CSALT and GMAT Optimal Control Documentation

Release R2022a

Noble Hatten, Steve Hughes, Joshua Raymond, Sean Napier

Nov 22, 2022

CONTENTS:

CHAPTER

ONE

OVERVIEW

The Collocation Stand Alone Library and Toolkit (CSALT) is a C++ library that employs collocation to solve the optimal control problem. The library contains approximately 22,000 Source Lines of Code (SLOC) for the CSALT library and about 17,000 SLOC reused from utilities in the General Mission Analysis Tool (GMAT). The system has (loose) dependencies on the Boost C++ library for sparse matrix arithmetic and SNOPT for nonlinear programming. (See *Installation* for more details on how to install CSALT.) The software was developed in collaboration between GSFC engineers and software developers, GSFC support contractors, the Korea Aerospace Research Institute (KARI), and Yonsei University. CSALT is licensed using the Apache License 2.0.

The CSALT library is integrated into GMAT in the GMAT Optimal Control subsystem plugin. This document contains the user guide and the high-level software design documentation for the CSALT and the GMAT Optimal Control plugin. The first few chapters discuss the software contents (*Software Organization and Compilation*), the optimal control problem, and CSALT algorithms (*Concepts and Algorithms*) to a level appropriate to provide context for the software user guide and design sections. The *User Guide* presents the user interfaces for CSALT (which are C++ interfaces) and for the GMAT script interfaces. The documentation includes an overview of components used to solve optimal control problems, a tutorial for CSALT and GMAT applications, and extensive examples (*User Guide*). Reference material illustrates detailed user class/Resource interfaces. The reference material also incudes a complete user interface specification for the GMAT Optimal Control plugin (*User Interface Specification*) and a file specification for Evolutionary Mission Trajectory Generator (EMTG) spacecraft files (*EMTG Spacecraft File Specification*), which are used to specify spacecraft propulsion system properties when using the GMAT Optimal Control plugin.

CHAPTER

TWO

SOFTWARE ORGANIZATION AND COMPILATION

CSALT code and documentation is stored in the main public GMAT repository. Compilation instructions for building CSALT as a stand-alone library, or as a GMAT plugin, are contained in the GMAT build instructions located on gmatcentral.org (search for CMAKE Build System).

Classes and utilities that a user must modify or use to solve optimal control problems are located in the src/csalt/src/userfunutils directory. The doxygen reference material for low-level/executive components that a user need not modify or use are documented separately in the collutils, util, and executive folders, among others.

The CSALT software is organized as shown below:

- docs
  - benchmarking
  - user guide and system description (snapshots of this document)
- src/csalt/src
  - userfunutils
  - collutils
  - utils
  - executive
  - include
- src/csaltTester/src
  - directories for optimal control unit tests
  - TestOptCtrl directory, which holds example problems

An extensive set of optimal control problem examples is included in the subdirectories of src/csaltTester/src/TestOptCtrl. The example problem driver in src/csaltTester/src/TestOptCtrl/src/TestOptCtrl.cpp allows the user to run all or selected example problems via a command-line interface. Test problem source code drivers are located in the TestOptCtrl/src/drivers folder, and path and point function source code is located in the TestOptCtrl/src/pointpath folder.

2.1 Installation

Installation of CSALT and GMAT Optimal Control is complicated by the fact that CSALT relies on several software packages that are not currently bundled with the GMAT installation. As a result, a user must manually place several libraries in specific locations in order to execute CSALT and/or GMAT Optimal Control. The libraries are:

- Sparse Nonlinear OPTimizer (SNOPT) version 7.5 (all platforms)
- Fortran compiler (if compiling SNOPT) and/or Fortran redistributable libraries (if using a pre-compiled SNOPT)

2.1.1 Windows

The SNOPT libraries – snopt7.dll and snopt7_cpp.dll – must be placed in the GMAT bin directory (the same directory in which the GMAT executable is located).

2.1.2 Mac

GMAT provides a configuration file for users to set up MATLAB, PYTHON, SNOPT, and GFORTRAN locations for use with GMAT. This file is named MacConfigure.txt (previously MATLABConfigure.txt) and is located in the bin folder. To set up snopt7 and gfortran – which are needed if and only if you are using the CSALTPlugin:

- Open the MacConfigure.txt in the bin directory and edit the SNOPT_LIB_PATH field to point to the location of your snopt7 dynamic libraries (snopt7.dylib and snopt7_cpp.dylib).
- Edit MacConfigure.txt to point GFORTRAN_LIB_PATH to your gfortran dynamic libraries.

If the CSALT scripts do not run with the GmatConsole command line application, you may need to set up your Terminal so that the system can load the snopt7 and gfortran libraries. For example, if you are using a .bashrc file, you may need to add something like this:

- export SNOPT7 = <path/to/snopt7/libraries/location/>
- export FORTRAN = <path/to/fortran/libraries/location/>
- export DYLD_LIBRARY_PATH=$FORTRAN:$SNOPT7:$DYLD_LIBRARY_PATH

2.1.3 Linux

The SNOPT libraries – snopt7.dll and snopt7_cpp.dll – must be placed in the GMAT lib directory (a directory at the same level as the bin directory containing the GMAT executable).

The SNOPT libraries are built using a Fortran compiler. On Linux, the most likely compiler is the GCC Fortran compiler, gfortran. The compiled SNOPT libraries need access to the corresponding Fortran shared library (e.g. libgfortran.so.4). If that library is not installed on your workstation, place the library in the GMAT lib folder alongside the SNOPT libraries. You may also need to set the load library path (LD_LIBRARY_PATH) if your snopt libraries were not compiled with a run path setting.

CHAPTER

THREE

CONCEPTS AND ALGORITHMS

This chapter presents the optimal control problem and concepts,. It begins with a formal mathematical description of the problem and documents the notation selected for this work (based on Bett's notation). We provide practical discussion of the discretization of the optimal control problem using collocation, the concept of a "Trajectory", and the concept of a "Phase". Trajectory and Phase are the two primary Resources that the user employs to solve problems using collocation in GMAT.

3.1 The Optimal Control Problem

The optimal control problem, expressed in words, is the problem of finding a control and state history that minimizes a cost function (mass, time, money, etc.) subject to a set of dynamics and constraints. In mathematical form, the problem can be written as: Minimize the following cost function (written in what is called the Bolza form):

$$J = \sum_{k=1}^{N} \left(\Phi \left[y^{(k)} \left(t_0^{(k)} \right), t_0^{(k)}, y \left(t_f^{(k)} \right), t_f^{(k)} \right] + \int_{t_0}^{t_f} \lambda \left[y^{(k)}(t), u^{(k)}(t), t^{(k)} \right] dt \right)$$

subject to the dynamics constraints f

$$\dot{y}^{(k)}(t) = f \left[y^{(k)}(t), u^{(k)}(t), t^{(k)} \right]$$

the algebraic path constraints g

$$g_{min}^{(k)} \leq g \left[y^{(k)}(t), u^{(k)}(t), t^{(k)} \right] \leq g_{max}^{(k)}$$

and the boundary conditions ϕ

$$\phi_{min}^{(k)} \leq \phi \left[y^{(k)} \left(t_0 \right), t_0^{(k)}, y^{(k)} \left(t_f \right), t_f^{(k)} \right] \leq \phi_{max}^{(k)}$$

In this work, we will employ notation based on the work of Betts, where the definitions in Table 3.1 apply :

Table 3.1: CSALT terminology.

Term	Symbol	Description
Trajectory	N/A	Container of phases and how they are linked together
Phase	N/A	A segment of a trajectory that is modeled with a single dynamics model. When dynamics models must change, multiple phases are required.
Phase index	k	Phase is a sub-segment of the problem governed by dynamics potentially different from other phases of the problem. Not all problems require multiple phases, but many do. For a Lunar transfer orbit problem, we may have three phases: the escape trajectory near Earth, the trajectory from the lunar Sphere of Influence (SOI) to periapsis, and the trajectory in lunar orbit.
State vector	y	State component whose time evolutions is governed by ordinary differential equations. For orbit problems, this may be x, y, z, vx, yv, vz, mass.
Control vector	u	Control vector. For orbit problems, usually the thrust acceleration or Δv.
Static parameter	s	Static optimization parameters that are NOT governed by differential equations.
Cost function	J	The function to be minimized or maximized.
Algebraic path constraints	g	A constraint that must be satisfied at all quadrature points. These constraints can be expressed as an algebraic function of the optimization parameters and constants.
Boundary constraint	ϕ	A constraint that must be satisfied at a phase boundary. These constraints can be expressed as an algebraic function of the optimization parameters and constants. An example could be that at the end of phase 2, the spacecraft must be at a periapsis (one constraint), and that the radius must be greater than 10,000 km (a second constraint).
Linkage constraint	Ψ	A constraint that determines state continuity (or discontinuity) at the joint of two phases.
Decision vector	$\mathbf{Z}$	A vector containing all optimization parameters for a given optimal control problem discretization. It is organized in subvectors for each phase. $\mathbf{Z} = \begin{bmatrix} z^1 & z^2 \dots z^N \end{bmatrix}$, where N is the number of phases.
Decision vector sub-vector	z^k	The sub-vector of $\mathbf{Z}$ of decision variables that are associated with the "kth" phase. $z^k = \begin{bmatrix} y^k & u^k & s^k & q^k \end{bmatrix}$. Note that y^k is a vector containing states at all mesh/stage points in the phase. So, for example, y^k has length of approximately $(\mathrm{numStates} * (\mathrm{numMeshPoints} + \mathrm{numStages})$. The lengths of each sub-vector in each phase are NOT necessarily the same. The user selects the number of mesh points. The actual mesh point locations are then determined by the transcription.

3.2 CSALT Algorithms

CSALT employs collocation to solve the optimal control problem. Collocation converts an optimal control problem into a large, sparse Non-Linear Programming (NLP) [Nocedal, 2006] problem. The differential equations in the optimal control problem are converted to a set of differential algebraic equations whose solution approximates the solution to the differential equations. The approximation accuracy is governed by the transcription (how the differential equations are expressed as a system of algebraic equations) and how accurately those equations are solved. CSALT employs both low- and high-order transcriptions that are all implicit integration schemes. The transcriptions currently supported are Hermite-Simpson, Lobatto IIIa [Betts,2016] methods of order 4, 6, and 8, and Radau orthogonal collocation of user-specified order [Patterson, 2014]. CSALT casts all transcriptions in the form proposed by Betts (Practical Methods for Optimal Control ..., pg. 146):

$$f = Az + Bq \tag{3.1}$$

where A and B are constant matrices dependent on the transcription, z is the decision vector, f is the vector of NLP functions, and q is the vector of optimal control functions evaluated at the discretization points. The system currently supports Mayer-, Lagrange-, or Bolza-form cost functions, algebraic path constraints, and algebraic point constraints. Optimization parameters include state, control, and time parameters.

3.3 Derivatives and Sparsity Determination

Sparse derivatives are supplied to the NLP solver by differentiating Eq. (3.1), resulting in

$$\frac{\partial f}{\partial z} = A + B\frac{\partial q}{\partial z}$$

The derivatives are computed using sparse matrix representations of the arrays, where the optimal control problem derivatives $\partial q/\partial z$ can optionally be provided by the user. If some or all of the optimal control derivatives are not provided, CSALT performs finite differencing of the optimal control functions.

NLP sparsity is determined using

$$\text{Sparsity}\left(\frac{\partial f}{\partial z}\right) = \text{Sparsity}\left(A + B\frac{\partial q}{\partial z}\right)$$

where the sparsity of the user's optimal control functions are determined by randomly varying the decision variables within the user-defined bounds on those variables. In addition, GMAT Optimal Control implements analytical sparsity pattern determination for the partial derivatives of state variable derivatives with respect to state variables and control variables.

3.4 Mesh Refinement

Currently, CSALT supports mesh refinement for the Radau orthogonal collocation method. The role of the mesh-refinement algorithm is to apply proper changes to the discretization (i.e., the length of the mesh interval and the degree of the approximating polynomial) in order to satisfy the user-defined tolerance on the relative collocation error. The relative collocation error represents the quality of the collocated solution, and the mesh-refinement algorithm estimates the relative collocation error as the difference between the approximating polynomials and the quadrature integration results of the dynamics functions. The required polynomial degree of a mesh interval is obtained as follows:

$$P^k = \log_{N^k}\left(\frac{e}{\epsilon}\right), \quad k = 0, 1, 2, \ldots, k_{max}$$

where N^k is the current polynomial degree after the k-th mesh refinement, P^k is the required polynomial degree change, ϵ is the collocation error tolerance, k_{max} is the maximum number of mesh refinement iterations, and e is the

current estimate of relative collocation error. In addition, there are static tuning parameters N_{min} and N_{max} of the mesh refinement algorithm that define the boundaries of the polynomial degree such that:

$$N_{min} \leq N \leq N_{max}, \quad \text{for any } k$$

CSALT adopts $N_{min} = 3$, and $N_{max} = 14$. If $N^k + P^k \leq N_{max}$, the degree of the polynomial is updated using

$$N^{k+1} = N^k + P^k$$

If $N^k + P^k > N_{max}$, the mesh refinement algorithm divides the mesh interval into subintervals having $N^{k+1} = N_{min}$. The number of subintervals B is given as follows:

$$B = \max\left(\left[\frac{N^k + P^k}{N_{min}}\right], 2\right)$$

3.5 A Note on Optimization

The NLP solver used by CSALT does not enforce that all constraints be satisfied at every evaluation of the user path and point functions until convergence is achieved. This feature allows for the NLP solver to "explore" the solution space. However, this feature may cause problems if there exists the possibility for encountering numerical difficulties in user path/point functions for certain values of decision variables. For example, in GMAT, there are atmospheric models that can only be evaluated in certain altitude ranges (e.g., altitudes greater than 100 km). An exception will be thrown and optimization will cease if a user path/point function attempts to evaluate at an invalid altitude. Other similar cases may exist, and the user must select their problem setup carefully to avoid such issues.

CHAPTER

FOUR

USER GUIDE

This section contains the user guide for CSALT and the GMAT Optimal Control subsystem that is built upon CSALT. To provide context, the high-level architecture of CSALT and the GMAT Optimal Control subsystem is shown in Fig. 4.1. The core of the system is the CSALT API. The GMAT Optimal Control plugin is built upon the CSALT API and is integrated into GMAT using the GMAT plugin interface.

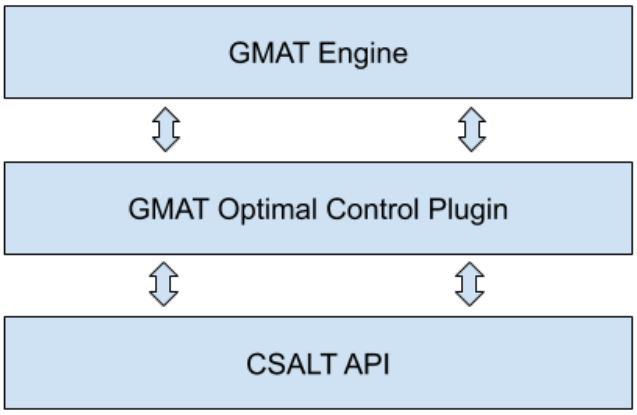

Fig. 4.1: High-level architecture of CSALT and GMAT Optimal Control.

This section contains two tutorials and extensive reference material for CSALT and GMAT Optimal Control components. Tutorials include how to use CSALT as a stand-alone C++ library, and how to use the GMAT Optimal Control subsystem. For each interface, we include a high-level overview of the classes/Resources used to specify optimal control problems and intermediate-level details for key classes. Additional detailed reference material is provided in *User*

Interface Specification and *EMTG Spacecraft File Specification* and via Doxygen output and extensive examples that are included in the CSALT distribution.

4.1 Tutorial: Setting Up an Optimal Control Problem in CSALT

In this section, we describe how to set up an optimal control problem using the Brachistichrone problem as an example. We begin by providing an overview of the problem setup procedures for a general optimal control problem, then present the Brachistichrone problem statement, followed by the C++ source code for the Brachistichrone problem.

4.1.1 CSALT C++ User Interface Component Overview

The CSALT API is composed of several classes shown in Fig. 4.2. The user classes that define the API are Trajectory, Phase, UserPathFunction, UserPointFunction, OptimalControlFunction, and Publisher. Note that UserFunction is an abstract base class that contains commonality between path and point functions.

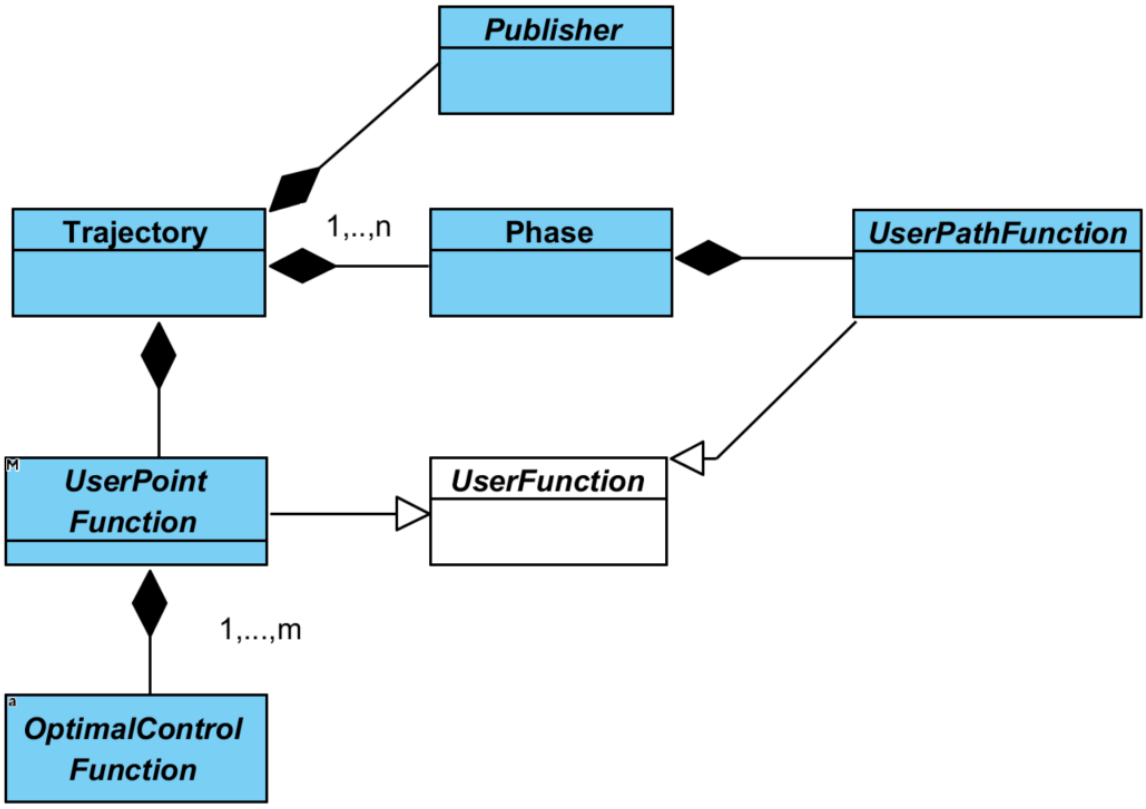

Fig. 4.2: CSALT C++ user classes.

The responsibilities of classes in the CSALT API are highlighted in Table 4.1.

Table 4.1: CSALT API classes.

Class	Required?	Responsibility
Trajectory	Yes	Trajectory is the container for all phases in the problem and for the UserPointFunction and UserPathFunction objects. Internally, the Trajectory concatenates functions and Jacobians provided on different phases and from UserPointFunction.
Phase	Yes	Phase defines the transcription employed and manages/concatenates the defect constraints, integral cost, and algebraic path functions.
UserPathFunction	Yes	Computes path function values (dynamics, integral cost, and algebraic path constraints) and, optionally, analytic Jacobians of path functions. This class provides the generic interface for those computations. The user provides problem-specific functions by deriving a path function class from the UserPathFunction base class.
UserPointFunction	Yes	Computes boundary functions (cost and algebraic point constraints) and, optionally, analytic boundary Jacobians. This class provides the generic interface for those computations. The user provides problem-specific functions by deriving a point function class from the UserPointFunction base class.
OptimalControlFunction	No	A helper class for optimal control functions and Jacobians. This class provides a generic interface for optimal control functions and helps modularize problems that have large numbers of boundary constraints and allows the user to re-use functions across different problems without duplicating code.
Publisher	No	A helper class to publish data before, during, or after optimization.

4.1.2 Configuring Files for Tutorial

Configuring and executing files for the tutorial is done by adding the example problem to the already existing example problems.

Six C++ files must be created: a driver .cpp file, a path function .cpp file, a point function .cpp, and header .hpp files for each .cpp file. The driver class derives from the `CsaltTestDriver` class; the path function class derives from the `UserPathFunction class`; and the point function class derives from the `UserPointFunction` class. Driver .cpp and .hpp files are placed in the gmat/src/csaltTester/src/TestOptCtrl/drivers directory. Path function and point function .cpp and .hpp files are placed in the gmat/src/csaltTester/src/TestOptCtrl/src/pointpath directory. Depending on the method of compilation used, paths to .cpp files may need to be added to the list of `CSALT_TEST_SRCS` in gmat/src/csaltTester/CMakeLists.txt.

Once the driver class, point function class, and path function class are created, the problem is added to the list of test problems in TestOptCtrl.cpp. Within TestOptCtrl.cpp, do the following:

1. Find text "`*driver21 = new HohmannTransferDriver()`". At the end of the list of driver objects, add your own in the same style as the others. Note that this code block occurs in two places in the file, and both locations should be updated.
2. Directly below the previously referenced list of `driver` objects, add a call to the `Run` method of your driver to the end of the list of `Run` methods already in the file. Note that this code block occurs in two places in the file, and both locations should be updated.
3. Delete your driver object. (Add to list of delete calls directly below list of Run calls.) Note that this code block occurs in two places in the file, and both locations should be updated.
4. Find text "`else if (test == "HohmannTransfer")`". At the end of the list of `else if` blocks, add a new `else if` block for your own CSALT driver. Note that this code block occurs in two places in the file, and both locations should be updated.

5. Find text "`msg += " HohmannTransfer"`". At the end of the list of additions to the `msg` variable – but prior to the new line before the `"exit"` line – add to `msg` the name of your example, modeled after the problems already listed in the file. Note that this code block occurs in two places in the file, and both locations should be updated.
6. Find text "`*driver21 = new HohmannTransferDriver()`". Add a line creating your driver with an unused pointer name at the end of the block of pointer declarations. Scroll down to the next blocks of code, and add a `Run()` command and `Delete` command for the pointer to your driver class.
7. Find text "`test = "HohmannTransferDriver"`". Add a case block for your driver to the bottom of the already existing list of case blocks.

4.1.3 Overview of CSALT Configuration Process

The key steps in setting up an optimization problem in CSALT are:

1. Configure the Path Functions
2. Configure the Boundary Functions
3. Configure the Phases and Transcription
4. Configure the Trajectory
5. Solve the problem and examine results

To solve the Brachistichrone problem using CSALT, let's begin with the Brachistichrone problem statement:

Minimize the final time

$$\text{Minimize } J = t_f$$

Subject to the dynamics

$$\begin{aligned}\dot{x} &= v \sin u \\ \dot{y} &= v \cos u \\ \dot{v} &= g_0 \cos u\end{aligned}$$

the initial boundary conditions

$$\begin{aligned}t_0 &= 0 \\ x\left(t_0\right) &= 0 \\ y\left(t_0\right) &= 0 \\ v\left(t_0\right) &= 0\end{aligned}$$

and the final boundary conditions

$$\begin{aligned}0 \leq t_f &\leq 10 \\ x\left(t_f\right) &= 1 \\ -10 \leq y\left(t_f\right) &\leq 10 \\ -10 \leq v\left(t_f\right) &\leq 0\end{aligned}$$

4.1.4 Step 1: Configure the Path Functions

Path functions, including dynamics, algebraic path constraints, and, if applicable, an integral cost function, are implemented by deriving a class from the CSALT UserPathFunction base class and implementing the EvaluateFunctions() and EvaluateJacobians() methods. The EvaluateFunctions() method is required. The EvaluateJacobians() method is required but may be empty. If EvaluateJacobians() does not set a particular Jacobian, CSALT will finite-difference that Jacobian. For the Brachistichrone example, we have provided all path function Jacobians analytically in the source code.

