorial. More information is available in the GMAT Optimal Control Specification section 4.8.6 and Section 5.6 of the EMTG Software Design Document.

MassPerString: The mass of the propulsion system for one thruster "string". The total mass of the propulsion system for a given stage is MassPerString * NumberOfStrings.

NumberOfStrings: The number of propulsion system "strings". In other words, the number of thrusters. The logic for the number of strings/thrusters to use is set using PyEMTG in the "Spacecraft Options" tab using the "Throttle logic mode" setting. The number of strings/thrusters that can be used depends on the amount of power available. For example, two strings/thrusters cannot be operated if there is only enough power for one string/thruster. EMTG uses Heaviside step function approximations to continuously transition in an almost-discrete way from n strings to n+1 or n-1 strings.

Pmin: Minimum power necessary for operation of one string/thruster.

Pmax: Maximum power useable by one string/thruster if supplied by the power system.

ConstantThrust: Thrust value for "ThrusterMode 0: ConstantThrustIsp"

ConstantIsp: Isp for constant-Isp thruster modes.

FixedEfficiency: Fixed efficiency if operating in a fixed-efficiency mode.

ThrustScaleFactor: Constant factor by which to scale thrust. Note that this is not the same as the averaged duty cycle that may be set in PyEMTG. The averaged duty cycle represents a multiplier on both the thrust and the mass flow rate, while the ThrustScaleFactor is only applied to the thrust. One application of ThrustScaleFactor is to take into account, to first order, cosine losses caused by a canted thruster.

ThrustCoefficients: Thrust coefficients of P, the input power to the thruster. Assumes P is in kW and Thrust is in mN. Only used if ThrottleTableFile is "none".

$$\text{Thrust} = \sum_{i=0}^{6} \text{thrust\_coefficient}_i \cdot P^i$$

MassFlowCoefficients: Mass flow coefficients of P, the input power to the thruster. Assumes P is in kW and Mass Flow Rate is in mg/s. Only used if ThrottleTableFile is "none".

$$\text{Massflow} = \sum_{i=0}^{6} \text{massflow\_coefficient}_i \cdot P^i$$

3 Launch Vehicle Options

EMTG can also utilize a Launch Vehicle Options file (extension: `.emtg_launchvehicleopt`) to specify the capabilities of one or more launch vehicles. Launch Vehicle Options files are significantly simpler than the spacecraft files, specifying the name, DLA and C3 bounds, and polynomial coefficients for injected mass as a function of C3. (C3 is the energy needed to acheive a particular orbit.) These options are specified in Table 5

Index	Variable name	Data type
1	Name	String
2	ModelType	Integer (0: Polynomial, this is the only type currently available)
3	DLA_Lowerbound	Real (deg)
4	DLA_Upperbound	Real (deg)
5	C3Lowerbound	Real (km^2/s^2)
6	C3Upperbound	Real (km^2/s^2)
7-end of line	C3Coefficient	Real (C3 is in (km^2/s^2)

Table 5: Launch Vehicle Variables.

The equation below describes how the C3 coefficients are used to model the capability of the launch vehicle as a function of C3 in km^2/s^2 within the bounds set by values 5 and 6 in launch vehicle line.

$$\text{Injected\_mass} = \sum_{i=0}^{n} \text{C3\_Coefficient}_i \cdot C3^i$$

The bounds on C3 and DLA set in the launch vehicle file should be the range over which the coefficients are valid. Further constraints (e.g., max C3, min/max DLA) can be set on a mission-by-mission basis using PyEMTG.

Usually, C3 coefficients are set for polynomials up to fifth order (n = 5). Data for creating a polynomial fit is often obtained from NASA's Launch Services Program Launch Vehicle Performance Website: `https://elvperf.ksc.nasa.gov/Pages/Default.aspx`.

4 LowSIRIS-REx High Thrust

Now let's use the spacecraft options file to see how the LowSIRIS-REx solution changes with increased thrust. Make a new directory for this tutorial called `Config_files` and copy and paste in the `LowSIRIS-REx.emtgopt` file and `hardware_models` folder from the LowSIRIS-REx tutorial. Open the `LowSIRIS-REx.emtgopt` file in PyEMTG and change the paths for "Hardware library path" in the "Spacecraft Options" tab, and "Working directory" in the "Output Options" tab to a results directory in the new `Config_files` directory. Don't forget the trailing slash in "Hardware library path". In the "Global Mission Options" tab, change the "Mission Name" to "LowSIRIS-REx_high_thrust".