The dynamics model for the Brachistichrone problem is:

$$\dot{x} = v \sin u$$
$$\dot{y} = v \cos u$$
$$\dot{v} = g_0 \cos u$$

The EvaluateFunctions() method that implements those dynamics is shown below. For this example, the Path Function class is named BrachistichronePathObject.

```
// Class constructor
BrachistichronePathObject::BrachistichronePathObject() :
UserPathFunction(),
gravity(-32.174)
{
}

// The EvaluateFunctions method
void BrachistichronePathObject::EvaluateFunctions()
{
    // Extract parameter data
    Rvector stateVec = GetStateVector();
    Rvector controlVec = GetControlVector();

    // Compute the ODEs
    Real u = controlVec(0);
    Real x = stateVec(0);
    Real y = stateVec(1);
    Real v = stateVec(2);
    Real xdot = v * sin(u);
    Real ydot = v * cos(u);
    Real vdot = gravity * cos(u);

    // Set the ODEs by calling SetFunctions with DYNAMICS enum value
    Rvector dynFunctions(3, xdot, ydot, vdot);
    SetFunctions(DYNAMICS, dynFunctions);
}

void BrachistichronePathObject::EvaluateJacobians()
{
    // Get state and control
    Rvector stateVec = GetStateVector();
    Rvector controlVec = GetControlVector();
    Real u = controlVec(0);
    Real x = stateVec(0);
    Real y = stateVec(1);
```

(continues on next page)

(continued from previous page)

```
    Real v = stateVec(2);

    // The dynamics state Jacobian
    Real dxdot_dv = sin(u);
    Real dydot_dv = cos(u);
    Rmatrix dynState(3, 3,
        0.0, 0.0, dxdot_dv,
        0.0, 0.0, dydot_dv,
        0.0, 0.0, 0.0);

    // The dynamics control Jacobian
    Real dxdot_du  = v * cos(u);
    Real dydot_du = -v * sin(u);
    Real dvdot_du = -gravity * sin(u);
    Rmatrix dynControl(3, 1, dxdot_du, dydot_du, dvdot_du);

    // The dynamics time Jacobian
    Rmatrix dynTime(3, 1, 0.0, 0.0, 0.0);

    // Set the Jacobians
    SetJacobian(DYNAMICS, STATE, dynState);
    SetJacobian(DYNAMICS, CONTROL, dynControl);
    SetJacobian(DYNAMICS, TIME, dynTime);
}
```

Note that CSALT uses GMAT variable types like Real, Rvector, and Rmatrix.

4.1.5 Step 2: Configure the Boundary Functions

Boundary functions, including algebraic constraints and cost function, are implemented by deriving a class from the CSALT UserPointFunction base class and implementing the EvaluateFunctions() method. The EvaluateFunctions() method is required.

The boundary functions for the Brachistochrone problem are

$$\text{Minimize } J = t_f$$

subject to the initial boundary conditions

$$\begin{aligned} t_0 &= 0 \\ x\left(t_0\right) &= 0 \\ y\left(t_0\right) &= 0 \\ v\left(t_0\right) &= 0 \end{aligned}$$

and the final boundary conditions

$$\begin{aligned} 0 \leq t_f &\leq 10 \\ x\left(t_f\right) &= 1 \\ -10 \leq y\left(t_f\right) &\leq 10 \\ -10 \leq v\left(t_f\right) &\leq 0 \end{aligned}$$

The EvaluateFunctions() method that implements those boundary functions is shown below. For this example, the Point Function class is named BrachistichronePointObject.

```
//------------------------------------------------------------------------------
// void EvaluateFunctions()
//------------------------------------------------------------------------------
void BrachistichronePointObject::EvaluateFunctions()
{
    // Extract parameter data
    Rvector stateInit = GetInitialStateVector(0);
    Rvector stateFinal = GetFinalStateVector(0);
    Real tInit= GetInitialTime(0);
    Real tFinal = GetFinalTime(0);
    Real xInit = stateInit(0);
    Real yInit = stateInit(1);
    Real vInit = stateInit(2);
    Real xFinal = stateFinal(0);
    Real yFinal = stateFinal(1);
    Real vFinal = stateFinal(2);

    // Dimension the function array and bounds arrays
    Integer numFunctions = 8;
    Rvector funcValues(numFunctions);
    Rvector lowerBounds(numFunctions);
    Rvector upperBounds(numFunctions);

    // Initial Time Constraint: t_0 = 0;
    funcValues(0) = tInit;
    lowerBounds(0) = 0.0;
    upperBounds(0) = 0.0;
    // Constraint on initial x value: x(0) = 0.0;
    funcValues(1) = xInit;
    lowerBounds(1) = 0.0;
    upperBounds(1) = 0.0;
    // Constraint on initial y value : y(0) = 0.0;
    funcValues(2) = yInit;
    lowerBounds(2) = 0.0;
    upperBounds(2) = 0.0;
    // Constraint on initial v value : v(0) = 0.0;
    funcValues(3) = vInit;
    lowerBounds(3) = 0.0;
    upperBounds(3) = 0.0;
    // Final time constraint : 0.0 <= t_F = 10;
    funcValues(4) = tFinal;
    lowerBounds(4) = 0;
    upperBounds(4) = 10.00;
    // Constraint on final x value : x(t_f) = 1.0;
    funcValues(5) = xFinal;
    lowerBounds(5) = 1.0;
    upperBounds(5) = 1.0;
    // Constraint on final y value; -10 <= y(t_f) <= 10
    funcValues(6) = yFinal;
    lowerBounds(6) = -10.0;
    upperBounds(6) = 10.0;
    // Constraint of final v value: -10 <= v(t_f) <= 0.0
    funcValues(7) = vFinal;
```

(continues on next page)

(continued from previous page)

```
    lowerBounds(7) = -10.0;
    upperBounds(7) = 0.0;

    // Set the cost and constraint functions
    Rvector costFunc(1, tFinal);
    SetFunctions(COST, costFunc);
    SetFunctions(ALGEBRAIC, funcValues);
    SetFunctionBounds(ALGEBRAIC, LOWER, lowerBounds);
    SetFunctionBounds(ALGEBRAIC, UPPER, upperBounds);
}
```

Note that interfaces for configuring problems with large numbers of boundary constraints and/or analytic Jacobians are explained in *Boundary Functions Reference*.

4.1.6 Step 3: Configure the Phases and Transcription

The next step in configuring the Brachistochrone problem is to set up a Phase. (This example is a single-phase problem.) The Phase object contains the configuration for the selected transcription, the bounds on optimization parameters (e.g., time, state, and control), and the initial guesses for those parameters. In this example, the phase is configured to use Radau orthogonal collocation with two equally spaced mesh intervals, each containing five points. (The mesh interval configuration and initial-guess options are presented in more detail in *Phase Reference*).

```
// Create a phase and set transcription configuration
RadauPhase *phase1 = new RadauPhase();
std::string initialGuessMode = "LinearNoControl";
Rvector meshIntervalFractions(3,-1.0, 0.0,1.0);
IntegerArray meshIntervalNumPoints;
meshIntervalNumPoints.push_back(5);
meshIntervalNumPoints.push_back(5);

// Set time properties
Real timeLowerBound   = 0.0;
Real timeUpperBound   = 100.0;
Real initialGuessTime = 0.0;
Real finalGuessTime   = .3;

// Set state properties
Integer numStateVars  = 3;
Rvector stateLowerBound(3, -10.0, -10.0, -10.0);
Rvector stateUpperBound(3,  10.0,   0.0,   0.0);
Rvector initialGuessState(3, 0.0,   0.0,   0.0);
Rvector finalGuessState(3,   2.0,  -1.0,  -1.0);

// Set control properties
Integer numControlVars  = 1;
Rvector controlUpperBound(1,  10.0);
Rvector controlLowerBound(1, -10.0);

// Set the phase configuration
phase1->SetInitialGuessMode(initialGuessMode);
phase1->SetInitialGuessArrays(timeArray, stateArray, controlArray);
```

(continues on next page)

(continued from previous page)

```
phase1->SetNumStateVars(numStateVars);
phase1->SetNumControlVars(numControlVars);
phase1->SetMeshIntervalFractions(meshIntervalFractions);
phase1->SetMeshIntervalNumPoints(meshIntervalNumPoints);
phase1->SetStateLowerBound(stateLowerBound);
phase1->SetStateUpperBound(stateUpperBound);
phase1->SetStateInitialGuess(initialGuessState);
phase1->SetStateFinalGuess(finalGuessState);
phase1->SetTimeLowerBound(timeLowerBound);
phase1->SetTimeUpperBound(timeUpperBound);
phase1->SetTimeInitialGuess(initialGuessTime);
phase1->SetTimeFinalGuess(finalGuessTime);
phase1->SetControlLowerBound(controlLowerBound);
phase1->SetControlUpperBound(controlUpperBound);
```

4.1.7 Step 4: Configure the Trajectory

The next step is to add the path function, boundary function, and phase objects to a Trajectory.

```
// Create a trajectory object
// To illustrate the high level approach, default tolerances are used.
traj = new Trajectory();

// Create the path and point function objects and add to Trajectory
pathObject = new BrachistichronePathObject();
pointObject = new BrachistichronePointObject();
traj->SetUserPathFunction(pathObject);
traj->SetUserPointFunction(pointObject);

// Add the Phase to the Trajectory
std::vector<Phase*> phaseList;
phaseList.push_back(phase1);
traj->SetPhaseList(phaseList);
```

4.1.8 Step 5: Run the Optimizer and Examine the Solution

Now that the path, boundary, phase, and trajectory objects are configured, we are ready to optimize the problem and write the solution to a file.

```
// Optimize the problem
traj->Optimize();

// Write the solution to file
std::string solutionFile = "DocTestFile.och";
traj->WriteToFile(solutionFile);
```

The console output for the Brachistichrone problem solved using SNOPT is shown in Fig. 4.3.

In the example above, we called WriteToFile() to write the state and control history to a text file. The contents of the file are shown in Fig. 4.4.

```
The user has defined     299   out of     299   first  derivatives


Major Minors     Step   nCon Feasible  Optimal  MeritFunction     nS Penalty
    0     24              1  1.2E-02  1.2E-01  3.0000000E-01      9          r
    1      1  1.0E+00      2  5.6E-03  2.9E-03  3.0766234E-01      9 3.4E+01   r
    2      1  1.0E+00      3  7.0E-05  1.4E-03  3.1282467E-01      9 9.1E+00 s
    3      1  1.0E+00      4  6.6E-04  1.5E-03  3.1262882E-01      9 9.1E+00
    4      1  1.0E+00      5  4.4E-04  4.9E-04  3.1253234E-01      9 9.1E+00
    5      1  1.0E+00      6  4.7E-05  3.6E-04  3.1249480E-01      9 9.1E+00
    6      1  1.0E+00      7  9.9E-05  1.3E-04  3.1248886E-01      9 9.1E+00
    7      1  1.0E+00      8  3.6E-06  1.7E-04  3.1248715E-01      9 9.1E+00
    8      1  1.0E+00      9  1.4E-05  1.7E-04  3.1248230E-01      9 9.1E+00
    9      1  1.0E+00     10  1.6E-06  1.1E-04  3.1248064E-01      9 9.1E+00

Major Minors     Step   nCon Feasible  Optimal  MeritFunction     nS Penalty
   10      1  1.0E+00     11  3.6E-06 (6.2E-05) 3.1248022E-01      9 9.1E+00
   11      1  1.0E+00     12 (3.2E-07)(2.3E-05) 3.1248018E-01      9 9.1E+00

SNOPTA EXIT   0 -- finished successfully
SNOPTA INFO   1 -- optimality conditions satisfied

Problem name                     CSALT
No. of iterations                   35   Objective           3.1248015730E-01
No. of major iterations             11   Linear    obj. term 0.0000000000E+00
Penalty parameter             9.092E+00  Nonlinear obj. term 3.1248015730E-01
User function calls (total)         12
No. of superbasics                   9   No. of basic nonlinears            34
No. of degenerate steps              0   Percentage                       0.00
Max x                       45 6.4E+00   Max pi                     39 1.0E+00
Max Primal infeas            0 0.0E+00   Max Dual infeas            26 2.3E-05
Nonlinear constraint violn     2.1E-06
```

Fig. 4.3: Brachistichrone problem optimization console output.

```
Optimal Control History file written by CSALT, 13-07-2020 06:08:04

EXIT_CONDITIONS_START
    OPTIMIZER_FEASIBILITY_METRIC    = 3.236575009953831 56e-07
    OPTIMIZER_OPTIMALITY_METRIC     = 2.28796088525624822e-05
    COST                            = 3.12480157301683104e-01
    MAX_CONSTRAINT_VIOLATION        = 2.07159665421841055e-06
    OPTIMIZER_EXIT_STATUS           = Exited with code 1, optimizer converged
    MAX_RELATIVE_MESH_ERROR         = 1.01597150454856537e-05
    SOLUTION_TYPE                   = Converged solution
EXIT_CONDITIONS_END

META_START
    NUM_STATES      = 3
    NUM_CONTROLS    = 1
    NUM_INTEGRALS   = 0
META_STOP

DATA_START
    0                       0                       0                       0                       0
    0.021836092197303213    0.00055787506564573916  -0.0076369297373904436  -0.70114547731859134    -0.10975154388499489
    0.065059863597643361    0.014534613833649646    -0.065705707014355519   -2.0561291979611163     -0.32699883702371246
    0.11298610442608509     0.072880583549247091    -0.18420847211881816    -3.4429453906000518     -0.56793320627145105
    0.14731811455861624     0.15442039138179042     -0.28980381249726744    -4.3183289061078485     -0.74068456477484423
    0.15624007865084155     0.18169227800465793     -0.31830789965396161    -4.5257570428278768     -0.7891630263868874
    0.17807617084814475     0.25930051051530384     -0.38752442937536663    -4.9935977443629636     -0.89504107647773623
    0.22129994224848493     0.45560732314902141     -0.51193563653045981    -5.7395831402299278     -1.1122925103904031
    0.26922618307692664     0.72749364473832012     -0.60697104195508922    -6.2495197653332042     -1.3531588560440539
    0.30355819320945776     0.94292576079011325     -0.6353699244191795     -6.3941722372596717     -1.5257824749693465
    0.3124801573016831      1                       -0.63665476032305746    -6.4005828625858454     -1.5406984905484249
DATA_STOP
```

Fig. 4.4: Brachistichrone problem file output.

4.1.9 Additional Tutorials

The C++ version of CSALT is distributed with many example problems. The examples are located in the folder gmat/src/csaltTester/src/TestOptCtrl/src. The subfolders "drivers" and "pointpath" contain the test drivers and point and path objects, respectively.

4.2 Tutorial: Setting up an Optimal Control Problem in MATLAB

In this section, we describe how to set up an optimal control problem using the Brachistichrone problem as an example. We begin by providing an overview of the problem setup procedures for a general optimal control problem, then present the Brachistichrone problem statement, followed by the MATLAB source code for the Brachistichrone problem.

4.2.1 MATLAB User Interface Component Overview

The MATLAB API is composed of several classes shown in Fig. 4.5. The user classes that define the API are Trajectory, Phase, UserPathFunction, and UserPointFunction. Note that UserFunction is an abstract base class that contains commonality between path and point functions. The user implements path and boundary functions for a problem by deriving from the UserPathFunction and UserPointFunction base classes as shown in Fig. 4.5.

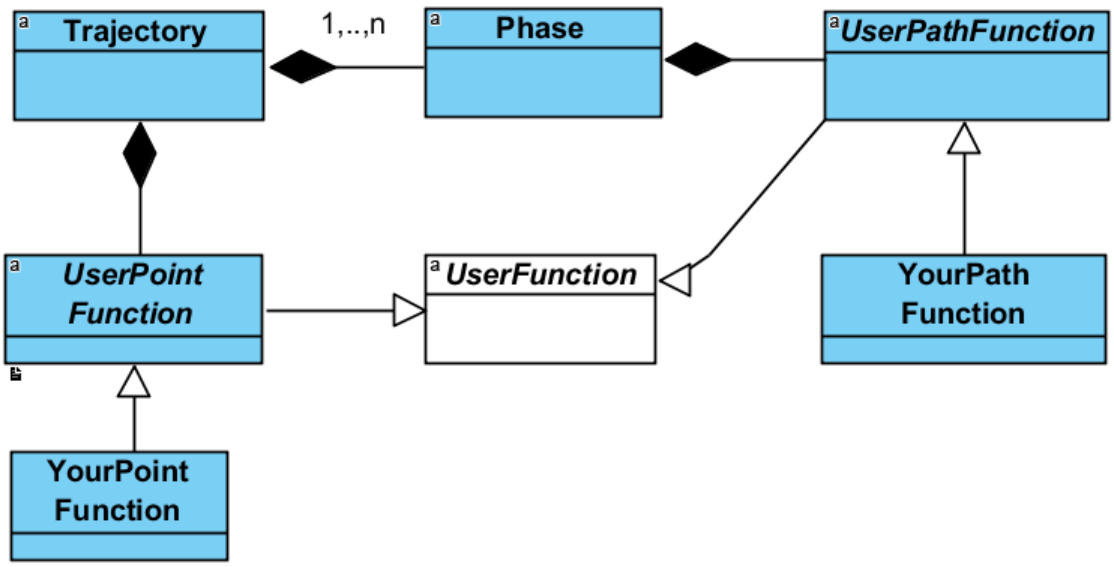

Fig. 4.5: CSALT MATLAB user classes.

The responsibilities of classes in the CSALT API are highlighted in Table 4.2.

Table 4.2: CSALT MATLAB API classes.

Class	Required?	Responsibility
Trajectory	Yes	Trajectory is the container for all phases in the problem and for the UserPointFunction and UserPathFunction objects. Internally, the Trajectory concatenates functions and Jacobians provided on different phases and from UserPointFunction.
Phase	Yes	Phase defines the transcription employed and manages/concatenates the defect constraints, integral cost, and algebraic path functions.
UserPathFunction	Yes	Computes path function values (dynamics, integral cost, and algebraic path constraints) and, optionally, analytic Jacobians of path functions. This class provides the generic interface for those computations. The user provides problem-specific functions by deriving a path function class from the UserPathFunction base class.
UserPointFunction	Yes	Computes boundary functions (cost and algebraic point constraints) and, optionally, analytic boundary Jacobians. This class provides the generic interface for those computations. The user provides problem-specific functions by deriving a point function class from the UserPointFunction base class.

4.2.2 Overview of CSALT Configuration Process

The key steps in setting up an optimization problem in CSALT are:

1. Configure the Path Functions
2. Configure the Boundary Functions
3. Configure the Phases and Transcription
4. Configure the Trajectory
5. Solve the problem and examine results

When the tutorial is complete, the user will have three files: a path function object, a point function object, and a driver script that contains the contents of steps 3-4. The user runs the tutorial by executing the driver script.

To solve the Brachistichrone problem using CSALT, let's begin with the Brachistichrone problem statement:

Minimize the final time

$$\text{Minimize } J = t_f$$

Subject to the dynamics

$$\begin{aligned}\dot{x} &= v \sin u \\ \dot{y} &= v \cos u \\ \dot{v} &= g_0 \cos u\end{aligned}$$

the initial boundary conditions

$$\begin{aligned}t_0 &= 0 \\ x\left(t_0\right) &= 0 \\ y\left(t_0\right) &= 0 \\ v\left(t_0\right) &= 0\end{aligned}$$

the final boundary conditions

$$
\begin{aligned}
0 \leq t_f &\leq 10 \\
x\left(t_f\right) &= 1 \\
-10 \leq y\left(t_f\right) &\leq 10 \\
-10 \leq v\left(t_f\right) &\leq 0
\end{aligned}
$$

and the following bound constraints on the state and control at any time:

$$
\begin{aligned}
-10 \leq x(t) &\leq 10 \\
-10 \leq y(t) &\leq 10 \\
-10 \leq v(t) &\leq 10 \\
-10 \leq u(t) &\leq 10
\end{aligned}
$$

4.2.3 Configuring the CSALT Path

CSALT MATLAB is distributed with two folders as shown in Fig. 4.6.

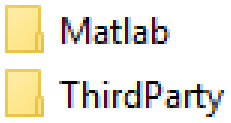

Fig. 4.6: Directories in which CSALT MATLAB is distributed.

The code for CSALT and tutorial problems is contained in the MATLAB folder, and third-party files such as SNOPT are contained in the ThirdParty folder. All files in both directories must be added to the user's MATLAB path.

4.2.4 Step 1: Configure the Path Functions

Path functions, including dynamics, algebraic path constraints, and, if applicable, an integral cost function, are implemented by deriving a class from the CSALT UserPathFunction base class and implementing the EvaluateFunctions() and EvaluateJacobians() methods. The EvaluateFunctions() method is required. The EvaluateJacobians() method is required but may be empty. If EvaluateJacobians() does not set a particular Jacobian, CSALT will finite-difference that Jacobian.

The dynamics model for the Brachistichrone problem is:

$$
\begin{aligned}
\dot{x} &= v \sin u \\
\dot{y} &= v \cos u \\
\dot{v} &= g_0 \cos u
\end{aligned}
$$

The EvaluateFunctions() method that implements those dynamics is shown below. For this example, the Path Function class is named BrachistichronePathObject. Create a file named BrachistichronePathObject.m and configure the contents as shown below. Note in MATLAB the name of a class .m file must match the name of the class.

```
classdef BrachistichronePathObject < userfunutils.UserPathFunction
    % Path functions for Brachistichrone problem.

    properties
```

(continues on next page)

(continued from previous page)

```
    % Set the gravity coefficient
    gravity = -32.174
    end

    methods

    function EvaluateFunctions(obj)

        %  Get the state and control
        stateVec = obj.GetStateVec();
        controlVec = obj.GetControlVec();
        v = stateVec(3);
        u = controlVec(1);

        %  Set the dynamics functions
        obj.SetDynFunctions([v*sin(u);
            v*cos(u);
            obj.gravity*cos(u)]);

    end

    function EvaluateJacobians(obj)

        % Get the state and control
        stateVec = obj.GetStateVec();
        controlVec = obj.GetControlVec();
        v = stateVec(3);
        u = controlVec(1);

        % Set the Jacobian of the dynamics w/r/t state
        obj.SetDynStateJac([0 0 sin(u);
            0 0 cos(u);
            0 0 0]);

        % Set the Jacobian of the dynamics w/r/t control
        obj.SetDynControlJac([v*cos(u);
            -v*sin(u);
            -obj.gravity*sin(u)]);

        % Set the Jacobian of dynamics w/r/t time
        obj.SetDynTimeJac([0;0;0]);

    end
    end
end
```

4.2.5 Step 2: Configure the Boundary Functions

Boundary functions, including algebraic constraints and cost function, are implemented by deriving a class from the CSALT UserPointFunction base class and implementing the EvaluateFunctions() method. The EvaluateFunctions() method is required. Note the MATLAB version of CSALT does not currently support analytic Jacobians of boundary functions. The EvaluateJacobians() method should still be provided but left empty.

The boundary functions for the Brachistochrone problem are

$$\text{Minimize } J = t_f$$

subject to the initial boundary conditions

$$\begin{aligned} t_0 &= 0 \\ x\left(t_0\right) &= 0 \\ y\left(t_0\right) &= 0 \\ v\left(t_0\right) &= 0 \end{aligned}$$

and the final boundary conditions

$$\begin{aligned} 0 \leq t_f &\leq 10 \\ x\left(t_f\right) &= 1 \\ -10 \leq y\left(t_f\right) &\leq 10 \\ -10 \leq v\left(t_f\right) &\leq 0 \end{aligned}$$

The EvaluateFunctions() method that implements those boundary functions is shown below. For this example, the Point Function class is named BrachistichronePointObject. Create a file named BrachistichronePointObject.m and configure the contents as shown below. Note in MATLAB the name of a class .m file must match the name of the class.

```
classdef BrachistichronePointObject < userfunutils.UserPointFunction
    % Boundary functions for Brachistichrone problem.

    methods

    function EvaluateFunctions(obj)

        %  Extract the initial and final time and state for
        %  phase 1 (the only phase in the problem)
        initTime = obj.GetInitialTime(1);
        finalTime = obj.GetFinalTime(1);
        initState = obj.GetInitialStateVec(1);
        finalState = obj.GetFinalStateVec(1);

        % Set the cost function
        obj.SetCostFunction(finalTime);

        % Set the boundary functions
        obj.SetAlgFunctions([initTime;
            finalTime;...
            initState;
            finalState]);

        % If initializing, set the bounds using the same order
        % as in the call to SetAlgFunctions()
```

(continues on next page)

(continued from previous page)

```
        if obj.IsInitializing
            obj.SetAlgFuncLowerBounds([0 0   0 0  0  1 -10 -10]');
            obj.SetAlgFuncUpperBounds([0 100 0 0  0  1 10 0 ]');
        end

    end

    function EvaluateJacobians(~)

    end

    end
end
```

4.2.6 Step 3: Configure the Phases and Transcription

The next step in configuring the Brachistichrone problem is to set up a Phase. (This example is a single-phase problem.) The Phase object contains the configuration for the selected transcription, the bounds on optimization parameters (e.g., time, state, and control), and the initial guesses for those parameters. In this example, the phase is configured to use Radau orthogonal collocation with two equally spaced mesh intervals, each containing five points. (The mesh interval configuration and initial-guess options are presented in more detail in *Phase Reference*). Create a file named Brachistichrone_Driver.m and configure the contents as shown below. Code written in steps 3-5 should be added to the driver script file.

```
%% ===== Initializations
clear all
global traj  % Required for SNOPT function wrapper. Can't pass a Trajectory to SNOPT.

% Import CSALT classes
import exec.Trajectory
import exec.RadauPhase

%% =====  Define Properties for the Phase

% Create a Radau phase and set preliminary mesh configuration
phase1 = RadauPhase;
phase1.initialGuessMode = 'LinearNoControl';
meshIntervalFractions = [-1 0 1];
meshIntervalNumPoints = [5 5];

%  Set time bounds and guesses
initialGuessTime  = 0;
finalGuessTime      = .3;
timeLowerBound      = 0;
timeUpperBound      = 100;

%  Set state bounds and guesses
numStateVars        = 3;
stateLowerBound   = [-10 -10 -10]';
initialGuessState = [0 0 0]'; % [x0 y0 v0]
finalGuessState   = [.4 -.5 -1]'; % [xf yf vf]
```

(continues on next page)

(continued from previous page)

```
stateUpperBound   = [10 0 0]';

%  Set control bounds (guess is set to 1s)
numControlVars = 1;
controlUpperBound = 10;
controlLowerBound = -10;

% Call set methods to configure the data
phase1.SetNumStateVars(numStateVars)
phase1.SetNumControlVars(numControlVars);
phase1.SetMeshIntervalFractions(meshIntervalFractions);
phase1.SetMeshIntervalNumPoints(meshIntervalNumPoints);
phase1.SetStateLowerBound(stateLowerBound)
phase1.SetStateUpperBound(stateUpperBound)
phase1.SetStateInitialGuess(initialGuessState)
phase1.SetStateFinalGuess(finalGuessState)
phase1.SetTimeLowerBound(timeLowerBound)
phase1.SetTimeUpperBound(timeUpperBound)
phase1.SetTimeInitialGuess(initialGuessTime)
phase1.SetTimeFinalGuess(finalGuessTime)
phase1.SetControlLowerBound(controlLowerBound)
phase1.SetControlUpperBound(controlUpperBound)
```

4.2.7 Step 4: Configure the Trajectory

The next step is to add the path function, boundary function, and phase objects to a Trajectory.

```
%% =====  Define Properties for the Trajectory

%  Create trajectory and configure user function names
traj = Trajectory();

% Add the path, boundary function, and phases to the trajectory
import BrachistichronePathObject
import BrachistichronePointObject
traj.pathFunctionObject = BrachistichronePathObject();
traj.pointFunctionObject = BrachistichronePointObject();
traj.phaseList = {phase1};

% Set other settings
traj.showPlot = false();
traj.plotUpdateRate = 100;
traj.costLowerBound = -Inf;
traj.costUpperBound = Inf;
traj.maxMeshRefinementCount = 2;
```

4.2.8 Step 5: Run the Optimizer and Examine the Solution

Now that the path, boundary, phase, and trajectory objects are configured, we are ready to optimize the problem and write the solution to a file. Add the following lines to your driver script, and then run the script in MATLAB.

```
%% =====  Run The Optimizer and plot the solution
traj.Initialize();
[z,info] = traj.Optimize();

stateSol = traj.GetStateArray(1);
plot(stateSol(:,1),stateSol(:,2))
grid on
```

The console output for the Brachistichrone problem solved using SNOPT is shown in Fig. 4.7.

```
   14      1  1.0E+00     15  3.5E-05 (5.1E-05) 3.1248582E-01    22 2.2E-03
   15      1  1.0E+00     16  7.2E-05 (3.8E-05) 3.1248338E-01    22 2.2E-03
   16      1  1.0E+00     17  1.0E-04 (2.5E-05) 3.1248168E-01    22 2.2E-03
   17      1  1.0E+00     18  1.0E-04 (1.8E-05) 3.1248055E-01    22 2.2E-03
   18      1  1.0E+00     19  1.7E-05 (6.6E-06) 3.1248019E-01    22 2.2E-03
   19      1  1.0E+00     20  7.6E-07 (3.0E-06) 3.1248014E-01    22 2.2E-03
Major Minors     Step   nCon Feasible  Optimal  MeritFunction    nS Penalty
   20      1  1.0E+00     21 (8.6E-08)(6.3E-07) 3.1248013E-01    22 2.2E-03
Major Minors     Step   nCon Feasible  Optimal  MeritFunction    nS Penalty
   20      1  1.0E+00     21 (8.6E-08)(6.3E-07) 3.1248013E-01    22 2.2E-03

SNOPTA EXIT   0 -- finished successfully
SNOPTA INFO   1 -- optimality conditions satisfied
Problem name
No. of iterations                 70   Objective value      3.1248012683E-01
No. of major iterations           20   Linear objective     0.0000000000E+00
Penalty parameter          2.176E-03   Nonlinear objective  3.1248012683E-01
No. of calls to funobj            45   No. of calls to funcon             45
Calls with modes 1,2 (known g)    21   Calls with modes 1,2 (known g)     21
Calls for forward differencing     0   Calls for forward differencing      0
Calls for central differencing     0   Calls for central differencing      0
No. of superbasics                22   No. of basic nonlinears            73
No. of degenerate steps            0   Percentage                        .00
Max x                     97 6.4E+00   Max pi                    78 1.0E+00
Max Primal infeas          4 1.4E-17   Max Dual infeas           46 1.3E-06
Nonlinear constraint violn    1.2E-06
```

Fig. 4.7: Brachistichrone problem optimization MATLAB console output.

4.2.9 Additional Tutorials

The MATLAB version of CSALT is distributed with many tutorials in the lowthrustMatlabTestProblems folder.

4.3 Tutorial: Setting up an Optimal Control Problem in GMAT

In this section, we describe how to set up an optimal control problem in GMAT using a simple, single-phase, low-thrust optimization problem. We begin by providing an overview of the GMAT components used to solve optimal control problems, then present an overview of the steps to set up a problem, and finally walk through the GMAT script for the problem configuration.

4.3.1 GMAT Script User Interface Component Overview

The GMAT Optimal Control subsystem is composed of several Resources shown in Fig. 4.8. The user Resources are Trajectory, Phase, OptimalControlFunction, OptimalControlGuess, CustomLinkageConstraint, DynamicsConfiguration, EMTGSpacecraft, Spacecraft, ForceModel, ChemicalTank, and ElectricTank. The diagram uses the UML "has-a" relationship that employs a diamond to indicate an "owned" object. In this case, for example, a Trajectory Resource "has a" list of OptimalControlFunctions, a list of Phases, and an OptimalControlGuess. Bulleted lists in the diagram indicate key local data. For example, convergence tolerances are set on the Trajectory Resource.

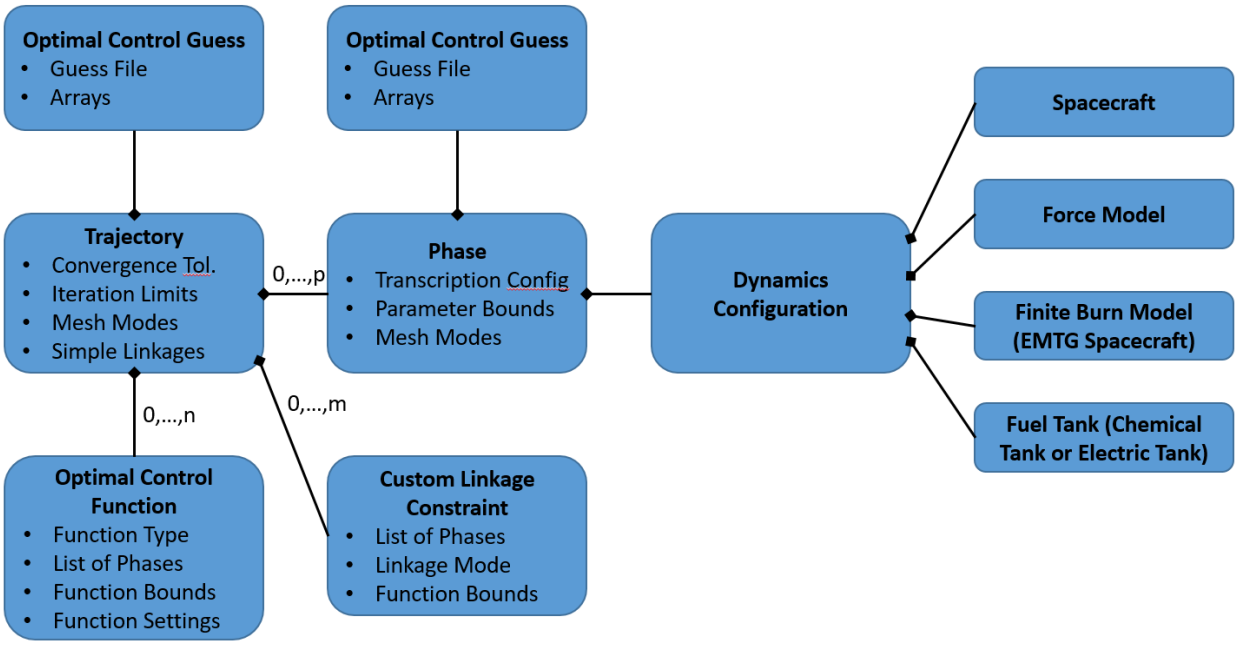

Fig. 4.8: GMAT Optimal Control interface.

A brief description of the new Resources and Command introduced in GMAT Optimal Control is contained in Table 4.3.

Table 4.3: GMAT Optimal Control resources and command.

Resource	Responsibility
Trajectory	The container for all phases and optimal control functions in the optimization problem and configuration for settings that apply to the whole problem (such as convergence tolerances and iterations limits).
Phase	A phase is used to model a segment of a trajectory that can be modeled using a single dynamics model. (If the dynamics must change, the problem requires a different phase for each subsection of the trajectory).
OptimalControlGuess	An object that provides a guess to the optimal control subsystem. The OptimalControlGuess supports file-based and GMAT-Array-based guess data. Guess information can be provided on a Phase or Trajectory.
CustomLinkageConstraint	A constraint that defines custom time and state linkages (i.e., not full state linkage) between phases or the start and end of a single phase. For example, if a phase's time duration must be less than some known duration, a custom linkage constraint is used. Other examples of custom linkage constraints include constraining a phase boundary to a known initial or final state and constraining mass change between different points in a trajectory.
DynamicsConfiguration	A configuration of a spacecraft, force model (orbital dynamics), and thruster model, and fuel tank used to model the spacecraft dynamics.
OptimalControlFunction	An optimal control function models algebraic constraints or cost functions (must be scalar in the case of a cost function). The OptimalControlFunction type field sets the function type, e.g., IntegratedFlyby, PatchedConicLaunch, Expression, etc.
EMTGSpacecraft	An interface to EMTG thruster models.
Collocate	The command to solve the optimal control problem

4.3.2 Overview of GMAT Script Configuration Process

The basic steps to set up an optimal control problem in GMAT are:

1. Configure path functions and dynamics
    a. Configure spacecraft
    b. Configure fuel tank
    c. Configure orbital dynamics and thruster models
    d. Configure the DynamicsConfiguration
2. Configure the OptimalControlGuess
3. Configure the boundary functions
4. Configure the phases
5. Configure the trajectory
6. Run the optimizer and examine the solution

Note that the GMAT script parser is a two-pass parser, and the items above need not be implemented in the order shown except that the last step (executing the optimizer) must be performed last. The sections below walk through a script configuration for a simple optimal control problem. The GMAT tutorial problem statement is:

Maximize the final distance from the Sun, i.e.,

$$\text{Minimize } J = -\sqrt{x\left(t_f\right)^2 + y\left(t_f\right)^2 + z\left(t_f\right)^2}$$

subject to the initial boundary conditions (Sun-centered MJ2000Eq reference frame)

$$
\begin{aligned}
t_0 &= 30542 \text{ A1MJD} \\
x\left(t_0\right) &= 125291184 \text{ km} \\
y\left(t_0\right) &= -75613036 \text{ km} \\
z\left(t_0\right) &= -32788527 \text{ km} \\
\dot{x}\left(t_0\right) &= 13.438 \text{ km/s} \\
\dot{y}\left(t_0\right) &= 25.234 \text{ km/s} \\
\dot{z}\left(t_0\right) &= 10.903 \text{ km/s} \\
m\left(t_0\right) &= 3930 \text{ kg}
\end{aligned}
$$

and the final boundary condition (a time of flight of 250 days)

$$
t_f = 30792 \text{ A1MJD}
$$

The initial conditions above start at about 1 AU from the Sun in an Earth-like orbit. We will configure GMAT to use a Sun-centered dynamics model and determine the thrust history to maximize the spacecraft's position from the Sun after 250 days.

Now let's walk through the GMAT script configuration for this problem.

4.3.3 Step 1: Configure Path Functions and Dynamics

The Resource that defines the dynamics for a phase is the DynamicsConfiguration Resource. To set up a DynamicsConfiguration, attach pre-configured Spacecraft, ForceModel, Tank, and EMTGSpacecraft Resources. The example below illustrates how to configure those models.

Configure the Spacecraft

Configuring a spacecraft for this example is relatively simple. When we configure the guess data later in the tutorial, which includes spacecraft state and mass, those settings on the spacecraft will be set according to the guess, so setting state and epoch information is not required here. However, the total mass of the spacecraft must be set along with fuel tank masses so that they can be updated by CSALT during optimization. Fuel tanks are required whenever a burn is being used along the trajectory. Note that the total mass of the spacecraft must at least be the upper bound of the mass constraint set later in phases, but the fuel tanks themselves will be internally set to allow negative mass values to allow CSALT to attempt to reach convergence. We need to create a spacecraft and its fuel tank.

```
% Create a spacecraft named aSat.  Guess is set later.
Create Spacecraft aSat
GMAT aSat.DryMass = 1000
GMAT aSat.Tanks = {ChemTank}

% Create a chemical tank named ChemTank.
Create ChemicalTank ChemTank;
GMAT ChemTank.FuelMass = 4000;
GMAT ChemTank.Volume = 4;
GMAT ChemTank.FuelDensity = 1260;
```

Configure Orbital Dynamics Models

Below is the script configuration for a simple Sun-centered dynamics model with Earth, Sun, and Moon point masses included in the model.

```
% Create an orbit dynamics model with Earth, Sun, Moon point mass
Create ForceModel DeepSpaceForceModel;
GMAT DeepSpaceForceModel.CentralBody = Sun;
GMAT DeepSpaceForceModel.PointMasses = {Sun,Earth,Luna};
GMAT DeepSpaceForceModel.Drag = None;
GMAT DeepSpaceForceModel.SRP = On;
```

Configure the Thruster Models (EMTGSpacecraft)

The thrust model is configured by setting up EMTG options files and setting those files on an EMTGSpacecraft Resource. The EMTG options files contain the configuration for the thrust model. See *EMTG Spacecraft Reference* for more information.

```
% Create an EMTGSpacecraft Resource
Create EMTGSpacecraft emtgThrustModel;
emtgThrustModel.SpacecraftFile = SpacecraftModel.emtg_spacecraftopt
emtgThrustModel.DutyCycle = 0.9
```

Warning: This EMTG interface is alpha, and will be replaced by a new interface to EMTG models that is production quality.

Note that the EMTGSpacecraft SpacecraftFile field specifies the path relative to the GMAT /data/emtg directory.

Configure the Dynamics Configuration

Now that the Spacecraft, ChemicalTank, ForceModel, and EMTGSpacecraft Resources are configured, they are added to a DynamicsConfiguration Resource.

```
% Create a DynamicsConfiguration object and add Spacecraft,
% ForceModel, and Thrust model
Create DynamicsConfiguration SunThrustDynConfig
SunThrustDynConfig.ForceModels = {DeepSpaceForceModel}
SunThrustDynConfig.Spacecraft = {aSat}
SunThrustDynConfig.FiniteBurns = {emtgThrustModel}
SunThrustDynConfig.EMTGTankConfig = {ChemTank}
```

These Resources are added to the Phase Resources later in the tutorial.

4.3.4 Step 2: Configure the Guess

One significant difference between GMAT parameter optimization interfaces (Vary, Nonlinear Constraint, Minimize, etc.) and the optimal control subsystem is that a guess for time, state, and control over the whole trajectory is required to use the optimal control subsystem. Guesses can be generated from a GMAT script and propagation, external tools, or quasi-random number generation processes.

The OptimalControlGuess Resource supports several guess data types, including a file-based guess and GMAT-Array-based guesses. Here, we use a file-based guess using an Optimal Control History (.och) file.

```
% Create a guess object that uses a file-based guess source
Create OptimalControlGuess trajectoryGuess
trajectoryGuess.Type = CollocationGuessFile
trajectoryGuess.FileName = ../data/misc/GuessWithUnityControl.och
```

The guess file contains a header with metadata that describes the data content. Three rows of data are shown below, with line wrapping to show all of the data contained in the rows. The rows contain time, Cartesian state, mass, and control. Currently EME2000 is the only supported reference frame, and A1ModJulian is the only supported time system when using a file-based guess. If the guess file comes from a previous run's output file, the file will also contain the exit condition data from the run. It is safe to leave this data in the file when used as a guess file, as it will be ignored by CSALT when collecting the guess data.

The guess source may include data at times that are beyond the initial guesses for the initial and final epochs of the problem. However, because interpolation is used to obtain guess values at times between or beyond the times listed in the guess, it is strongly recommended to provide a guess whose timespan completely encompasses the initial guess for the initial and final epochs.

```
META_START
    CENTRAL_BODY = Sun
    REF_FRAME = EME2000
    TIME_SYSTEM = A1ModJulian
    NUM_STATES = 7
    NUM_CONTROLS = 3
    NUM_INTEGRALS = 0
META_STOP

DATA_START
30537.58130403295           120045407.6646147       -84769659.07938783      -36758301.
↪62764073      13.68515738249333       26.23213523803535       10.71343366494688       ␣
↪3928.8979              1                       0                       0
30537.68130403294           120166876.6672654       -84550219.83076148      -36666345.
↪46750365      14.29149185537749       24.96927614272317       10.58473151780541       ␣
↪3928.8979              1                       0                       0
30537.78130403294           120291284.2924708       -84335930.88030888      -36575177.
↪55088668      14.47799877262803       24.68188855218928       10.5266647230637        ␣
↪3928.8979              1                       0                       0
.
.
.
DATA_STOP
```

4.3.5 Step 3: Configure the Phases

The Phase Resource includes settings for the transcription, bounds on optimization parameters during the phase, bounds on the phase initial and final epoch, the guess for the initial and final epoch, and the OptimalControlGuess Resource that contains the guess for the phase. The description for some of these settings is beyond the scope of the Tutorial. See *Phase Reference* for more details.

```
%  Create a phase and set transcription and parameter bounds
Create Phase thePhase
thePhase.Type = RadauPseudospectral
thePhase.ThrustMode = Thrust
thePhase.DynamicsConfiguration = SunThrustDynConfig
thePhase.SubPhaseBoundaries = [-1   -0.75 -0.5 -0.25 0 0.25 .5 0.75 1]
thePhase.PointsPerSubPhase = [7 7 7 7 7 7 7 7]
thePhase.GuessSource = trajectoryGuess
thePhase.EpochFormat = A1ModJulian
thePhase.EpochLowerBound = 30500.0
thePhase.EpochUpperBound = 31000.0
thePhase.InitialEpoch = 30542.0 % initial guess
thePhase.FinalEpoch = 30792.0 % initial guess
thePhase.StateLowerBound = [ -1.49598e+09 -1.49598e+09 -1.49598e+09 ...
                             -100 -100 -100 1]
thePhase.StateUpperBound = [ 1.49598e+09  1.49598e+09 1.49598e+09 ...
                             100 100 100 5000]
thePhase.ControlLowerBound = [ -1  -1  -1]
thePhase.ControlUpperBound = [ 1  1  1]
```

4.3.6 Step 4: Configure the Boundary Functions

The tutorial problem has boundary conditions on the initial state and a phase-duration constraint, which are configured using CustomLinkageConstraint Resources illustrated in the next two subsections.