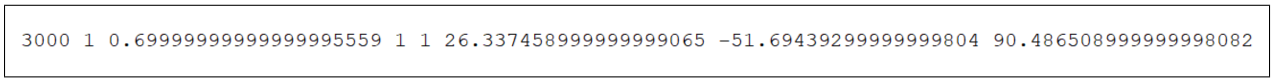
Recall that during the original LowSIRIS-REx tutorial, you selected "2: Assemble from missionoptions object" for "Spacecraft model type" in the "Spacecraft Options" tab. When you ran EMTG, EMTG created a spacecraft options file and placed it alongside the solution file. Open that file, which should be called `LowSIRIS-REx.emtg_spacecraftopt`. Notice that the spacecraft power and propulsion options you set in PyEMTG are now filled out in this spacecraft options file. Make a copy of this file and place it in the `Config_files/hardware_models` directory, renaming it `LowSIRIS-REx_high_thrust.emtg_spacecraftopt`.

Return to PyEMTG and change the following options:

- Solver Options
  - **Inner-loop Solver Mode:** "Nonlinear Program (NLP) with initial guess"
  - **Trial decision vector or initial guess:** "On"
- Spacecraft Options
  - **Spacecraft model type:** "1: Read .emtg_spacecraftoptions file"
  - **Spacecraft file:** "`LowSIRIS-REx_high_thrust.emtg_spacecraftopt`"

Save the EMTG options file as `LowSIRIS-REx_high_thrust.emtgopt`.

Now that you have configured PyEMTG to use a spacecraft options file and provided an initial guess, let's modify the spacecraft propulsion system in the spacecraft options file. Open the `LowSIRIS-REx_high_thrust.emtg_spacecraftopt` file in a text editor and notice there is a `ElectricPropulsionSystemFromMissionOptions` line in the Propulsion Library Block. This line has the propulsion system options you set in the LowSIRIS-REx tutorial. Let's give the spacecraft more thrust. Find the column for the zeroth-order coefficient to the propulsion system power-to-thrust polynomial. It should be variable number 14 and have the value "26.337", partly shown in Figure 7. Change this to "36.337", which will increase the thrust available for all power settings.



```
3000 1 0.69999999999999995559 1 1 26.337458999999999065 -51.69439299999999804 90.486508999999998082
```

Figure 7: Original Electric Propulsion Settings.

Note: Currently, the spacecraft options file EMTG generated for the LowSIRIS-REx mission names the electric propulsion system in the Stage Unblocked section `ElectricPropulsionFromMissionOptions`. However, the name in the Stage Propulsion Library Block is `ElectricPropulsionSystemFromMissionOptions`. Remove the word "System" from the Stage Propulsion Library Block to avoid EMTG throwing an error at runtime because these two names need to be consistent.

After saving the new spacecraft options file, run EMTG (File ->run, Ctrl+r) and open the solution file and the LowSIRIS-REx solution file in a text editor after the NLP solve finishes. Scroll down to the section just above Objective function (line ~143). Notice that the spacecraft dry mass increased, and the electric propellant used has decreased in the higher-thrust mission compared to the lower-thrust mission! This change will affect other portions of the solution, as well.

5 LowSIRIS-REx Low C3

Let's re-run LowSIRIS-REx with a new launch vehicle. When this mission was run to create the earlier tutorial, the launch C3 from Earth was about 10.4km^2/s^2 for LowSIRIS-REx and

7.85km^2/s^2 for the chemical-thrust version. (You may have different values.) Let's modify the launch vehicle configuration. Open the `default.emtg_launchvehicleopt` file in your `Config_files/hardware_models` directory. Copy the `ExampleRocket` line and paste it just below `ExampleRocket` and above `#EndHardwareBlock`. Name the launch vehicle `SmallExampleRocket`. Change the `C3_upperbound` variable (variable 6) from 50 to 6.5 (or something lower than the C3 of your converged solutions from the earlier tutorials). Save the file.

In PyEMTG, switch to the "Global Options" tab and rename the mission "LowSIRIS-REx_high_thrust_low_c3". Switch to the "Spacecraft Options" tab and select the "SmallExampleRocket" as the "Launch vehicle" option (you may need to re-select the "Launch vehicle library file" to see the new launch vehicle).

Run EMTG (File ->run, Ctrl+r) and open the solution file in a text editor after the NLP solve finishes. Look for the first entry in the Journey 0 trajectory (line ~18) and scroll over to the C3 column. You should see 6.5km^2/s^2 as the C3 value (or whatever value you set as the maximum C3 in the launch vehicle configuration file). Scroll down to the spacecraft section. You should see that lowering the C3 also lowered the dry mass and increased the electric propellant required.

This concludes the tutorial on EMTG launch vehicle and spacecraft configuration files. You can now use these files to model real-world thrusters, power systems, and launch vehicles in EMTG.