Set the Initial Conditions

For convenience, we repeat the initial boundary conditions for the tutorial:

$$
\begin{aligned}
t_0 &= 30542 \text{ A1MJD} \\
x\left(t_0\right) &= 125291184 \text{ km} \\
y\left(t_0\right) &= -75613036 \text{ km} \\
z\left(t_0\right) &= -32788527 \text{ km} \\
\dot{x}\left(t_0\right) &= 13.438 \text{ km/s} \\
\dot{y}\left(t_0\right) &= 25.234 \text{ km/s} \\
\dot{z}\left(t_0\right) &= 10.903 \text{ km/s} \\
m\left(t_0\right) &= 3930 \text{ kg}
\end{aligned}
$$

To apply these constraints, configure a CustomLinkageConstraint Resource as shown below.

```
% Create constraint that applies the initial conditions
Create CustomLinkageConstraint initialConditions;
initialConditions.ConstraintMode = Absolute
```

(continues on next page)

(continued from previous page)

```
initialConditions.InitialPhase = thePhase;
initialConditions.InitialPhaseBoundaryType = Start;
initialConditions.SetModelParameter('TimeLowerBound', A1ModJulian, 30542)
initialConditions.SetModelParameter('TimeUpperBound', A1ModJulian, 30542)
initialConditions.SetModelParameter('PositionLowerBound', [125291184 -75613036 -
↪32788527])
initialConditions.SetModelParameter('PositionUpperBound', [125291184 -75613036 -
↪32788527])
initialConditions.SetModelParameter('VelocityLowerBound', [13.438 25.234 10.903])
initialConditions.SetModelParameter('VelocityUpperBound', [13.438 25.234 10.903])
initialConditions.SetModelParameter('MassLowerBound', 3930)
initialConditions.SetModelParameter('MassUpperBound', 3930)
```

Configure the phase duration constraint

For convenience, we repeat the phase duration constraint for the tutorial:

$$t_f = 30792 \text{ A1MJD}$$

This final time constraint corresponds to 250 days past the initial time. To apply this constraint, configure a CustomLinkageConstraint Resource as shown below.

```
% Create a constraint on Phase duration. Duration must be 250 days
Create CustomLinkageConstraint duration_Thrust;
duration_Thrust.ConstraintMode = Difference;
duration_Thrust.InitialPhase = thePhase;
duration_Thrust.InitialPhaseBoundaryType = Start;
duration_Thrust.FinalPhase = thePhase;
duration_Thrust.FinalPhaseBoundaryType = End;
duration_Thrust.SetModelParameter('TimeLowerBound', 'ElapsedDays', 250)
duration_Thrust.SetModelParameter('TimeUpperBound', 'ElapsedDays', 250)
```

Setting the Cost Function

The cost function for this tutorial is:

$$\text{Minimize } J = -\sqrt{x\left(t_f\right)^2 + y\left(t_f\right)^2 + z\left(t_f\right)^2}$$

This specific cost function is built into GMAT. To configure the cost function, use the following script.

```
thePhase.BuiltInCost = 'RMAGFinal'
```

4.3.7 Step 5: Configure the Trajectory

The last configuration step is to create a Trajectory Resource and add the guess, phase, and constraint Resources and include a few extra settings. Note that there are significantly more Trajectory fields that may be set (see *Trajectory Reference*), but, for a minimal example such as this tutorial, it is sufficient to set the fields shown below.

```
Create Trajectory traj
traj.PhaseList = {thePhase}
traj.GuessSource = trajectoryGuess;
traj.SolutionFile = 'Solution.och';
traj.StateScaleMode = Canonical;
traj.MassScaleFactor = 4000;
traj.MaxMeshRefinementIterations = 5;
traj.CustomLinkages = {initialConditions, duration_Thrust}
traj.MajorIterationsLimits = [1000]
traj.TotalIterationsLimits = [20000]
traj.FeasibilityTolerances = [1e-7]
traj.MajorOptimalityTolerances = [1e-4]
traj.OptimizationMode = {Maximize}
```

4.3.8 Step 6: Run the Optimizer and Examine the Solution

The Collocate command is used to run the problem. To optimize a problem, use the Collocate command including the name of the Trajectory to be optimized. Note that the Collocate command is placed after a BeginMissionSequence line in a GMAT script.

```
Collocate traj
```

The solution may be examined in several ways. First, the GMAT console displays information about the optimization process, including SNOPT iteration-by-iteration output and a problem summary once the optimization is completed. The summary contains information on optimizer status and constraint violations. A snippet of the console output for this example is shown below. (The output was reformatted slightly to fit on the page.)

```
================================================================================
==== Constraint Summary                                                  ====
================================================================================

Max. Simple Linkage Constraint Error :   0.000000000000e+00
Max. Custom Linkage Constraint Violation :   2.883374691010e-06
Max. Custom Constraint :   1.252911840000e+08
Max. Defect Constraint :   2.220179595724e-11
Cost Function Value :   1.554290155413e+00
Convergence Status : Optimizer converged

...
...

================================================================================
==== Custom Linkage Constraints                                          ====
================================================================================

Constrained Phases: thePhase.Start
```

(continues on next page)

(continued from previous page)

```
                Lower Bounds  Constraint Value  Upper Bounds
Time = 3.054200000000e+04  3.054200000000e+04  3.054200000000e+04
X = 1.252911840000e+08  1.252911840000e+08  1.252911840000e+08
Y = -7.561303600000e+07  -7.561303600000e+07  -7.561303600000e+07
Z = -3.278852700000e+07  -3.278852700000e+07  -3.278852700000e+07
VX =  1.343800000000e+01  1.343800000000e+01  1.343800000000e+01
VY =  2.523400000000e+01  2.523400000000e+01  2.523400000000e+01
VZ = 1.090300000000e+01  1.090300000000e+01  1.090300000000e+01
Mass = 3.930000000000e+03  3.930000000000e+03   3.930000000000e+03

Constrained Phases: thePhase.Start to thePhase.End

                Lower Bounds  Constraint Value  Upper Bounds
Time = 2.500000000000e+02  2.500000000000e+02  2.500000000000e+02

Max. Custom Linkage Constraint Violation:   2.883374691010e-06
```

State and control history of either the most feasible or best feasible solution is output in an optimal control history (.och) file. In the case where the optimizer converged, the converged solution is printed. If the optimizer did not converge, but one or more feasible solutions were found in the final mesh iteration, the feasible solution with the minimum cost function value is printed. Finally, if no feasible solutions were found during the last mesh iteration, the solution closest to achieving feasibility is printed.

The format of a .och file is described in *Step 2: Configure the Guess*. The only difference from the guess format is that here, exit conditions are provided. The exit conditions contain various optimizer metrics for the solution printed, including whether the solution was a converged, feasible but not optimal, or infeasible solution. For this example, the exit conditions and the first and last lines of the .och data block are:

```
30542                      125291184.00000001         -75613036                  -
↪32788526.999997117         13.438000000000237         25.233999999999995         10.903␣
↪                     3930                       0.2065315270878699         0.
↪89816159643679194        0.38813718716109147
...
...
30791.999999999993         -167583548.9232918         -75171810.936078712        -
↪32391739.681220464         9.6520747091844967         -20.867812869758279        -9.
↪0315868085278339        3822.5243683003659        -0.89871563615191197       -0.
↪40299676014008162       -0.17294310281002576
```

Traditional GMAT Reports and Plots may also be used to examine the solution. A plot of the optimized solution is shown in Fig. 4.9.

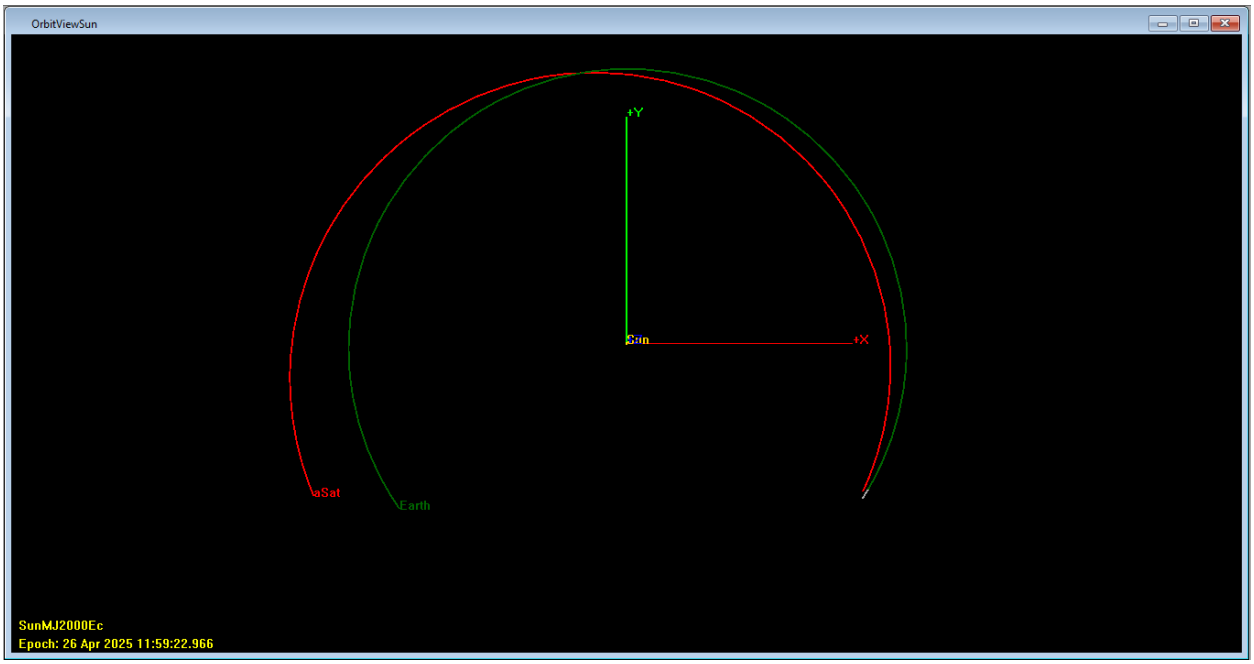

Fig. 4.9: GMAT plot generated from GMAT Optimal Control.

4.4 Trajectory Reference

A Trajectory is the primary manager class for optimal control problems and coordinates and concatenates data provided by phases and point functions. There are some differences between the implementation of Trajectory *resource* in GMAT and the Trajectory *class* in CSALT. In GMAT, much of the configuration is performed automatically by the system, and GMAT supports built-in cost and constraint functions. On the other hand, CSALT is a low-level library and does not contain built-in models.

In both GMAT and CSALT, the following parameters are set on the Trajectory:

- Phases to be included in the problem
- Feasibility tolerances
- Optimality tolerances
- Mesh refinement iteration limits
- Optimization iteration limits
- Cost function bounds
- Optimization mode (i.e., maximize, minimize, or feasible point)
- Guess data

In the GMAT interface, the Trajectory resource additionally supports:

- Constraints to be included in the solution, including:
  - Full state linkage constraints
  - Custom linkage constraints
  - General optimal control constraints
- Scaling configuration

- Output file and publisher configurations

In CSALT interfaces, the Trajectory class additionally supports:

- Point function pointer (class used to define optimal control point functions)
- Path function pointer (class used to define optimal control path functions)

This reference material discusses how to configure a Trajectory in CSALT (C++ and MATLAB) and GMAT. The first sections contain interfaces common to both GMAT and CSALT. The subsequent sections contain information relevant only to GMAT or CSALT. In

4.4.1 Setting the Phase Objects

Phases are included in an optimization by adding them to the Trajectory. The examples below show how to include phases in C++ and in a GMAT script. The Phase object is described in detail in *Phase Reference*.

GMAT Script Example

One or more Phase resources may be provided to a Trajectory in one script line.

```
traj.PhaseList = {phase1,phase2}
```

C++ Example

The Phase objects are added to the Trajectory as a std::vector of phase pointers. In the example below, we assume the Phase objects have been created and configured and illustrate how to provide the phases to the Trajectory object.

```
// Function Prototype:
// virtual void SetPhaseList(std::vector<Phase*> pList);

// Example:
std::vector<Phase*> phaseList;
phaseList.push_back(phase1);
phaseList.push_back(phase2);
traj->SetPhaseList(phaseList);
```

MATLAB Example

One or more Phase objects may be provided to a Trajectory in one script line.

```
traj.phaseList = {phase1,phase2};
```

4.4.2 Setting Cost Function Bounds

Cost function bounds set minimum and/or maximum values on the cost function.

GMAT Script Example

In GMAT, the cost function bounds are always +/-inf and the user does not explicitly set them.

C++ Example

Cost function bounds are set using the SetCostLowerBound and SetCostLowerBound methods. The code snippet below shows how to set the cost function bounds to -10 and 10, respectively.

```
// Function Prototype:
// virtual void SetCostLowerBound(Real costLower);
// virtual void SetCostUpperBound(Real costUpper);

// Example:
Real lowerBound = -10;
Real upperBound = 10;
traj = new Trajectory();
traj.SetCostLowerBound(lowerBound);
traj.SetCostUpperBound(upperBound);
```

MATLAB Example

```
traj.costLowerBound = -10;
traj.costUpperBound = 10;
```

4.4.3 Setting Optimization Convergence Tolerances

CSALT supports several convergence tolerance settings, including feasibility tolerance and optimality tolerance. Examples are shown below for each setting. Optimization tolerances are vectors of real numbers, where entry i in the vector corresponds to the tolerance for mesh iteration i. This allows for rapid, approximate solutions (i.e., coarse tolerances) during early mesh iterations and more stringent tolerances in later mesh iterations. During early mesh refinement iterations, the mesh is often too coarse to allow an accurate solution to the optimal control problem, so tight optimization tolerances waste computational time.

The following example for GMAT and C++ starts with a tolerance of 1e-4 for the first mesh refinement iteration and lowers the tolerance by an order of magnitude per iteration until 1e-7. Then, for all iterations where $i > 2$ (zero-indexed), the feasibility tolerance is 1e-7.

GMAT Script Example

```
traj.FeasibilityTolerances = [1e-4, 1e-5, 1e-6, 1e-7]
```

C++ Example

```
// Function Prototype:
// virtual void SetFeasibilityTolerances(const Rvector &tol);

// Example:
Rvector feasTol(4, 1e-4, 1e-5, 1e-6, 1e-7);
myTraj.SetFeasiblityTolerances(feasTol);
```

MATLAB Example

```
% Note the MATLAB interface for setting tolerances is different than
% GMAT interface and CSALT C++ interface.
% Tolerances are set in Optimizer.m
snset('Major feasibility tolerance 5e-7')
```

The example below performs the first two mesh iterations with a major optimality tolerance of 1e-2 for GMAT and C++. Then, all subsequent mesh iterations use a tolerance of 1e-5.

GMAT Script Example

```
traj.MajorOptimalityTolerances = [1e-2, 1e-2, 1e-5]
```

C++ Example

```
// Function Prototype:
// virtual void SetOptimalityTolerances(const Rvector &tol);

// Example:
Rvector optTol(3, 1e-2, 1e-2, 1e-5);
myTraj.SetOptimalityTolerances(optTol)
```

MATLAB Example

```
% Note the MATLAB interface for setting tolerances is different than
% GMAT interface and CSALT C++ interface.
% Tolerances are set in Optimizer.m
snset('Major feasibility tolerance 5e-7')
```

4.4.4 Setting Optimization Iterations Limits

CSALT supports several optimization iterations limits, including the total optimization iterations, major optimization iterations, and mesh refinement iterations. Examples are shown below for each setting.

The example below configures CSALT for GMAT and C++ to only allow 50, 100, 500, and 1000 optimizer major iterations for the first four mesh refinement iterations, respectively. The final value in the vector passed to SetMajorIterationsLimit (in this case, 1000) is then used for all subsequent mesh iterations.

GMAT Script Example

```
traj.MajorIterationsLimits = [50, 100, 500, 1000]
```

C++ Example

```
// Function Prototype:
// virtual void SetMajorIterationsLimit(const IntegerArray &iter);

// Example:
Rvector iterLimit(4, 50, 100, 500, 1000);
traj.SetMajorIterationsLimit(iterLimit);
```

MATLAB Example

```
% Note the MATLAB interface for setting tolerances is different than
% GMAT interface and CSALT C++ interface.
% Tolerances are set in Optimizer.m
snseti('Major iterations limit',75);
```

The example below illustrates how to set the total iterations limit to be 1000 and 2000 for the first two iterations, respectively, and to 10000 for all subsequent iterations.

GMAT Script Example

```
traj.TotalIterationsLimits = [1000, 2000, 10000]
```

C++ Example

```
// Function Prototype:
// virtual void SetTotalIterationsLimit(const IntegerArray &iter);

// Example:
Rvector iterLimit(3, 1000, 2000, 10000);
traj.SetTotalIterationsLimit(iterLimit);
```

MATLAB Example

```
% Note the MATLAB interface for setting tolerances is different than
% GMAT interface and CSALT C++ interface.
% Tolerances are set in Optimizer.m
snseti('Iterations limit',3000);
```

4.4.5 Setting Mesh Refinement Iterations

The example below shows how to set the maximum number of mesh refinement iterations to 7.

GMAT Script Example

```
traj.MaxMeshRefinementIterations = 7
```

C++ Example

```
// Function Prototype:
// virtual void SetMaxMeshRefinementCount(Integer toCount);

// Example:
Integer iterLimit = 7;
traj.SetMaxMeshRefinementCount(iterLimit);
```

MATLAB Example

```
traj.maxMeshRefinementCount = 7;
```

4.4.6 Setting Optimization and Mesh Modes

CSALT supports several settings that determine optimization modes and behavior. These settings allow the user to tell CSALT to find a feasible or optimal solution, whether to allow mesh refinement to continue if an optimization fails, and to choose between guess options to use for new mesh refinement iterations. Supported optimization modes are “Minimize”, “Feasible point”, and “Maximize”. The examples below illustrate how to configure CSALT to find a feasible solution for the first two mesh iterations, and to find a minimum point for all subsequent mesh iterations.

GMAT Script Example

```
traj.OptimizationMode = {Feasible point, Feasible point, Minimize}
```

C++ Example

```
// Function Prototype:
// virtual void SetOptimizationMode(const StringArray &optMode);

// Example:
optMode = StringArray(3, "Feasible point", "Feasible point", "Minimize");
SetOptimizationMode(optMode);
```

The user can optionally allow mesh iterations to continue in the event an optimization fails to converge. Often, if the optimizer fails but is "reasonably close" to a solution, subsequent mesh iterations will converge. The example below shows how to configure CSALT to continue mesh refinement iterations even if an optimization (i.e., mesh refinement iteration) fails to converge.

GMAT Script Example

```
traj.AllowFailedMeshOptimizations = true
```

C++ Example

```
// Function Prototype:
// virtual void SetFailedMeshOptimizationAllowance(bool toAllowance);

// Example:
SetFailedMeshOptimizationAllowance(true);
```

MATLAB Example

Note: AllowFailedMeshOptimizations is not supported in CSALT MATLAB.

When performing mesh iterations after a failed mesh refinement iteration (due to reaching the maximum number of iterations or encountering numerical issues in the previous mesh refinement iteration), it is often advantageous to use the "best" previous solution as the initial guess for the next mesh refinement iteration rather than the unconverged final decision variable values from the most recent mesh refinement iteration. This behavior is especially useful when an optimization run is performing reasonably well and then a mesh refinement iteration unexpectedly fails to converge.

CSALT allows the user to specify the previous solution used to generate the initial guess for the next mesh refinement iteration. Supported modes are "LastSolutionMostRecentMesh", "BestSolutionMostRecentMesh" and "BestSolutionAnyMesh". "Last" refers to the last solution obtained during a mesh refinement iteration. The following conditions are used to define "best": (1) If no feasible solutions have been found, then "best" refers to the previous solution with the smallest infeasibility. (2) If one feasible solution has been found, then that solution is "best," regardless of its optimality. (3) If multiple feasible solutions have been found, then the feasible solution with the smallest value of the merit function (i.e., the most optimal solution) is "best." "MostRecentMesh" means a guess for the new mesh will only be taken from the previous mesh refinement iteration, while "AnyMesh" means the guess can come from any previous mesh refinement iteration.

The example below shows how to configure CSALT to use either the most recent mesh refinement iteration or any mesh refinement iteration to obtain the initial guess for a given subsequent mesh refinement iteration.

GMAT Script Example

```
traj.MeshRefinementGuessMode = 'BestSolutionAnyMesh'
```

C++ Example

```
// Function Prototype:
// virtual void SetMeshRefinementGuessMode(const std::string &toGuessMode);

// Example:
SetMeshRefinementGuessMode("BestSolutionAnyMesh");
```

4.4.7 Setting SNOPT Output Data File

SNOPT writes data to an output file during optimization. The examples below show how to set the filename for the SNOPT output file.

GMAT Script Example

```
traj.SNOPTOutputFile = fileName
```

C++ Example

The Trajectory object's Optimize method takes as an argument a string for the name of the optimizer output file. This string is passed to the Trajectory's SnoptOptimizer object.

```
// function prototype for setting the SNOPT output file
// void SnoptOptimizer::SetOptimizerOutputFile(const std::string &optFile)

// function prototype for Trajectory::Optimize (Note that Optimize has multiple␣
↪overloaded definitions.)
// void Trajectory::Optimize(const std::string &optFile)

// Example:
Optimize("/home/username/file.txt")
```

4.4.8 Setting SNOPT Print-To-Console Verbosity

By default, SNOPT writes output to the console during execution. In some circumstances, this behavior may be undesirable. The example below shows how to quiet SNOPT output. Specifically, the following SNOPT options are set:

- Major Print Level = 0
- Minor Print Level = 0
- Print No = 0
- Summary file = 0

- Suppress parameters

By default, the default values are used for all of the above settings.

> **Warning:** This option is only available in CSALT C++.

C++ Example

```
Trajectory* traj; // create pointer to Trajectory object
traj->SetSnoptConsoleOutputLevel(0); // 0: quiet output; all other integers (and␣
↪default): default output
```

4.4.9 Scaling the Optimal Control Problem

GMAT currently supports one option for scaling optimal control problems using canonical units, which are defined as follows: If the Sun is the central body, then the distance unit is an astronomical unit and the GM of the Sun is 1. If the Earth is the central body, then the distance unit is 42000 km and the GM of the Earth is 1. Otherwise, the radius of the central body is the distance unit and the GM of the central body is 1.

Mass values can change significantly from one application to the next and the user can set the mass scale factor as shown below. It is a best practice to choose a mass scale factor such that the total mass is approximately 1.0 at the beginning of the simulation period.

GMAT Script Example

```
traj.StateScaleMode = Canonical
traj.MassScaleFactor = 1000 % in Kg.
```

> **Warning:** CSALT CURRENTLY REQUIRES USERS TO PROVIDE DATA IN NON-DIMENSIONAL UNITS. BUILT-IN SCALING IS IN PROGRESS BUT NOT COMPLETE.

4.4.10 Setting the Solution Output Configuration in GMAT

GMAT can optionally write solution data to an optimal control history (.och) file. The user may select the output coordinate system by providing the name of a configured coordinate system. Alternatively, the user can use the keyword "UsePhaseCoordinateSystems" and the solution for each Phase will be reported in the Phase's coordinate system.

GMAT Script Example

```
traj.SolutionFile = PATH_AND_FILENAME.EXTENSTION
traj.OutputCoordinateSystem = UsePhaseCoordinateSystems

% Or to write the data in EarthMJ2000Eq, for example

traj.OutputCoordinateSystem = EarthMJ2000Eq
```

4.4.11 Setting Simple Linkage Constraints in GMAT

A simple linkage constraint requires that the full state (and time) of a phase at one endpoint be equal to the full state (and time) of another phase at one endpoint. In order for a simple linkage constraint to be enforced, a list of linked phases must be added to a trajectory.

GMAT Script Example

```
traj.AddSimpleLinkageChain = {phase1, phase2, phase3}
```

4.4.12 Setting the Guess Source in GMAT

GMAT requires an initial guess for the trajectory states and controls. The guess source may be set at the Trajectory level and/or at the Phase level; if a guess resource is set for a Phase, then that guess source overrides the Trajectory-level guess source for that Phase.

```
Create OptimalControlGuess trajectoryGuess
trajectoryGuess.Type = CollocationGuessFile
trajectoryGuess.FileName = ../data/misc/GuessWithUnityControl.och

traj.GuessSource = trajectoryGuess
```

4.4.13 Setting the Publisher Update Rate in GMAT

GMAT publishes state data during optimization to subscribers such as plots, graphic windows, reports, and ephemeris files. You can control how much data is sent to subscribers by setting the PublishUpdateRate field. For example, if PublishUpdateRate is set to 10, then data is only sent to subscribers every 10 cost/constraint function evaluations.

```
traj.PublishUpdateRate = 10
```

4.4.14 Setting Path Functions in CSALT

The path functions include dynamics models, algebraic path functions, and integral cost functions. The examples below show how to include Path Function objects in an optimization problem. Configuring those objects is discussed in *Path Functions Reference*.

GMAT Script Example

Note: Path functions are NOT accessed directly by users in GMAT scripts.

C++ Example

```
// Here, BrachistichronePathObject is derived from UserPathObject

// Function Prototype:
// virtual void SetUserPathFunction(UserPathFunction *func);

// Example:
Traj = new Trajectory();
pathObject = new BrachistichronePathObject();
traj->SetUserPathFunction(pathObject);
```

MATLAB Example

```
% Here MyPathObject is a class derived from UserPathFunction.
% This creates an object of type MyPathObject.
thePathObject = MyPathObject();

% Now add the object to the trajectory
traj.pathFunctionObject = thePathObject;
```

4.4.15 Setting Boundary Functions in CSALT

Boundary functions include algebraic boundary constraints and algebraic cost function terms that depend on only static variables and parameters at phase boundaries. Boundary Function objects are created and then added to a Trajectory object. If a Boundary Function is not added to a Trajectory object, then the boundary function will not be applied during optimization.

C++ Example

CSALT supports two interfaces for providing Boundary Functions to a Trajectory: the UserPointFunction class and the OptimalControlFunction utility class. Those classes are described in detail in *Boundary Functions Reference*. The examples below show how to include Boundary Functions in an optimization by adding them to the Trajectory.

A Trajectory can only have one UserPointFunction object. To include a UserPointFunction object:

```
// Here, BrachistichronePointObject is derived from UserPointObject

// Function Prototype:
// virtual void SetUserPointFunction(UserPointFunction *func);

// Example:
Traj = new Trajectory();
pointObject = new BrachistichronePointObject();
traj->SetUserPathFunction(pointObject);
```

C++ Example

A Trajectory can have a list of OptimalControlFunction objects. Optimal control functions allow re-use of models, cost, and constraint functions between different problems and applications. The example below contains the function prototype to add an OptimalControlFunction object to a Trajectory.

```
// Function Prototpye:
void UserPointFunction::AddFunctions(
    std::vector<OptimalControlFunction*> funcList)
```

MATLAB Example

Like in C++, a MATLAB Trajectory object can only have one UserPointFunction object.

```
% Here MyBoundaryObject is a class derived from UserPointFunction.
% This creates an object of type MyBoundaryObject.
theBoundaryObject = MyBoundaryObject();

% Now add the object to the trajectory
traj.pointFunctionObject = theBoundaryObject;
```

4.4.16 Setting the ExecutionInterface Object in CSALT

The ExecutionInterface object is an optional referenced object that CSALT calls during optimization to provide CSALT state data during execution. The ExecutionInterface is described in detail in *ExecutionInterface Reference*. In the example below, we assume the ExecutionInterface object has been created and configured and illustrate how to provide the ExecutionInterface to the Trajectory object.

GMAT Script Example

Note: This interface is not applicable to GMAT script users. Publishing in GMAT is handled via the GMAT publisher using GMAT publish/subscribe components that are transparent to GMAT users.

C++ Example

```
// Function Prototpye:
// void GmatTrajectory::SetExecutionInterface(CsaltExecutionInterface *intf)

// Example:
csaltInterface = new CsaltExecutionInterface(this);
execInterface = new Publisher();
traj = new Trajectory();
Trajectory::SetExecutionInterface(execInterface)
```

4.5 Phase Reference

A Phase is a segment of a Trajectory (a Trajectory must have a least one Phase). Phases are often used to differentiate portions of a trajectory that require different dynamics models or to break a problem down into segments to reduce the non-linearity of the problem and therefore improve sensitivity to poor guesses.

In GMAT, the Phase Resource is where the transcription and dynamics model are configured (different phases can use different transcriptions and dynamics models). Other settings configured on the phase object include bounds on state, control, time, and static parameters, and the initial guess for the phase. Examples for each of these settings are shown in the subsections below.

Note that GMAT supports pre-defined dynamics models while the C++ and MATLAB systems are low level APIs where you must code your own dynamics models. There are also differences in how guesses are provided in GMAT and CSALT. Below we present an overview of state and control representations, how to select and configure a transcription, and then provide code examples that illustrate how to configure a Phase in GMAT, CSALT C++ and CSALT MATLAB.

4.5.1 State and Control Representations in GMAT

The default state representation in GMAT is the Cartesian state and total mass (7 components) and the default control representation is "up-to-unit-vector" control (three components). The coordinate system for a phase uses MJ2000Eq axes and the central body of the Force Model selected in the DynamicsConfiguration for the phase. State bounds are defined with respect to that coordinate system.

The default control representation is "up-to-unit-vector" control. When using up-to-unit vector control, the control decision vector at a given discretization point has three elements that define the direction and magnitude of the thrust. The applied thrust in the equations of motion is equal to the maximum available thrust from the propulsion system, multiplied by the control decision vector components. If the control decision vector has magnitude 1.0, then the applied thrust is the maximum thrust available from the system. If the control decision vector has magnitude 0.0, then no thrust is applied.

Warning: GMAT supports setting the maximum control magnitude, MaxControlMagnitude, to be greater than 1.0. This allows the use of homotopy when starting from a poor guess or for troubleshooting issues in a problem's configuration. It is critical to understand that solutions found when MaxControlMagnitude > 1.0 are non-physical solutions that use MORE thrust than the chosen thrust system can produce.

Note: It is a best practice to set the lower bounds on mass to be positive and near, but not equal, to zero. If mass is zero, then a singularity in the dynamics occurs because mass appears in the denominator for the thrust acceleration term in the equations of motion.

4.5.2 Setting the Transcription

Configuring a transcription requires several steps including:

- Selecting a transcription algorithm
- Setting the discretization properties (how many points/steps, and the relative location of the points/steps)
- Setting the tolerance on the discretization accuracy

In this section, we discuss details of those setting and then show examples in GMAT, CSALT C++, and CSALT MATLAB.

CSALT supports several transcription types including Radau pseudospectral and implicit Runge Kutta. The system supports several implicit Runge-Kutta transcriptions including Hermite Simpson and several Lobatto IIIA types among others. To select a transcription algorithm, use the Type field on the Phase Resource.

The discretization configuration defines the number and relative location of discretization points within a Phase. In GMAT, the discretization points/steps are defined using the fields SubPhaseBoundaries and PointsPerSubPhase fields. In CSALT, the discretization points/steps are defined using the functions SetMeshIntervalFractions() and SetMeshIntervalNumPoints(). (SubPhaseBoundaries corresponds to SetMeshIntervalFractions() and PointsPerSubPhase corresponds to SetMeshIntervalNumPoints().) In brief, SubPhaseBoundaries defines the number of sub-phases within a phase and the span of the independent variable of each sub-phase in the initial mesh, while PointsPerSubPhase sets the number of collocation points within each sub-phase in the initial mesh. See the discussions in Table 5.25 and Table 5.26 for a more detailed description of SubPhaseBoundaries and PointsPerSubPhase, respectively.

Different transcription types require different discretization configurations. Radau phases are non-dimensionalized on the independent variable (e.g., time) from -1.0 to 1.0. SubPhaseBoundaries (in GMAT) and MeshIntervalFractions (in CSALT) are arrays of real numbers that, for Radau, must start with -1.0 and end with 1.0, with intermediate values (increasing monotonically) representing the relative spacing of the subphases within a phase. On the other hand, Runge-Kutta phases are non-dimensionalized on the independent variable from 0.0 to 1.0.

When dynamics are more rapid, more collocation points are required for accurate modeling. For example, if it is known that there is a relatively highly dynamic region in the first half of a phase, and relatively slow dynamic region in the second half of the phase, it is natural to place more mesh points in the first half of the phase. Mesh refinement refines the initial mesh, potentially increasing the number of subphases and/or the number of points in individual subphases to model the dynamics with enough accuracy to meet the user's mesh refinement tolerance.

Defining discretization points in GMAT or CSALT defines a guess for the discretization and the mesh refinement algorithm will refine the discretization to meet the requested accuracy. The MaxRelativeErrorTolerance field defined the maximum relative error allowed in a phase.

GMAT Script Example

The following example configures a Phase to use RadauPseudospectral transcription, with 5 subphases each with a third order polynomial.

```
Create Phase thePhase
thePhase.Type = RadauPseudospectral
thePhase.SubPhaseBoundaries = [-1.0,-0.75,-0.5,-0.25,0.0,1.0]
thePhase.PointsPerSubPhase = [4 4 4 4 4]
thephase.MaxRelativeErrorTolerance = 1e-5
```

C++ Example

In the example below, we place 4 equally spaced mesh intervals in the first half of the phase, and one mesh interval in the second half of the phase where all mesh intervals have 4 points.

```
// Create a phase pointer to a Radau phase.
Phase *phase1;
phase1 = new RadauPhase();

// Set the mesh interval fractions and number of points in each mesh interval. Here we␣
↪have one mesh interval with 5 points in it.
Rvector meshIntervalFractions(6,-1.0,-0.75,-0.5,-0.25,0.0,1.0);
IntegerArray meshIntervalNumPoints; // an std::vector of Integer variables
meshIntervalNumPoints.push_back(5);
```

(continues on next page)

(continued from previous page)

```
meshIntervalNumPoints.push_back(4);
meshIntervalNumPoints.push_back(4);
meshIntervalNumPoints.push_back(4);
meshIntervalNumPoints.push_back(4);
meshIntervalNumPoints.push_back(4);
phase1->SetMeshIntervalFractions(meshIntervalFractions);
phase1->SetMeshIntervalNumPoints(meshIntervalNumPoints);

// Set the mesh refinement relative error tolerance to 1e-5
phase1->SetRelativeErrorTol(1e-5);
```

MATLAB Example

```
phase1 = RadauPhase;
phase1.SetMeshIntervalFractions([-1.0,-0.75,-0.5,-0.25,0.0,1.0]);
phase1.SetMeshIntervalNumPoints([4 4 4 4 4]);
```

The example below illustrates how to configure a phase to use an ImplicitRungeKutta phase. There are several Implicit Runge-Kutta transcription types supported as shown below.

```
// Create an implicit Runge Kutta phase of order 8
ImplicitRKPhase *phase1 = new ImplicitRKPhase();
phase1->SetTranscription("RungeKutta8");

// Set it to have one mesh interval with 20 steps
Rvector meshIntervalFractions(2, 0.0, 1.0);
IntegerArray meshIntervalNumPoints;
meshIntervalNumPoints.push_back(20);
phase1->SetMeshIntervalFractions(meshIntervalFractions);
phase1->SetMeshIntervalNumPoints(meshIntervalNumPoints);
```

4.5.3 Setting Decision Vector Bounds and Dimensions

Examples below show how to set bounds on state, control, independent variable (e.g., time), and static variables. There are some differences in interfaces between GMAT and CSALT related to time, and setting parameter dimensions such as the number of states and controls. In CSALT, the user must explicitly set the number of states, controls, etc. In GMAT, the number of states and the number of controls are not currently user-settable: the number of states is seven (position vector, velocity vector, mass), and the number of controls is three (Cartesian control vector).

GMAT Script Example

```
% Create a phase
Create Phase thePhase
thePhase.Type = RadauPseudospectral

% Set lower bounds on Cartesian position (default state type) to
% -10000 km, bounds on Cartesian velocity to -10 km/s and mass
% to 0.01 kg.
```

(continues on next page)

(continued from previous page)

```
thePhase.StateLowerBound = [ -10000 -10000 -10000 -10 -10 -10 0.01]

% Set upper bounds on Cartesian position (default state type) to
% 10000 km, bounds on Cartesian velocity to 10 km/s and mass
% to 3000 kg.
thePhase.StateUpperBound = [  10000  10000  10000  10  10  10 3000]

% Set bounds on the phase epochs.
thePhase.EpochFormat = TAIModJulian
thePhase.EpochLowerBound = 32402
thePhase.EpochUpperBound = 32405

% Set the bounds on control to -2 and 2 respectively.
thePhase.ControlLowerBound = [-2 -2 -2]
thePhase.ControlUpperBound = [2 2 2]

% Set the bounds on the control vector magnitude.
% WARNING. SETTING MaxControlMagnitude > 1.0 SHOULD BE USED WITH CARE
% (FOR HOMOTOPY OR TROUBLESHOOTING) AND WILL RESULT IN NON-PHYSICAL SOLUTIONS.
thePhase.MinControlMagnitude = 0.0
thephase.MaxControlMagnitude = 1.0
```

C++ Example

```
// Create a phase
Phase *phase1;
phase1 = new RadauPhase();

// Configure problem with two state variables with lower and upper
// bounds of 0.0 and 2.0, respectively, for both state variables.
Integer numStateVars = 2;
Rvector stateLowerBound(2, 0.0, 0.0);
Rvector stateUpperBound(2, 2.0, 2.0);
phase1->SetNumStateVars(numStateVars);
phase1->SetStateLowerBound(stateLowerBound);
phase1->SetStateUpperBound(stateUpperBound);

// Configure problem with one control variable, with lower and upper
// bounds of -10.0 and 10.0 respectively.
Integer numControlVars = 1;
Rvector controlUpperBound(1,10.0);
Rvector controlLowerBound(1,-10.0);
phase1->SetNumControlVars(numControlVars);
phase1->SetControlLowerBound(controlLowerBound);
phase1->SetControlUpperBound(controlUpperBound);

// Configure problem with three static variables with lower and upper
// bounds to 0.0 and 20.0, respectively, for all three variables.
Integer numStaticVars = 3;
Rvector staticUpperBound(3, 20.0, 20.0, 20.0);
```

(continues on next page)

(continued from previous page)

```
Rvector staticLowerBound(3, 0.0, 0.0, 0.0);
phase1->SetNumStaticVars(numStaticVars);
phase1->SetStaticLowerBound(staticLowerBound);
phase1->SetStaticUpperBound(staticUpperBound);

// Configure lower and upper bounds on time (or independent variable)
// to 0.0 and 100.0 respectively.
Real timeLowerBound = 0.0;
Real timeUpperBound = 100.0;
phase1->SetTimeLowerBound(timeLowerBound);
phase1->SetTimeUpperBound(timeUpperBound);
```

MATLAB Example

```
% Configure problem with three state variables with lower and upper
% bounds of -10 and 10, respectively, for all state variables.
phase1.SetNumStateVars(3)
phase1.SetStateLowerBound([-10 -10 -10]')
phase1.SetStateUpperBound([10 0 0]')

% Configure problem with one control variable with lower and upper
% bounds of -10 and 10, respectively.
phase1.SetNumControlVars(1);
phase1.SetControlLowerBound([-10]')
phase1.SetControlUpperBound([10]')

% Configure lower and upper bounds on time (or independent variable)
% to 0.0 and 100.0 respectively.
phase1.SetTimeLowerBound(0)
phase1.SetTimeUpperBound(100)
```

4.5.4 Setting the Initial Guess in GMAT

In GMAT, state and control guesses are defined by an OptimalControlGuess Resource and Phase time guesses are defined via fields on the Phase. An OptimalControlGuess Resource supports text-based and array-based guess data to easily create randomized guesses, import guesses from GMAT utility scripts, call MATLAB or Python for guess data, or use data from an external tool such as EMTG, ATD, or MALTO among others. See the OptimalControlGuess documentation for further information on how to define the values of state and control guesses.

You can provide an OptimalControlGuess on a Phase and/or a Trajectory. If the Phase is provided with an OptimalControlGuess, that data is used as the guess for a phase, otherwise the guess on the Trajectory is used. If there is no guess on a Phase or Trajectory, an error is issued. The script example below shows how to define the OptimalControlGuess Resource (state and control) for a Phase, and the initial guess for phase epochs.

GMAT Script Example

```
% Use a pre-configured OptimalControlGuess Resource
% called "thePhaseGuess" as the guess for state and control
phase.GuessSource = thePhaseGuess

% Define the initial and final epochs using TAI modified Julian date
phase.EpochFormat = TAIModJulian
phase.InitialEpoch = 34050.99962787928
phase.FinalEpoch = 34513.39999998195

% Alternatively, define the initial and final epochs using the
% UTC Gregorian date
phase.EpochFormat = UTCGregorian
phase.InitialEpoch = 29 Mar 2034 11:58:52.849
phase.FinalEpoch = 04 Jul 2035 21:35:24.998
```

4.5.5 Setting the Initial Guess in CSALT C++ and MATLAB

CSALT supports several interfaces for setting guesses including a linear model for state and control, array based and file based guesses. The sections below contain examples for setting guesses for time (or independent variable), state, control, and static parameters using the various interfaces supported for initial guesses.

Setting the Initial Guess for Time (or Independent Variable)

C++ Example

```
// Set the initial and final time guess to 0.0 and 0.3 respectively.
Real initialGuessTime = 0.0;
Real finalGuessTime = 0.3;
phase1->SetTimeInitialGuess(initialGuessTime);
phase1->SetTimeFinalGuess(finalGuessTime);
```

MATLAB Example

```
%Set the initial and final time guess to 0.0 and 0.3 respectively.
phase1.SetTimeInitialGuess(0.0)
phase1.SetTimeFinalGuess(0.3)
```

Setting the Initial Guess for State and Control using Linear Guess Model

CSALT supports several methods for defining the initial guess for state and control, including several array-based approaches and a file-based approach.

The simplest guess modes are called "LinearUnityControl" and "LinearZeroControl". These two modes construct a straight-line guess (in the coordinates of the problem) and set control to all ones in the case of "LinearUnityControl" and all zeros in the case of "LinearZeroControl". State guesses are linearly interpolated to the time/independent variable values in the discretization.

C++ Example

```
// Set the state initial guess to vary linearly from 0.0 to 1.0 for
// all three state variables.  Control decision variable are
// all set to 1.0.
std::string initialGuessMode = "LinearUnityControl";
Rvector initialGuessState(3, 0.0, 0.0, 0.0);
Rvector finalGuessState(3, 1.0, 1.0, 1.0);
phase1->SetInitialGuessMode(initialGuessMode);
phase1->SetStateInitialGuess(initialGuessState);
phase1->SetStateFinalGuess(finalGuessState);
```

MATLAB Example

```
% Set the state initial guess to vary linearly from 0.0 to 1.0 for
% all three state variables.  Control decision variable are
% all set to 0.0.
phase1.initialGuessMode = 'LinearNoControl';
phase1.SetStateInitialGuess([0 0 0]')
phase1.SetStateFinalGuess([1.0 1.0 1.0]')
```

Setting State and Control Guesses in CSALT C++ using Arrays of Data

The CSALT C++ interface allows you to provide arrays of time, state, and control, and CSALT interpolates the guess data to the discretization times of the phase. The example below shows how to provide guess data arrays for state and control.

C++ Example

```
Rvector timeArray(11,
    0,
    0.021836090339203817,
    0.065059858061501996,
    0.11298609481175435,
    0.14731810202287049,
    0.15624006535589879,
    0.1780761556951026,
    0.22129992341740082,
    0.26922616016765311,
    0.30355816737876928,
    0.31248013071179759);

Rmatrix stateArray;
stateArray.SetSize(11, 3);
stateArray(0, 0) = 0;
stateArray(0, 1) = 0;
stateArray(0, 2) = 0;
stateArray(1, 0) = 0.000558;
stateArray(1, 1) = -0.0076368;
stateArray(1, 2) = -0.70114;
stateArray(2, 0) = 0.014536;
stateArray(2, 1) = -0.065705;
stateArray(2, 2) = -2.0561;
stateArray(3, 0) = 0.072888;
stateArray(3, 1) = -0.1842;
stateArray(3, 2) = -3.4429;
stateArray(4, 0) = 0.15442;
stateArray(4, 1) = -0.2898;
stateArray(4, 2) = -4.3183;
stateArray(5, 0) = 0.18169;
stateArray(5, 1) = -0.31831;
stateArray(5, 2) = -4.5258;
stateArray(6, 0) = 0.25921;
stateArray(6, 1) = -0.38763;
stateArray(6, 2) = -4.9943;
stateArray(7, 0) = 0.4556;
stateArray(7, 1) = -0.51198;
stateArray(7, 2) = -5.7398;
stateArray(8, 0) = 0.72747;
stateArray(8, 1) = -0.607;
stateArray(8, 2) = -6.2497;
stateArray(9, 0) = 0.94294;
stateArray(9, 1) = -0.63533;
stateArray(9, 2) = -6.394;
stateArray(10, 0) = 1;
stateArray(10, 1) = -0.63662;
stateArray(10, 2) = -6.4004;

Rmatrix controlArray;
```

(continues on next page)

(continued from previous page)

```
controlArray.SetSize(11, 1);
controlArray(0, 0) = 0;
controlArray(1, 0) = -0.10977;
controlArray(2, 0) = -0.32704;
controlArray(3, 0) = -0.56797;
controlArray(4, 0) = -0.74055;
controlArray(5, 0) = -0.78541;
controlArray(6, 0) = -0.89516;
controlArray(7, 0) = -1.1125;
controlArray(8, 0) = -1.3534;
controlArray(9, 0) = -1.5259;
controlArray(10, 0) = -1.5705;

std::string initialGuessMode = "GuessArrays";
phase1->SetInitialGuessMode(initialGuessMode);
phase1->SetInitialGuessArrays(timeArray, stateArray, controlArray);
```

Setting State and Control Guesses in CSALT C++ using an OCH Guess File

The CSALT C++ version supports file-based guesses for state and control using an Optimal Control History (OCH) file. The OCH file specification is located in the OptimalControlGuess documentation. The example below shows how to configure CSALT C++ to use an OCH file for the source of data for state and control guesses.

C++ Example

```
// Create a phase and set the guess mode to OCHFile
RadauPhase *phase1 = new RadauPhase();
std::string initialGuessMode = "OCHFile";
phase1->SetInitialGuessMode(initialGuessMode);

// Define the path to the guess file
std::string initialGuessFile = "PATH_TO_GUESS_FILE/GUESS_FILE_NAME.och";
phase1->SetGuessFileName(initialGuessFile);
```

Setting State and Control Guesses in CSALT MATLAB using a MATLAB Function

The CSALT MATLAB interface supports a function-based guess source. You can define a MATLAB function that provides state and control guess data. The example below shows how to configure a function-based guess in CSALT MATLAB, and the function prototype for MATLAB guess functions.

MATLAB Example

```
% Set the guess mode to UserGuessFunction for a Phase named thePhase
thePhase.initialGuessMode = 'UserGuessFunction';

% Set the name of the guess function on the Trajectory
theTrajectory.guessFunctionName = 'MyGuessFunction';
```

A MATLAB guess function must return arrays of state and control guess data interpolated to the phase discretization points. Inputs to a guess function are:

- A vector of discretization times/independent variable for phase.
- An array of integers describing the discretization point types. 1 means a discretization point needs state guess only, 2 means a discretization point needs state and control guess data.
- A handle to the phase Object for obtaining phase information such as number of states and controls for the phase.

Outputs from a MATLAB guess function are:

- State guess array with guesses for state as rows in the array.
- Control guess array with guesses for control as rows in the array.

MATLAB Function Prototype for Guess Functions

```
function [stateGuess,controlGuess] = MyGuessFunc(timeVector,timeVectorType,phaseObj)
```

Setting the Initial Guess for Static Variables

Static variables are defined on a per-phase basis and a phase need not have static variables. The example below shows how to set the guess for static variables.

C++ Example

```
// Configure guess for three static variables set to 1.0, 2.0, and
// 3.0 respectively.
Integer numStaticVars = 3;
phase1->SetNumStaticVars(numStaticVars);
Rvector staticGuess(numStaticVars, 1.0, 2.0, 3.0);
phase1->SetStaticGuess(staticGuess);
```

MATLAB Example

Note: The MATLAB version of CSALT does not support static variables.

4.5.6 Setting the Dynamics Configuration in GMAT

Phases in GMAT use a DynamicsConfiguration Resource to define the Spacecraft, Force Model, and propulsion model for the phase. To configure the phase dynamics you define the DynamicsConfiguration for a Phase, and the ThrustMode. The example below shows how to configure that data first by discussing the fields on the Phase Resource, and then on how to set up a DynamicsConfiguration Resource including the Spacecraft, Force Model, and propulsion model.

```
% Create a phase.  Set it to
% use the DeepSpaceDynConfig DynamicsConfiguration.
Create Phase thePhase
thePhase.DynamicsConfiguration = DeepSpaceDynConfig

% Set the Phase to a Coast.  (thrust is not applied)
thePhase.ThrustMode = Coast

% Alternatively, set the Phase to a Thrust.  (thrust is applied)
thePhase.ThrustMode = Thrust
```

The example above assumes a DynamicsConfiguration named DeepSpaceDynConfig is configured. Below we illustrate how to set up a that Resource.

```
% Create a chemical tank
Create ChemicalTank ChemicalTank1;
ChemicalTank1.AllowNegativeFuelMass = false;
ChemicalTank1.FuelMass = 4150;
ChemicalTank1.Pressure = 1500;
ChemicalTank1.Temperature = 20;
ChemicalTank1.RefTemperature = 20;
ChemicalTank1.Volume = 3.5;
ChemicalTank1.FuelDensity = 1260;
ChemicalTank1.PressureModel = PressureRegulated;

% Create a Spacecraft
Create Spacecraft mySat
mySat.Tanks = {ChemicalTank1};

% Create an orbit dynamics model with Earth, Sun, Moon point mass and SRP
Create ForceModel DeepSpaceForceModel;
GMAT DeepSpaceForceModel.CentralBody = Sun;
GMAT DeepSpaceForceModel.PointMasses = {Sun,Earth,Luna};
GMAT DeepSpaceForceModel.Drag = None;
GMAT DeepSpaceForceModel.SRP = On;

% Create a spacecraft thrust model
Create EMTGSpacecraft ThrustModel;
ThrustModel.SpacecraftFile = FixedThrustAndIsp_Model1.emtg_spacecraftopt
```

(continues on next page)

(continued from previous page)

```
ThrustModel.DutyCycle = 1.0;

% Create a DynamicsConfiguration and configure the Spacecraft,
% ForceModel, and Thrust model.
Create DynamicsConfiguration DeepSpaceDynConfig
DeepSpaceDynConfig.ForceModels = {DeepSpaceForceModel};
DeepSpaceDynConfig.Spacecraft = {mySat}
DeepSpaceDynConfig.FiniteBurns = {ThrustModel}
DeepSpaceDynConfig.EMTGTankConfig = {ChemicalTank1};
```

4.5.7 Setting Phase Graphics Configuration in GMAT

The Phase Resource supports options that can optionally override a spacecraft's default graphics configuration so that different phases are displayed using different colors in graphics windows, even if the phases use the same spacecraft. There are two fields, one to optionally override the spacecraft's color settings, and the other to specify the desired color for the phase (which is not used if OverrideColorInGraphics = false) . The example below shows how to set the Phase to be drawn using green in the graphics windows.

```
thePhase.OverrideColorInGraphics = true
thePhase.OrbitColor = 'Green'
```

4.5.8 Setting Phase Cost and Constraint Functions in GMAT

The Phase Resource has fields to set cost and constraint functions implemented in GMAT. These fields are documented in the BoundaryFunctions section to keep documentation on setting cost and constraints in a single location. The fields to set "built-in" cost and constraint functions are called BuiltInCost and BuiltInBoundaryConstraints. See the Boundary Functions Chapter for more details on supported built-in cost and constraint functions.

4.6 Dynamics Configuration Reference

The DynamicsConfiguration resource is used to set the natural and artificial forces acting on spacecraft. Each Phase of a CSALT Trajectory has one DynamicsConfiguration. In a GMAT script, the dynamics configuration for a phase is created and set using:

```
Create DynamicsConfiguration ExampleDynamicsConfiguration
<Set up ExampleDynamicsConfiguration>
Create Phase ExamplePhase
ExamplePhase.DynamicsConfiguration = ExampleDynamicsConfiguration
```

4.6.1 Setting the Force Model

A DynamicsConfiguration resource requires a list of ForceModel resources, which describe the non-control forces acting on the spacecraft. **Important note: Currently, CSALT only supports a single spacecraft, so the length of the list must be one.** In a GMAT script, the ForceModel resource is set using the following script block.

```
Create ForceModel ExampleForceModel1
<Set up ExampleForceModel1>

Create DynamicsConfiguration ExampleDynamicsConfiguration
ExampleDynamicsConfiguration.ForceModels = {ExampleForceModel1}
```

4.6.2 Setting the Spacecraft

A DynamicsConfiguration resource requires a list of Spacecraft resources, which describe the parameters of the modeled spacecraft. **Important note: Currently, CSALT only supports a single spacecraft, so the length of the list must be one.** In a GMAT script, the Spacecraft resource is set using the following script block.

```
Create Spacecraft ExampleSpacecraft1
<Set up ExampleSpacecraft1>

Create DynamicsConfiguration ExampleDynamicsConfiguration
ExampleDynamicsConfiguration.Spacecraft = {ExampleSpacecraft1}
```

4.6.3 Setting the Finite Burn

A DynamicsConfiguration resource **that is used by a Thrust phase** requires a list of FiniteBurn resources, which describe the control parameters of the modeled spacecraft. If a DynamicsConfiguration resource is only supplied to phases whose ThrustMode = Coast (via the phases' DynamicsConfiguration fields), then the user does not need to set the FiniteBurns field for that particular DynamicsConfiguration resource.

Important note: Currently, all FiniteBurn resources for a CSALT DynamicsConfiguration must be EMTGSpacecraft resources. GMAT FiniteBurn and ImpulsiveBurn resources are not valid FiniteBurn resources for a CSALT DynamicsConfiguration.

Important note: Currently, CSALT only supports a single spacecraft, so the length of the list must be one.

In a GMAT script, the FiniteBurn resource is set using the following script block.

```
Create EMTGSpacecraft ExampleEMTGSpacecraft1
<Set up ExampleEMTGSpacecraft1>

Create DynamicsConfiguration ExampleDynamicsConfiguration
ExampleDynamicsConfiguration.FiniteBurns = {ExampleEMTGSpacecraft1}
```

4.6.4 Setting the EMTG Tank Configuration

If a DynamicsConfiguration resource contains at least one FiniteBurn, at least one tank must be added to the EMTG tank configuration. Multiple tanks can be used; the fuel will be depleted in the order the tanks are listed in this field. If the fuel tank object does not allow negative masses, this will be changed internally to be allowed so that the optimizer can test any mass values as it attempts to reach convergence. New point functions and bounds are created automatically to attempt to only use the amount of fuel available. These point functions can be viewed at the end of a run in the GMAT Optimal Control solution report.

Additionally, he mass of the Spacecraft resource and the mass used in the optimal control problem are related. The sum of the dry mass of the Spacecraft set on a Phase's DynamicsConfiguration field and the starting fuel masses of all FuelTank resources attached to that Spacecraft must be equal to that Phase's StateUpperBound element corresponding to mass, or else an error is thrown.

Note: The chemical tank must be attached to the spacecraft being used.

```
Create Spacecraft ExampleSpacecraft1
ExampleSpacecraft1.Tanks = {ChemicalTank1};
<Set up ExampleSpacecraft1>

Create ChemicalTank ExampleChemTank1
<Set up ExampleChemTank1>

Create EMTGSpacecraft ExampleEMTGSpacecraft1
<Set up ExampleEMTGSpacecraft1>

Create DynamicsConfiguration ExampleDynamicsConfiguration
ExampleDynamicsConfiguration.Spacecraft = {ExampleSpacecraft1}
ExampleDynamicsConfiguration.FiniteBurns = {ExampleEMTGSpacecraft1}
ExampleDynamicsConfiguration.EMTGTankConfig = {ExampleChemTank1}
```

4.7 Optimal Control Guess Reference

A collocation optimizer like CSALT requires an initial guess in order to execute. The guess must contain values of the state and control at all times between the initial time and the final time at which the dynamics are evaluated.

In GMAT, the OptimalControlGuess resource is used to provide a guess for the trajectory. The Type field sets how the guess is set. If Type is set to CollocationGuessFile, then the guess is obtained from a user-specified .och-formatted file. In this case, the file name is provided by setting the field FileName. If the time at which states and controls are required is between two times that are present in the guess file, then CSALT internally performs interpolation to obtain state and control values at the intermediate time.

Warning: The guess file must have a minimum of five rows of data. If fewer than five rows are provided, an exception will occur because the polynomial interpolation used to obtain state and control values at intermediate times fails.

An example of creating an OptimalControlGuess GMAT resource and setting the guess resource to use a specified .och file for the initial guess is given below.

```
Create OptimalControlGuess trajectoryGuess
trajectoryGuess.Type = CollocationGuessFile
trajectoryGuess.FileName = ../data/misc/GuessWithUnityControl.och
```

Alternatively, Type may be set to GMATArray. As the name suggests, in this case, the guess is set my placing guess values in a GMAT Array and passing the array to the OptimalControlGuess resource. The input Array must be organized such that the first column is the independent variable (i.e., time), the next set of columns holds the states, and the final set of columns holds the controls. For example, a typical spacecraft trajectory optimization problem might have a guess array with 11 columns: [Time, Rx, Ry, Rz, Vx, Vy, Vz, Mass, Ux, Uy, Uz], where R is the position vector, V is the velocity vector, and U is the control vector. The subscripts x, y, z indicate vector components. (In fact, this 11-column format is currently required in GMAT.) Each row of the array corresponds to a different instant in time. (The Time column must increase monotonically as the row number increases.)

Warning: Like the guess file, the guess array must have a minimum of five rows of data. If fewer than five rows are provided, an exception will occur because the polynomial interpolation used to obtain state and control values at intermediate times fails.

The following presents a simple example.

```
Create Array guess[200,11]
guess(1,1) = 0
guess(1,2) = 5.5
% ... continuing setting the rest of the elements in guess

Create CoordinateSystem EarthICRFExample
EarthICRFExample.Origin = Earth
EarthICRFExample.Axes = ICRF

Create OptimalControlGuess trajectoryGuess
trajectoryGuess.Type = GMATArray
trajectoryGuess.TimeSystem = TDBModJulian
trajectoryGuess.CoordinateSystem = EarthICRFExample
```

As shown in the above example, when the OptimalControlGuess Type is set to GMATArray, the field TimeSystem is used to specify the time system in which the guess is specified, and the field CoordinateSystem is used to specify the coordinate system in which the guess is specified. TimeSystem must be a GMAT-supported modified-Julian-date time system, and CoordinateSystem must be an existing CoordinateSystem resource. TimeSystem and CoordinateSystem are not used if the OptimalControlGuess Type is set to CollocationGuessFile. In that case, the time system and coordinate system are read directly from the collocation guess file meta data.

4.8 EMTG Spacecraft Reference

The Evolutionary Mission Trajectory Generator (EMTG) is software developed at GSFC to optimize interplanetary space mission trajectories. CSALT uses an EMTG spacecraft file to define the capabilities of an in-space propulsion system.

4.8.1 Creating an EMTG Spacecraft Resource

In a GMAT script, an EMTGSpacecraft resource is created using:

```
Create EMTGSpacecraft ExampleEMTGSpacecraft1
```

4.8.2 Setting the EMTG Spacecraft File

EMTG spacecraft files are text files, typically with the extension emtg_spacecraftopt. The following GMAT script excerpt sets the EMTG spacecraft file to be used.

```
Create EMTGSpacecraft ExampleEMTGSpacecraft1
ExampleEMTGSpacecraft1.SpacecraftFile = ExampleEMTGSpacecraftFile.emtg_spacecraft
```

The SpacecraftFile field may be an absolute path or a path relative to gmat/data/emtg/.

4.8.3 Setting the EMTG Spacecraft Stage

A single EMTG spacecraft file may contain multiple spacecraft stages. Each stage has unique properties, such as its power system, its propulsion system, its dry mass, etc. A unique stage is set for each phase of a CSALT trajectory by providing EMTGSpacecraftStage with an integer array, as shown in the following example. The GMAT script excerpt sets the CSALT trajectory to use EMTG spacecraft stage 2 for the first phase, stage 3 for the second phase, and stage 1 for all subsequent phases. (EMTG spacecraft stage numbering is one-indexed and is the order in which the stage blocks appear in the EMTG spacecraft file.) Note that coast phases, in addition to thrust phases, are considered in this counting. In other words, if, in the below example, the first phase were a coast phase, then stage 2 would not be used. Stage 3 would still be used for the second phase, and stage 1 would still be used for the third phase and all subsequent phases.

```
Create EMTGSpacecraft ExampleEMTGSpacecraft1
ExampleEMTGSpacecraft1.SpacecraftStage = [2, 3, 1]
```

4.8.4 Setting the Propulsion System Duty Cycle

CSALT approximates a duty cycle by multiplying the calculated thrust magnitude by a constant factor between 0 and 1, inclusive. A value of 1 effectively means that no duty cycle is applied, and a value of 0 means that there is no thrust. The GMAT script excerpt below sets the duty cycle for an EMTGSpacecraft.

```
Create EMTGSpacecraft ExampleEMTGSpacecraft1
ExampleEMTGSpacecraft1.DutyCycle = 0.9
```

4.8.5 Format of EMTG Spacecraft File

An EMTG spacecraft file is a text file of data, typically with the extension emtg_spacecraft. The file consists of space-delimited lines of data. Lines of text that begin with a "#" are comment lines.

The file is separated into two types of "blocks":

1. Spacecraft block
2. One or more stage blocks

Each EMTG spacecraft file contains one spacecraft block, which contains "global" data that applies to all stage blocks. On the other hand, a single EMTG spacecraft file may contain multiple stage blocks. Each stage block defines a spacecraft stage (i.e., configuration).

Each stage block, in addition to containing "unblocked" data, contains a power library block and a propulsion library block. All data in a stage block is independent of all data in all other stage blocks. Thus, a file containing two spacecraft stages is organized as:

- Spacecraft block data
- Stage 1 block
  - Unblocked data
  - Power library block data
  - Propulsion library block data
- Stage 2 block
  - Unblocked data
  - Power library block data
  - Propulsion library block data

The data in each block, including format, is described in *EMTG Spacecraft File Specification*. In addition, when creating an EMTG Spacecraft file, it will likely be beneficial to consult the example .emtg_spacecraftopt files provided in gmat/data/emtg/.

4.8.6 Throttle Tables

One method by which a propulsion model may be set in an EMTG spacecraft file is by referencing an external throttle table file. A throttle table file is a comma-delimited text file of data, typically with extension ThrottleTable. The throttle table file describes discrete settings that the propulsion system may take, as well as best-fit polynomial coefficients of the discrete settings.

Format of Throttle Table File

A throttle table file is a comma-delimited file in which lines that begin with "#" are comment lines. An example of a populated throttle table file, including instructions of how to populate it, is given in a GMAT installation in gmat/data/emtg/NEXT_TT11_Discovery14_BOL_20190523.ThrottleTable.

4.9 Boundary Functions Reference

Boundary functions include algebraic boundary constraints and cost function contributions that are only dependent upon optimization parameters at phase boundaries.

CSALT supports two interfaces to define boundary constraints. The first interface, the UserPointFunction class, defines the interfaces for algebraic boundary functions, cost functions, and their bounds, but does not support analytic Jacobians. The OptimalControlFunction utility class is an alternative interface for defining boundary functions that supports analytic Jacobians, and makes it easy to re-use boundary function implementations in different problem configurations.

GMAT supports several interfaces for boundary functions. Those interfaces are discussed below. Note that GMAT uses the CSALT interfaces to implement boundary functions. However, that complexity is largely hidden from the user and configuring boundary constraints in GMAT is presented in separate sections below.

4.9.1 Setting C++ Boundary Functions via the UserPointFunction Class

Boundary functions are defined by deriving a new class from the CSALT abstract UserPointFunction base class illustrated in Fig. 4.10. (Note: Only key functions are illustrated below; see Doxygen output for the full reference.)

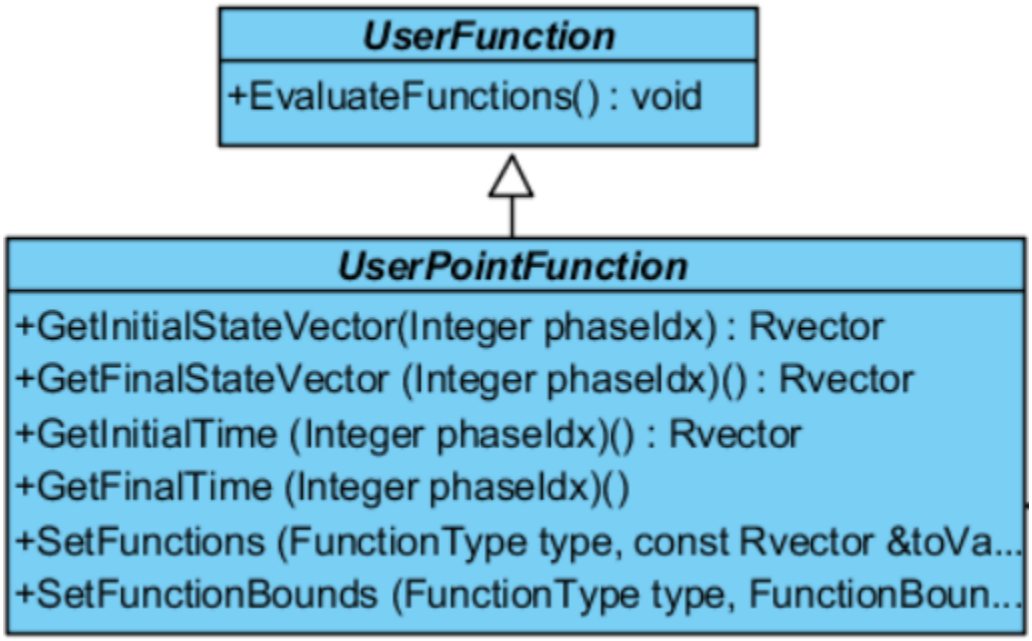

Fig. 4.10: CSALT UserFunction and UserPointFunction classes.

When using those interfaces on the UserPointFunction object, the user must concatenate all boundary functions and bounds and provide all functions in a single vector. In that case, CSALT uses finite differencing of all Jacobians. The UserPointFunction class provides Get() and Set() methods that get state data from CSALT to compute algebraic boundary constraints and cost functions. Functions are set by calling the SetFunctions method and providing the appropriate enumeration for the function type. To set algebraic functions, use the enumeration ALGEBRAIC. To set cost functions, use the enumeration COST. The code snippet below shows how to define the boundary functions for an example problem using Set() and Get() methods provided on UserPointFunction. For a complete reference, see the Doxygen output for UserFunction and UserPointFunction.

```
void BrysonDenhamPointObject::EvaluateFunctions()
{
    // Extract parameter data
    Rvector stateInit = GetInitialStateVector(0);
    Rvector stateFinal = GetFinalStateVector(0);
    Real tInit = GetInitialTime(0);
    Real tFinal = GetFinalTime(0);

    // Set the cost function passing in COST as the first parameter
    // and the cost function value as the second parameter
    Rvector costFunctions(1, stateFinal(2));
    SetFunctions(COST, costFunctions);

    // Compute the algebraic functions and bounds
    Rvector algFunctions(7, stateInit(0), stateInit(1), stateInit(2),
                            stateFinal(0), stateFinal(1), tInit, tFinal);
```

(continues on next page)

(continued from previous page)

```
    Rvector algFuncLower(7, 0.0, 1.0, 0.0, 0.0, -1.0, 0.0, 1.0);
    Rvector algFuncUpper(7, 0.0, 1.0, 0.0, 0.0, -1.0, 0.0, 1.0);

    // Set the algebraic functions with ALGEBRAIC as the first parameter
    // and the function values as the second parameter
    SetFunctions(ALGEBRAIC, algFunctions);
    SetFunctionBounds(ALGEBRAIC, LOWER, algFuncLower);
    SetFunctionBounds(ALGEBRAIC, UPPER, algFuncUpper);
}
```

4.9.2 Setting C++ Boundary Functions via the OptimalControlFunction Class

CSALT provides an interface to set boundary functions one at a time by providing an std vector of OptimalControlFunction objects. The OptimalControlFunction class defines the generic interface for granularly creating and providing to CSALT boundary functions. By using the OptimalControlFunction class, the user does not need to manage the concatenation of large numbers of boundary functions and the careful bookkeeping required to construct the overall NLP Jacobian from the Jacobian of individual functions. The ability to provide functions at a granular level is particularly important for large problems that often have hundreds of boundary functions and complicated dependencies. Additionally, using the OptimalControlFunction class allows the user to easily re-use an optimal control function created for one problem for a different problem configuration.

The class diagram for OptimalControlFunction with key data and methods is shown in Fig. 4.11. (See doxygen output for complete, up-to-date reference material.)

To configure a problem to use constraints created using the OptimalControlFunction class, derive a new class from the OptimalControlFunction base class and implement the EvaluateFunctions() method to compute the function values. The example below shows how to configure the boundary functions for the Brachistichrone optimization problem. The class BranchBoundConstraint is derived from OptimalControlFunction. When data is accessed to form function values, the first index represents which discretization point in the function the data vector is taken from, while the second index represents which element of that vector to use. For example, stateData[0][1] is the second element in the state vector at the first discretization point.

```
Rvector BrachBoundConstraint::EvaluateFunctions()
{
    Rvector algF(numFunctions);
    algF(0) = timeData[0][0];
    algF(1) = timeData[1][0];
    algF(2) = stateData[0][0];
    algF(3) = stateData[0][1];
    algF(4) = stateData[0][2];
    algF(5) = stateData[1][0];
    algF(6) = stateData[1][1];
    algF(7) = stateData[1][2];

    return algF;
}
```

Optimal control functions require information on the phases involved, the point type (initial, final, all), and the state dependencies. The values are typically configured in the problem driver, or, when integrated with GMAT, GMAT sets these values based on the user's scripted configuration. The code below illustrates how to configure the Brachistichrone bounds functions using an OptimalControlFunction object; the class BranchBoundConstraint is derived from OptimalControlFunction. Note that multiple OptimalControlFunction objects can be added to a UserPointFunction

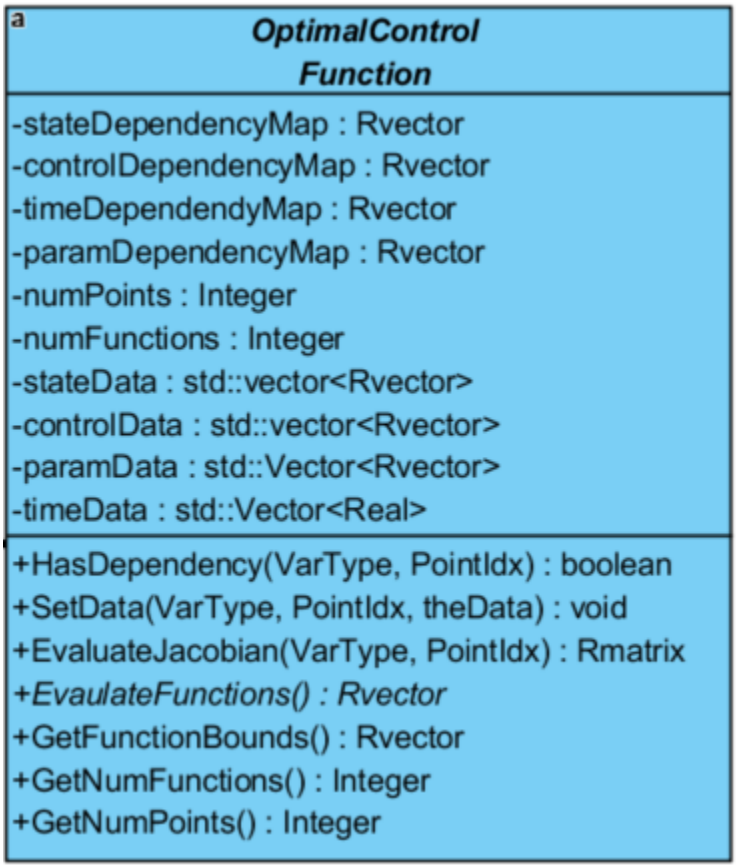

Fig. 4.11: CSALT OptimalControlFunction class.

object (which is then added to a Trajectory) through the vector sent to the AddFunctions() method seen at the end of this example.

```
// This function is dependent on the start of phase 0 and end of phase 0.
IntegerArray phaseDependencies(2,0,0);
// 0 for start, 1 for end, 2 for all points (i.e. path constraint)
IntegerArray pointDependencies(2,0,1);

// Define the functional dependencies
Real numPhases = 2;
Real numFunctions = 8;
BooleanArray stateDepMap(); // Dependent upon state at both points
stateDepMap.push_back(true);
stateDepMap.push_back(true);
BooleanArray controlDepMap(); // No control dependency
controlDepMap.push_back(false);
controlDepMap.push_back(false);
BooleanArray timeDepMap(); // Dependent upon time at both points
timeDepMap.push_back(true);
timeDepMap.push_back(true);
BooleanArray paramDepMap(); // No static params dependency
paramDepMap.push_back(false);
paramDepMap.push_back(false);

// Create the function object and initialize
OptimalControlFunction* brachFunc = new BrachBoundConstraint("BrachBound");
brachFunc->Initialize();

// Set the various dependencies
brachFunc->SetNumPhases(numPhases);
brachFunc->SetNumFunctions(numFunctions);
brachFunc->SetPhaseDependencies(phaseDependencies);
brachFunc->SetPointDependencies(pointDependencies);
brachFunc->SetStateDepMap(stateDepMap);
brachFunc->SetControlDepMap(controlDepMap);
brachFunc->SetTimeDepMap(timeDepMap);
brachFunc->SetParamDepMap(paramDepMap);

// Create and set the function bounds
Rvector lowerBounds(8,0.0,  0.0,0.0,0.0,0.0,1.0,-10.0,-10.0);
Rvector upperBounds(8,0.0,100.0,0.0,0.0,0.0,1.0, 10.0,  0.0);

brachFunc->SetUpperBounds(upperBounds);
brachFunc->SetLowerBounds(lowerBounds);

// Add the new function object to the list of functions
std::vector<OptimalControlFunction*> funcList;
funcList.push_back(brachFunc);

pointObject = new BrachistichronePointObject(); // derived from UserPointFunction
pointObject->AddFunctions(functionList);
```

Analytic functions and Jacobians are set up as part of the CSALT initialization process, and then used during optimization to evaluate the supplied functions and derivatives.

- **Custom Function Bounds:** A user's OptimalControlFunction object may allow or require custom bounds. Custom bounds are set through the SetLowerBounds(Rvector functionLB) and SetUpperBounds(Rvector functionUB) methods on the OptimalControlFunction object during initialization. If the selected OptimalControlFunction does not allow for custom bounds, an error is reported.
- **Custom Scaling:** For some problems, custom scaling may improve the optimization process. Custom scaling is implemented through the SetScaleFactors(Real referenceEpoch, Real timeUnit, Real distUnit, Real velUnit, Real massUnit) method in the AlgebraicFunction-derived class.[1] When SetScaleFactors() is called, function values and bounds are scaled using user-defined scale factors during optimization. Note that when custom scaling is used, the reference epoch must be provided as an A1 modified Julian date.

An example of the use of these two custom function bounds and custom scaling may look like this in a user's problem driver (note that the dependency setup was removed for simplicity, please refer to the Brachistichrone example above for use of dependencies):

```
OptimalControlFunction* stateFunc = new FullState1("StateBound");

// Apply custom state bounds
Rvector lowerBounds(7, 7100.0, 1500.0, 0.0, 0.0, 10.0, 0.0, 1000.0);
Rvector upperBounds(7, 7100.0, 1500.0, 0.0, 0.0, 10.0, 0.0, 1000.0);
stateFunc->SetLowerBounds(lowerBounds);
stateFunc->SetUpperBounds(upperBounds);

// Apply scale factors
Real refEpoch = 28437.0;
Real TU = 1.0;
Real DU = 1000.0;
Real VU = 10.0;
Real MU = 1000.0;
stateFunc->SetScaleFactors(refEpoch, TU, DU, VU, MU);

std::vector<OptimalControlFunction*> funcList;
funcList.push_back(stateFunc);
FullStatePointObject->AddFunctions(funcList); // FullStatePointObject's class is derived␣
↪from UserPointFunction
```

4.9.3 Optimal Control Function Example (C++)

Below is an example of the BrachistichronePointObject class being converted to use the OptimalControlFunction interface. Assume the OptimalControlFunction object for this problem is called BrachistichroneBounds. The constraint bounds originally found in the BrachistichronePointObject class are moved to the constructor of the new BrachistichroneBounds class, while the point functions are moved to the EvaluateFunctions() method in BrachistichroneBounds.

The first code block is the BrachistchronePointObject class derived from the UserPointFunction class.

```
void BrachistichronePointObject::EvaluateFunctions()
{
    // This method no longer needs any code
}

void BrachistichronePointObject::EvaluateJacobians()
```

(continues on next page)

[1] The ScaleUtility class also provides capabilities for scaling.

(continued from previous page)

```
{
    // This method no longer needs any code
}
```

Here is what the BrachistichroneBounds constructor looks like to set up default bounds and dependencies. Note that these values can be changed by the user through SetLowerBounds(), SetUpperBounds(), and various set dependancy map methods such as SetStateDepMap(). The required default parameters for a class derived from OptimalControlFunction are the number of points; all other parameters can be set by the user outside of the constructor.

The EvaluateFunctions() method is also shown. The first and second points in this problem are the initial point of the phase and the final point of the phase, respectively. (The trajectory is made up of a single phase.)

This code block shows the constructor and EvaluateFunctions() method of the BrachistichroneBounds class derived from the OptimalControlFunction class.

```
BrachistichroneBounds::BrachistichroneBounds(std::string funcName) :
    OptimalControlFunction(funcName)
{
    numPoints = 2;
    numFunctions = 8;
    stateDepMap.resize(numPoints);
    controlDepMap.resize(numPoints);
    timeDepMap.resize(numPoints);
    paramDepMap.resize(numPoints);
    for (Integer i = 0; i < numPoints; ++i)
    {
        stateDepMap.at(i) = true;
        controlDepMap.at(i) = false;
        timeDepMap.at(i) = true;
        paramDepMap.at(i) = false;
    }

    // Set default bound values
    lowerBounds.SetSize(numFunctions);
    upperBounds.SetSize(numFunctions);
    lowerBounds(0) = 0.0;
    upperBounds(0) = 0.0;
    lowerBounds(1) = 0.0;
    upperBounds(1) = 100.0;
    lowerBounds(2) = 0.0;
    upperBounds(2) = 0.0;
    lowerBounds(3) = 0.0;
    upperBounds(3) = 0.0;
    lowerBounds(4) = 0.0;
    upperBounds(4) = 0.0;
    lowerBounds(5) = 1.0;
    upperBounds(5) = 1.0;
    lowerBounds(6) = -10.0;
    upperBounds(6) = 10.0;
    lowerBounds(7) = -10.0;
    upperBounds(7) = 0.0;
}
```

(continues on next page)

(continued from previous page)

```
Rvector BrachistichroneBounds::EvaluateFunctions()
{
    Rvector algF(numFunctions);
    algF(0) = timeData[0][0];
    algF(1) = timeData[1][0];
    algF(2) = stateData[0][0];
    algF(3) = stateData[0][1];
    algF(4) = stateData[0][2];
    algF(5) = stateData[1][0];
    algF(6) = stateData[1][1];
    algF(7) = stateData[1][2];

    return algF;
}
```

To provide an analytic Jacobian when using OptimalControlFunction, the method EvaluateAnalyticJacobian() is used. The prototype of the method is:

```
void OptimalControlFunction::EvaluateAnalyticJacobian(VariableType varType,
   Integer pointIdx, bool &hasAnalyticJac, Rmatrix &jacArray)
```

The user creates an EvaluateAnalyticJacobian() method on their derived OptimalControlFunction class. Within the method, the user calculates and sets the hasAnalyticJac and jacArray arguments based on the values of varType and pointIdx.

The variable varType is an enumeration that describes the variable with respect to which the function is differentiated. The possible values for for varType are [STATE, CONTROL, TIME, STATIC].

The variable pointIdx represents at which discretization point the Jacobian is being calculated. In other words, if pointIdx == i, then EvaluateAnalyticJacobian() must set the derivatives of the optimal control function with respect to variables (time, state, control, static parameters) at discretization point i. Recall that the discretization point dependencies are set by setting the SetNumPhases, SetPhaseDependencies, and SetPointDependencies OptimalControlFunction methods.

The variable hasAnalyticJac is set by the user to true if the user provides an analytic Jacobian and to false if the user does not provide an analytic Jacobian. If hasAnalyticJac is set to false, then finite differencing is used to approximate the Jacobian. Note that hasAnalyticJac may be set differently for different values of varType, as shown in the example below.

The user sets the variable jacArray to the hold the derivatives of the optimal control function with respect to the variable(s) specified by varType:

$$\texttt{jacArray}(i,j) = \frac{\partial\left[\texttt{function}_i\right]}{\partial\left[\texttt{variable}_j\right]}$$

Note that the indexing for the variable resets when varType is changed but the indexing for function does NOT change when varType is changed. For example, if the optimal control function has three elements, and there is one time variable and seven state variables, then:

- if varType == TIME, the dimensions of jacArray are 3×1
- if varType == STATE, the dimensions of jacArray are 3×7

The code below gives an example of an EvaluateAnalyticJacobian method for a full-state linkage constraint for a seven-element state vector. Note that, while this example does not reference any discretization point data, discretization point data may be obtained via the timeData, stateData, etc. class variables, as was done in the BrachBoundConstraint::EvaluateFunctions() example code snippet.

```
void FullStateLinkage::EvaluateAnalyticJacobian(VariableType varType,
   Integer pointIdx,
   bool &hasAnalyticJac,
   Rmatrix &jacArray)
{
   ValidatePointIdx(pointIdx); // OptimalControlFunction method to ensure pointIdx is␣
↪within bounds
   hasAnalyticJac = false; // assume false until proven otherwise

   switch (varType) // Jacobian calculation differs based on varType
   {
   case TIME:
      hasAnalyticJac = true; // we are calculating analytic Jacobian
      jacArray.SetSize(numFunctions, 1); // one time variable
      // function 0 is time_1 - time_0
      // pointIdx 0 corresponds to time_0
      // pointIdx 1 corresponds to time_1
      if (pointIdx == 0)
         jacArray(0, 0) = -1.0;
      else if (pointIdx == 1)
         jacArray(0, 0) = 1.0;
      break;
   case STATE:
      hasAnalyticJac = true; // we are calculating analytic Jacobian
      jacArray.SetSize(numFunctions, 7); // seven state variables
      // function j for j = 1, ..., 7 is state_element_j-1 (time_1) - state_element_j-1␣
↪(time_0)
      if (pointIdx == 0)
      {
         jacArray(1, 0) = -1.0;
         jacArray(2, 1) = -1.0;
         jacArray(3, 2) = -1.0;

         jacArray(4, 3) = -1.0;
         jacArray(5, 4) = -1.0;
         jacArray(6, 5) = -1.0;

         jacArray(7, 6) = -1.0;
      }
      else if (pointIdx == 1)
      {
         jacArray(1, 0) = 1.0;
         jacArray(2, 1) = 1.0;
         jacArray(3, 2) = 1.0;

         jacArray(4, 3) = 1.0;
         jacArray(5, 4) = 1.0;
         jacArray(6, 5) = 1.0;

         jacArray(7, 6) = 1.0;
      }
      break;
   default: // no dependency on control or static parameters
```

(continues on next page)

(continued from previous page)

```
      OptimalControlFunction::EvaluateAnalyticJacobian(varType, pointIdx,
         hasAnalyticJac, jacArray); // simply error-checks and breaks
   }
}
```

4.9.4 Setting Boundary Functions in GMAT

Setting Full State Linkage Constraints

The GMAT CSALT interface supports a constraint function interface that connects phases via full time and state linkage. The script below configures a simple linkage between the two phases named firstPhase and secondPhase.

GMAT Script Example

```
aTrajectory.AddSimpleLinkageChain = {firstPhase,secondPhase}
```

The simple linkage constraint computes the linkages by subtracting values of the first phase from the second phase as shown below.

$$\Delta t = \text{secondPhase Start A1MJD} - \text{firstPhase End A1MJD}$$

Similarly, if v_y is the velocity difference, it is computed as

$$\Delta v_y = \text{secondPhase Start y component} - \text{firstPhase End y component}$$

Multiple simple linkages can be added as a complete list if desired. The script below shows four phases being linked together.

```
aTrajectory.AddSimpleLinkageChain = {firstPhase, secondPhase, thirdPhase, fourthPhase}
```

Before returning the function values to CSALT, the linkage constraints are non-dimensionalized. Time is non-dimensionalized, and the TU is chosen as the smallest TU between the two phases.

The bounds for simple linkages are all zeros for both the upper and lower bounds and do not require non-dimensionalization before providing to CSALT.

Setting Custom Linkage Constraints

A CustomLinkageConstraint allows customized constraints on time, position, velocity, and/or mass parameters between phases, or between user-provided state data. To define a custom linkage constraint, the user must define the phases involved, the constraint mode (absolute or difference), which quantities to constrain, and bounds on the constrained quantities. We begin by discussing the constraint modes and then discuss how to define specific constraint quantities.

Difference Mode

The CustomLinkageConstraint supports two modes set using the ConstraintMode field: Difference and Absolute. In Difference mode, the constraint function differences values between specified phases and boundary points of the phases (PhaseBoundaryType = Start or End) according to the following equation:

$$g = \text{FinalPhase.FinalPhaseBoundaryType.Parameter} - \text{InitialPhase.InitialPhaseBoundaryType.Parameter}$$

The example below illustrates how to constrain the time between the start of a phase called launchPhase and the start of a phase called thrustPhase to be greater than or equal to 30 days and less than or equal to 40 days. NIf either an upper or lower bound is not necessary, the user does not have to set it, and the system will use the defaults described in *Defining Constraint Types*.

```
% Define a custom linkage constraint that constrains time between phase
% boundaries be >= 30 days and <= 40 days
Create CustomLinkageConstraint aCustomLinkage
aCustomLinkage.ConstraintMode = 'Difference';
aCustomLinkage.InitialPhase = launchPhase;
aCustomLinkage.InitialPhaseBoundaryType = Start;
aCustomLinkage.FinalPhase = thrustPhase;
aCustomLinkage.FinalPhaseBoundaryType = Start;
aCustomLinkage.SetModelParameter('TimeLowerBound','ElapsedDays', 30)
aCustomLinkage.SetModelParameter('TimeUpperBound','ElapsedDays', 40)

% Add the constraint to the trajectory
traj.CustomLinkages =  {aCustomLinkage}
```

Absolute Mode

When ConstraintMode is set to Absolute, the linkage constraints are computed as absolute parameter quantities on the initial phase.

$$g = \text{InitialPhase.InitialPhaseBoundaryType.Parameter}$$

To further illustrate how constraints are constructed, the example below enforces an equality constraint on the initial time, position, velocity, and mass at the start of a phase named launchPhase.

```
Create CustomLinkageConstraint aCustomLinkage
aCustomLinkage.ConstraintMode = Absolute;
aCustomLinkage.InitialPhase = launchPhase;
aCustomLinkage.InitialPhaseBoundaryType = Start;
aCustomLinkage.SetModelParameter(
              'TimeLowerBound', UTCGregorian, 01 Jan 2034 12:23:45.111)
aCustomLinkage.SetModelParameter(
             'TimeUpperBound', UTCGregorian, 01 Jan 2034 12:23:45.111)
aCustomLinkage.SetModelParameter(
             'PositionLowerBound', [125291184.0 -75613036.0 -32788527.0])
aCustomLinkage.SetModelParameter(
             'PositionUpperBound', [125291184.0 -75613036.0 -32788527.0])
aCustomLinkage.SetModelParameter(
             'VelocityLowerBound', [13.438 25.234 10.903])
aCustomLinkage.SetModelParameter(
```

(continues on next page)

(continued from previous page)

```
            'VelocityUpperBound', [13.438 25.234 10.903])
aCustomLinkage.SetModelParameter('MassLowerBound', 4000)
aCustomLinkage.SetModelParameter('MassUpperBound', 4000)
```

Defining Constraint Types

Constraint types are defined by setting a bound on the desired parameter by calling the SetModelParameter() method. If a bound is not set for a parameter, then no constraint is applied on that parameter. The user can optionally set the upper and/or lower bounds. If only one bound is set, the default is used for the unset bound. See Table 4.4 for supported parameter types, descriptions, and default bounds. The general syntax for setting constraint parameters and bounds is

ConstraintName.SetModelParameter(Parameter, arg1, ..., argN)

The argument list is overloaded and is dependent upon the parameter type. Table 4.4 defines the argument list for each parameter type.

Table 4.4: Setting GMAT Optimal Control constraints.

Parameter	Description
PositionUpperBound PositionLowerBound	Defines the upper and lower bounds on the position vector (in the selected phase's coordinate system if in absolute mode). **Example** aCustomLinkage.SetModelParameter('PositionLowerBound', [-1e7 -1e7 -1e7]) aCustomLinkage.SetModelParameter('PositionUpperBound', [1e7 1e7 1e7]) **Default Bounds and Units** Unit = km Lower = 100*[-Max(Real) -Max(Real) -Max(Real)] Upper = 100*[Max(Real) Max(Real) Max(Real)]
VelocityUpperBound VelocityLowerBound	Defines the upper and lower bounds on the velocity vector (in the selected phase's coordinate system if in absolute mode). **Example** aCustomLinkage.SetModelParameter('VelocityLowerBound', [-1e2 -1e2 -1e2]) aCustomLinkage.SetModelParameter('VelocityUpperBound', [1e2 1e2 1e2]) **Default Bounds and Units** Unit = km/s Lower = 100*[-Max(Real) -Max(Real) -Max(Real)] Upper = 100*[Max(Real) Max(Real) Max(Real)]
MassUpperBound MassLowerBound	Defines the upper and lower bounds on the total spacecraft mass. **Example** aCustomLinkage.SetModelParameter('MassLowerBound', 1e-7) aCustomLinkage.SetModelParameter('MassUpperBound', 1e4) --- **Note:** The lower bound on mass cannot be set less than 1e-7 kg. --- **Default Bounds and Units** Unit = kg Lower = 1e-7 Upper = 100*Max(Real)
TimeUpperBound TimeUpperBound	Defines the upper and lower bounds on time. Note, this field has a dependency on the ConstraintMode setting. The argument list is different depending upon the selected mode as shown below. The function prototype is: SetModelParameter(ParameterName, Epoch Format, Epoch Value) **Example in Absolute Mode** aCustomLinkage.SetModelParameter('TimeLowerBound', TAIModJulian, 32060.0) aCustomLinkage.SetModelParameter('TimeUpperBound', TAIModJulian, 32560.0) All GMAT epoch formats are supported. **Default Bounds and Units in Absolute Mode** The default bounds when ConstraintMode is Absolute are LowerBound = 04 Oct 1957 12:00:00.000 UTC UpperBound = 28 Feb 2100 00:00:00.000 UTC **Warning:** Many SPICE files do not cover the default time span. In that case you may need to specifically set the time bounds to be consistent with the ephemeris files. **Example in Difference Mode** aCustomLinkage.SetModelParameter('TimeLowerBound', 'ElapsedDays', 32060.0) aCustomLinkage.SetModelParameter('TimeUpperBound', 'ElapsedDays', 32560.0) Epoch formats are: ElapsedDays, ElapsedSeconds **Default Bounds and Units in Difference Mode** Units are determined by Time Format input. Lower Bound = -100*Max(Real) Upper Bound = 100*Max(Real)

Custom linkage functions and bounds are scaled before being sent to CSALT. Time is non-dimensionalized using the smallest TU between the two phases. The smallest TU also determines which of the other scale factors are used between the two phases.

Setting a Celestial Body Rendezvous Constraint

The GMAT CSALT interface supports a rendezvous constraint that enforces time and Cartesian state equality between a user-specified phase and a user-specified celestial body. The GMAT script to configure a rendezvous between the end of a Phase named ArrivalPhase and a celestial body named RQ36 is shown below.

```
Create OptimalControlFunction CometRendezvous
CometRendezvous.Function = CelestialBodyRendezvous;
CometRendezvous.Type = AlgebraicConstraint
CometRendezvous.PhaseList = {arrivalPhase.Final}
CometRendezvous.SetModelParameter('CelestialBody', RQ36)
```

Warning: The gravitational force of a body with which a spacecraft is to perform a celestial body rendezvous should be turned OFF for the phase(s) in which the rendezvous is to occur to avoid a potential singularity in the dynamics model.

Setting an Integrated Flyby Constraint

The GMAT CSALT interface provides an integrated flyby constraint given two user-specified phases and a user-specified celestial body. The flyby is modeled at the end of the first phase and the beginning of the second phase. The GMAT script to configure an integrated flyby between the end of a Phase named PreEGA1 and the beginning of a Phase named PostEGA1 and a celestial body named Earth is shown below.

Note: Note: To configure an integrated flyby constraint, the user must also apply a simple linkage constraint between the two phases in the flyby. The simple linkage constraint applies full-state continuity at the linkage point, and the integrated flyby constraint ensures the phase boundary is at periapsis and the magnitude of the position is within the bounds.

```
Create BoundaryFunction EarthGravAssist
EarthGravAssist.Function = IntegratedFlyBy
EarthGravAssist.Type = AlgebraicConstraint
EarthGravAssist.PhaseList = {phase1.Final,phase2.Initial}
EarthGravAssist.SetModelParameter('CelestialBody','Earth')
EarthGravAssist.SetModelParameter('PeriapsisRadiusLowerBound',6000)
EarthGravAssist.SetModelParameter('PeriapsisRadiusUpperBound',100000)
```

Setting a Patched Conic Launch Constraint

The patched conic launch model uses performance polynomials to compute mass-to-orbit for a launch vehicle and the transfer orbit insertion state. The launch is modeled at the beginning of the phase. To configure a patched conic launch, provide the phase, the central body name, the launch vehicle model file, and the name of the launch vehicle as shown in the example below. Note: To constrain the launch epoch, apply a CustomLinkageConstraint in addition to the Patched Conic Launch Constraint.

```
Create OptimalControlFunction pcLaunch
pcLaunch.Type = AlgebraicConstraint
pcLaunch.Function = PatchedConicLaunch
pcLaunch.PhaseList = {phase1.Initial}
pcLaunch.SetModelParameter(VehicleName, Atlas_V_401)
pcLaunch.SetModelParameter(CentralBody,Earth)
pcLaunch.SetModelParameter(EMTGLaunchVehicleOptionsFile, ../data/emtg/filename.emtg_
↪launchvehicleopt)
```

Patched Conic Launch Model File

A launch model file contains data for different launch vehicles by vehicle name. The file /gmat/data/emtg/NLSII_August2018_Augmented.emtg_launchvehicleopt is distributed with GMAT. The file can be edited to add additional data and new files may be created and referenced. The data in the launch file is organized as described below.

The first few lines are a header, to be ignored as comments (using the # symbol), which describes the proceeding tokens. Each line contains 8 tokens in order as listed, and all elements are delimited via whitespace.

1. Launch vehicle name (string)
2. Model type (integer); always zero, represents polynomial C3 curve
3. Declination of launch asymptote (DLA) lower bound, degrees (double)
4. DLA upper bound, degrees (double)
5. C3 lower bound, km^2/s^2 (double)
6. C3 upper bound, km^2/s^2 (double)
7. Launch vehicle adapter mass (double), always zero for our purposes
8. Coefficients for the polynomial giving mass-to-orbit as a function of C3 (double). The first coefficient in a line is the 0th-order coefficient, to be multiplied by C3^0, followed by the 1st-order coefficient to be multiplied by C3^1 and so on.

Setting a Custom In-Line Constraint

An in-line constraint allows a user to set constraints from within a GMAT script using a scripted equation. Using this capability, a user can quickly set a simple constraint without writing a new CSALT class. In the examples below, the argument to SetModelParameter following 'Expression' may be any GMAT Equation.

```
Create OptimalControlFunction boundCon
boundCon.Function = Expression
boundCon.Type = AlgebraicConstraint
boundCon.PhaseList = {phase4.Final}
boundCon.SetModelParameter('LowerBounds', -6000)
```

(continues on next page)

(continued from previous page)

```
boundCon.SetModelParameter('UpperBounds', -6000)
boundCon.SetModelParameter('Expression', 'mySat.EarthMJ2000eq.BdotT')
boundCon.SetModelParameter('ScaleFactor', 6000)
```

Setting Maximum Mass Cost Function

See *Built-In Cost and Constraint Functions*.

Set Time Cost Function

See *Built-In Cost and Constraint Functions*.

Set RMAG Cost Function

See *Built-In Cost and Constraint Functions*.

Built-In Cost and Constraint Functions

The GMAT Phase Resource supports a set of "built-in" cost and constraint functions. These are set on the Phase Resource, but are documented here to keep the documentation of cost and constraint functions consolidated to a single chapter. A script example for setting the cost function to total mass is shown below, followed by a section documenting all supported built-in cost functions.

```
// Define the cost function as the total spacecraft mass at the end of "thePhase".
thePhase.BuiltInCost = 'TotalMassFinal'
```

Built-in Cost Functions

The GMAT Phase Resource supports several built-in cost functions. The names of those functions and the descriptions are contained in Table 4.5.

Table 4.5: Built-in cost functions.

Cost Function Name	Description
RMAGFinal	The magnitude of the Cartesian state at the end of the phase computed with respect to the central body of the ForceModel used in the Phase's dynamics configuration.
TotalMassFinal	The total spacecraft mass at the end of the phase.
AbsoluteEpochFinal	The A1 modified Julian epoch at the end of the phase.

The built-in cost functions may be minimized or maximized based on the value of Trajectory.OptimizationMode.

4.10 Path Functions Reference

Path function types include dynamics models, algebraic path constraints, and integral cost functions. CSALT and GMAT interfaces for defining these function types and their Jacobians are described below with detailed examples.

4.10.1 Setting Dynamics Models in C++

The example below shows how to set the dynamics and Jacobians for the Brachistichrone test problem. The BrachistichronePathObject class is derived from UserPathFunction.

```
// Setting the dynamics model equations
void BrachistichronePathObject::EvaluateFunctions()
{
    // Extract parameter data
    Rvector yVec = GetStateVector();
    Rvector uVec = GetControlVector();

    Real    u = uVec(0);
    Real    y  = yVec(2) * sin(u);
    Real    y2 = yVec(2) * cos(u);
    Real    y3 = gravity * cos(u);

    // Evaluate dynamics and compute Jacobians
    Rvector dynFunctions(3,y, y2, y3);
    SetFunctions(DYNAMICS, dynFunctions);
}

// Setting the Dynamics Model Jacobians
void BrachistichronePathObject::EvaluateFuncJacobians()
{
    // Get state and control
    Rvector stateVec = GetStateVector();
    Rvector controlVec = GetControlVector();
    Real u = controlVec(0);
    Real x = stateVec(0);
    Real y = stateVec(1);
    Real v = stateVec(2);

    // The dynamics state Jacobian
    Real dxdot_dv = sin(u);
    Real dydot_dv = cos(u);
    Rmatrix dynState(3, 3,
        0.0, 0.0, dxdot_dv,
        0.0, 0.0, dydot_dv,
        0.0, 0.0, 0.0);

    // The dynamics control Jacobian
    Real dxdot_du  = v * cos(u);
    Real dydot_du = -v * sin(u);
    Real dvdot_du = -gravity * sin(u);
    Rmatrix dynControl(3, 1, dxdot_du, dydot_du, dvdot_du);
```

(continues on next page)

(continued from previous page)

```
    // The dynamics time Jacobian
    Rmatrix dynTime(3, 1, 0.0, 0.0, 0.0);

    // Set the Jacobians
    SetJacobian(DYNAMICS, STATE, dynState); // derivatives of dynamics with respect to␣
↪state
    SetJacobian(DYNAMICS, CONTROL, dynControl); // derivatives of dynamics with respect␣
↪to control
    SetJacobian(DYNAMICS, TIME, dynTime); // derivatives of dynamics with respect to time
}
```

4.10.2 Setting Dynamics Models in GMAT

The Resource that defines the dynamics for a phase is the DynamicsConfiguration Resource. To set up a DynamicsConfiguration, attach pre-configured Spacecraft, ForceModel, Fuel Tank, and thrust model Resources. The example below illustrates how to configure those models.

Configure the Fuel Tank

A fuel tank is required to handle the fuel mass used by the spacecraft when performing thrust.

```
% Create a chemical tank
Create ChemicalTank ChemicalTank1;
ChemicalTank1.AllowNegativeFuelMass = false;
ChemicalTank1.FuelMass = 4150;
ChemicalTank1.Pressure = 1500;
ChemicalTank1.Temperature = 20;
ChemicalTank1.RefTemperature = 20;
ChemicalTank1.Volume = 3.5;
ChemicalTank1.FuelDensity = 1260;
ChemicalTank1.PressureModel = PressureRegulated;
```

Configure the Spacecraft

Configuring a spacecraft for this example is relatively simple. When we configure the guess data later in the tutorial, which includes spacecraft state and mass, those settings on the Spacecraft will be set according to the guess, so setting state and epoch information is not required here. We simply need to create a spacecraft and add a fuel tank.

```
% Create a spacecraft names aSat. Guess is set later.
Create Spacecraft aSat
aSat.Tanks = {ChemicalTank1};
```

Configure Orbital Dynamics Models

Below is the script configuration for a simple Sun-centered dynamics model with Earth, Sun, and Moon point masses included in the model.

```
% Create an orbit dynamics model with Earth, Sun, Moon point mass
Create ForceModel DeepSpaceForceModel;
GMAT DeepSpaceForceModel.CentralBody = Sun;
GMAT DeepSpaceForceModel.PointMasses = {Sun,Earth,Luna};
GMAT DeepSpaceForceModel.Drag = None;
GMAT DeepSpaceForceModel.SRP = On;
```

Configure the Thruster Models

The thrust model is configured by setting up EMTG options files and setting those files on an EMTGSpacecraft Resource. The EMTG options files contain the configuration for the thrust model. See *EMTG Spacecraft Reference* for additional information.

```
% Create an emtgThrustModel Resource
Create EMTGSpacecraft emtgThrustModel;
emtgThrustModel.SpacecraftFile = FixedThrustAndIsp_Model1.emtg_spacecraftopt
emtgThrustModel.SpacecraftStage = [1]
emtgThrustModel.DutyCycle = 1.0;
```

Note: The EMTGDataPath field specifies the path relative to the GMAT bin directory.

Configure the Dynamics Configuration

Now that the Spacecraft, Fuel Tank, ForceModel, and EMTGSpacecraft Resources are configured, they are added to a DynamicsConfiguration Resource.

```
% Create a DynamicsConfiguration object and add spacecraft
% ForceModel and Thrust model
Create DynamicsConfiguration SunThrustDynConfig
SunThrustDynConfig.DynamicsModels = {DeepSpaceForceModel}
SunThrustDynConfig.Spacecraft = {aSat}
SunThrustDynConfig.FiniteBurns = {emtgThrustModel}
SunThrustDynConfig.EMTGTankConfig = {ChemicalTank1}
```

4.10.3 Setting Integral Cost Functions

C++ Example

The example below shows how to set the cost function integrand and Jacobians for the Hypersenstive problem.

```
// Setting the Cost Function
void HyperSensitivePathObject::EvaluateFunctions()
{
```

(continues on next page)

(continued from previous page)

```
    // Extract parameter data
    Rvector stateVec = GetStateVector();
    Rvector controlVec = GetControlVector();

    // Compute intermediate quantities
    Real y = stateVec(0);
    Real ySquared = y * y;
    Real yCubed = ySquared * y;
    Real u = controlVec(0);
    Real uSquared = u * u;

    // Set the cost function integrand
    // J = integral y^2 + u^2
    Rvector costF(1, ySquared + uSquared);
    SetFunctions(COST, costF);
}

// Setting the Cost Function Jacobians
void HyperSensitivePathObject::EvaluateJacobians()
{
    // Computes the user Jacobians
    Rvector stateVec = GetStateVector();
    Rvector controlVec = GetControlVector();
    Real y = stateVec(0);
    Real u = controlVec(0);

    // Cost Function Partials
    Rmatrix costStateJac(1,1, 2.0*y);
    Rmatrix costControlJac(1,1, 2.0*u);
    Rmatrix costTimeJac(1,1, 0.0);
    SetJacobian(COST, STATE, costStateJac); // derivatives of cost with respect to state
    SetJacobian(COST, CONTROL, costControlJac); // derivatives of cost with respect to␣
↪control
    SetJacobian(COST, TIME, costTimeJac); // derivatives of cost with respect to time
}
```

4.10.4 Setting Algebraic Path Constraints

C++ Example

The example below shows how to set algebraic path constraints for the ObstacleAvoidance example problem.

```
// Setting the algebraic path functions
void ObstacleAvoidancePathObject::EvaluateFunctions()
{

    // Extract parameter data and define constants
    Rvector stateVec = GetStateVector();
    Rvector controlVec = GetControlVector();
    Real V = 2.138;
    Real x = stateVec(0);
```

(continues on next page)

(continued from previous page)

```
    Real y = stateVec(1);
    Real theta = controlVec(0);

    // Set the algebraic path constraints
    Real con1 = (x - 0.4)*(x - 0.4) + (y - 0.5)*(y - 0.5);
    Real con2 = (x - 0.8)*(x - 0.8) + (y - 1.5)*(y - 1.5);
    Rvector algFunctions(2, con1, con2);
    Rvector algFuncUpper(2, 100.0, 100.0);
    Rvector algFuncLower(2, 0.1, 0.1);
    SetFunctions(ALGEBRAIC, algFunctions);
    SetFunctionBounds(ALGEBRAIC, UPPER, algFuncUpper);
    SetFunctionBounds(ALGEBRAIC, LOWER, algFuncLower);

}

// Setting the algebraic path function Jacobians
void ObstacleAvoidancePathObject::EvaluateJacobians()
{
    // Extract parameter data and define constants
    Rvector stateVec = GetStateVector();
    Rvector controlVec = GetControlVector();
    Real x = stateVec(0);
    Real y = stateVec(1);
    Real V = 2.138;
    Real theta = controlVec(0);

    // Set the algebraic path function state Jacobians
    Rmatrix algStateJac(2, 2);
    Rmatrix algControlJac(2, 1);
    Rmatrix algTimeJac(2, 1);
    algStateJac(0, 0) = 2 * x - 0.8;
    algStateJac(0, 1) = 2 * y - 1.0;
    algStateJac(1, 0) = 2 * x - 1.6;
    algStateJac(1, 1) = 2 * y - 3.0;
    SetJacobian(ALGEBRAIC, STATE, algStateJac); // derivatives of algebraic constraint␣
↪with respect to state
    SetJacobian(ALGEBRAIC, CONTROL, algControlJac); // derivatives of algebraic␣
↪constraint with respect to control
    SetJacobian(ALGEBRAIC, TIME, algTimeJac); // derivatives of algebraic constraint␣
↪with respect to time
    // Control and state Jacobians are zero
}
```

4.11 ExecutionInterface Reference

The ExecutionInterface class provides an interface to request data from CSALT during optimization/execution. For example, if it is desired to update plots or reports at various stages of optimization, the interface allows the user to obtain information on the optimization process at various stages, such as after initialization, during optimizer iterations, during mesh refinement iterations, and during finalization. Additionally, the ExecutionInterface provides an interrupt interface to stop the optimization process based on user commands.

When implementing an ExecutionInterface for an application, derive an ExecutionInterface class from the CSALT ExecutionInterface base class. The following pure virtual functions must be overloaded:

```
pure virtual Publish(CSALTState)
```

The CSALT states are [Initializing, Optimizing, EvaluatingMesh, ReInitializingMesh, Finalizing].

To obtain the times, states, and controls for a given phase from inside the Publish() function, the following access methods are provided.

```
// Returns the array of states in a requested phase.
Rmatrix GetStateArray(phaseIndex)
// Returns the array of controls in a requested phase
Rmatrix GetControlArray(phaseIndex)
// Returns the array of times of the mesh points in a phase
Rvector GetTimeArray(phaseIndex)
```

During problem setup, provide the custom publisher to the Trajectory object.

```
MyExecutionInterface *myPub = new MyExecutionInterface();
MyTraj *traj = new Trajectory();
traj->SetPublisher(MyExecutionInterfaceb);
```

The CSALT application calls the custom MyExecutionInterface's Publish() method at key execution points, and passes in the CSALT state at each call to allow the user's custom publisher object to provide custom publishing functionality depending upon the CSALT state. Table 4.6 defines the states.

Table 4.6: CSALT states.

CSALT State	Definition
Initializing	The state for all computations and preparations performed to prepare for optimization. The state changes from Initializing to Optimizing immediately before the call to the optimizer.
Optimizing	The state for all computations performed during optimization on a given mesh.
EvaluatingMesh	The state for all computations performed during the evaluation of mesh for accuracy of the solution compared to the user's mesh tolerance.
ReInitializingMesh	The state when CSALT is re-initializing to optimize on a new mesh grid.
Finalizing	The state for all computations performed after both the mesh refinement and optimization have converged.

CHAPTER

FIVE

USER INTERFACE SPECIFICATION

This section describes the GMAT Optimal Control user interface. The user interface specification is given in the tables in this section and are organized by Resource name.

5.1 Trajectory

Table 5.1: User interface specification for Trajectory.GuessSource.

Resource/Command Name	Trajectory
Field Name	GuessSource
Data Type	OptimalControlGuess Resource
Default Value	No default
Allowed Values	Guess Resource
Description	The Guess source for the trajectory. If no guess is provided on a Phase, then the GuessSource from Trajectory is used. For more information see the OptimalControlGuess Resource documentation.
Notes	N/A
Units	N/A
Field Couplings	Phase.GuessSource
Access	set
Interfaces	script

Table 5.2: User interface specification for Trajectory.MajorOptimalityTolerances.

Resource/Command Name	Trajectory
Field Name	MajorOptimalityTolerances
Data Type	Real Array
Default Value	[1e-4]
Allowed Values	All elements must be greater than 0.0
Description	An array of optimality tolerances for optimizer termination during mesh refinement. For iteration i, the optimization tolerance is MajorOptimalityTolerances(i). If `length` (MajorOptimalityTolerances) < mesh refinement iteration number, then the optimality tolerance is `last` (MajorOptimalityTolerances).
Notes	Caution: If `length` (MajorOptimalityTolerances) > MaxMeshRefinementIterations, the last elements of MajorOptimalityTolerances will not be used.
Units	The units are defined by the user's problem scaling.
Field Couplings	MaxRelativeErrorTolerance
Access	set
Interfaces	script

Table 5.3: User interface specification for Trajectory.MajorIterationsLimits.

Resource/Command Name	Trajectory
Field Name	MajorIterationsLimits
Data Type	Integer Array
Default Value	[1000]
Allowed Values	All elements must be greater than or equal to 0.
Description	An array of major iterations limits for optimizer termination during mesh refinement iteration. For iteration i, the major iterations limit is MajorIterationsLimits(i). If `length` (MajorIterationsLimits) < mesh refinement iteration number, then the major iteration limit is `last` (MajorIterationsLimits).
Notes	Caution: If `length` (MajorIterationsLimits) > MaxMeshRefinementIterations, the last elements of MajorIterationsLimits will not be used.
Units	dimensionless
Field Couplings	MaxMeshRefinementIterations
Access	set
Interfaces	script

Table 5.4: User interface specification for Trajectory.TotalIterationsLimits.

Resource/Command Name	Trajectory
Field Name	TotalIterationsLimits
Data Type	Integer Array
Default Value	[20000]
Allowed Values	All elements must be greater than or equal to 0.
Description	An array of total iterations limits for optimizer termination during mesh refinement iteration. For iteration i, the total iterations limit is TotalIterationsLimits(i). If `length` (TotalIterationsLimits) < mesh refinement iteration number, then the total iteration limit is `last` (TotalIterationsLimits). Note, for SNOPT, the total iterations is the sum of the Major (SQP) and minor (QP) iterations.
Notes	Caution: If `length` (TotalIterationsLimits) > MaxMeshRefinementIterations, the last elements of TotalIterationsLimits will not be used.
Units	dimensionless
Field Couplings	MaxMeshRefinementIterations
Access	set
Interfaces	script

Table 5.5: User interface specification for Trajectory.FeasibilityTolerances.

Resource/Command Name	Trajectory
Field Name	FeasibilityTolerances
Data Type	Real Array
Default Value	[1e-6]
Allowed Values	All elements must be greater than 0.0
Description	An array of feasibility tolerances for optimizer termination during mesh refinement iteration. For iteration i, the feasibility tolerance is FeasibilityTolerances(i). If `length` (FeasibilityTolerances) < mesh refinement iteration number, then the feasibility tolerance is `last` (FeasibilityTolerances).
Notes	Caution: If `length` (FeasibilityTolerances) > MaxMeshRefinementIterations, the last elements of FeasibilityTolerances will not be used.
Units	The units are defined by the user's problem scaling.
Field Couplings	This field interacts with Phase.MaxRelativeErrorTol. When trying to achieve a specified MaxRelativeErrorTol for a phase, if the feasiblity tolerance is set too tightly, then the mesh refinement algorithm may not converge.
Access	set
Interfaces	script

Table 5.6: User interface specification for Trajectory.MaxMeshRefinementIterations.

Resource/Command Name	Trajectory
Field Name	MaxMeshRefinementIterations
Data Type	Integer
Default Value	0
Allowed Values	$0 <$ `Integer` $<$ `Inf`
Description	Maximum number of mesh refinement iterations
Notes	Passed through to CSALT Trajectory
Units	N/A
Field Couplings	MajorIterationsLimits TotalIterationsLimits FeasibilityTolerances MajorOptimalityTolerances
Access	set
Interfaces	script

Table 5.7: User interface specification for Trajectory.PhaseList.

Resource/Command Name	Trajectory
Field Name	PhaseList
Data Type	List of Phase Resources
Default Value	Empty List
Allowed Values	A list of user-defined Phase Resources.
Description	A list of user-defined Phase Resources. These phases are included in the optimzation.
Notes	Passed through to CSALT Trajectory
Units	N/A
Field Couplings	None.
Access	set
Interfaces	script

Table 5.8: User interface specification for Trajectory.SNOPTOutputFile.

Resource/Command Name	Trajectory
Field Name	SNOPTOutputFile
Data Type	FileName
Default Value	SNOPTOutputFile.txt
Allowed Values	Valid file name
Description	File containing SNOPT output data.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.9: User interface specification for Trajectory.AddSimpleLinkageChain.

Resource/Command Name	Trajectory
Field Name	AddSimpleLinkageChain
Data Type	List of Phase Resources
Default Value	Empty List
Allowed Values	A list of user-defined Phase Resources. The length of the list be ≥ 2.
Description	Adds a list of phases to be linked with full time and state continuity. Including multiple lines with "AddSimpleLinkageChain" adds multiple simple linkage chain configurations. For example `myTrajectory.AddSimpleLinkageChain` $= \{$`Phase1`, `Phase2`$\}$ `myTrajectory.AddSimpleLinkageChain` $= \{$`Phase3`, `Phase4`, `Phase5`$\}$ results in full time/state continuity between the end `Phase1` and the start of `Phase2`, and full time/state continuity between end of `Phase3` and the start of `Phase4`, and full time/state continuity between end of `Phase4` and the start of `Phase5`.
Notes	N/A
Units	N/A
Field Couplings	Trajectory.PhaseList. Phases in AddSimpleLinkageChain must also be in PhaseList.
Access	set
Interfaces	script

Table 5.10: User interface specification for Trajectory.CustomLinkages.

Resource/Command Name	Trajectory
Field Name	CustomLinkages
Data Type	List of CustomLinkageCosntraints
Default Value	Empty List
Allowed Values	Any CustomLinkageConstraint Resources created.
Description	Adds custom linkage constraints to the trajectory as opposed to the full state/time constraints from the AddSimpleLinkageChain field. For example, a CustomLinkageConstraint object can be added to only restrict the position state. In that case, this field is used to add that constraint to the trajectory.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.11: User interface specification for Trajectory.OutputCoordinateSystem.

Resource/Command Name	Trajectory
Field Name	OutputCoordinateSystem
Data Type	Coordinate System
Default Value	UsePhaseCoordinateSystems
Allowed Values	Any created coordinate system object or the keyword "UsePhaseCoordinateSystems"
Description	Sets the coordinate system for data written to the output optimal control history file. If "UsePhaseCoordinateSystems" is used, the data from each phase is printed in their respective coordinate systems used in their force models.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.12: User interface specification for Trajectory.SolutionFile.

Resource/Command Name	Trajectory
Field Name	SolutionFile
Data Type	String
Default Value	<TrajectoryResourceName>Solution.och
Allowed Values	File name that the output data will be sent to in the GMAT output folder
Description	Used to set the output Optimal Control History file name sent to the GMAT output folder. A new file is created if the current file name is not found
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.13: User interface specification for Trajectory.AllowFailedMeshOptimizations.

Resource/Command Name	Trajectory
Field Name	AllowFailedMeshOptimizations
Data Type	Boolean
Default Value	false
Allowed Values	true, false
Description	Sets whether the optimization should be stopped if convergence is not achieved in any mesh refinement iteration (`FALSE`) or if mesh refinement should be continued regardless of the final result of the previous optimization attempt (`TRUE`).
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.14: User interface specification for Trajectory.StateScaleMode.

Resource/Command Name	Trajectory
Field Name	StateScaleMode
Data Type	String
Default Value	Canonical
Allowed Values	Canonical
Description	The scaling mode for the orbit state. If the Sun is the central body, then the distance unit is an astronomical unit and the GM of the Sun is 1. If the Earth is the central body, then the distance unit is 42000 km and the GM of the Earth is 1. Otherwise, the radius of the central body is the distance unit and the GM of the central body is 1.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.15: User interface specification for Trajectory.MassScaleFactor.

Resource/Command Name	Trajectory
Field Name	MassScaleFactor
Data Type	Real
Default Value	1
Allowed Values	`REAL_MIN` $<$ `Real` $<$ `REAL_MAX`
Description	The mass scale factor (non-dimensional mass is mass/MassScaleFactor). A best-practice is to set the mass scale factor to be approximately the initial total mass of the spacecraft to result in an initial non-dimensionalized mass of approximately 1.0. Note `REAL_MIN` is currently set to 2.2250738585072014e-308.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.16: User interface specification for Trajectory.AddBoundaryFunction.

Resource/Command Name	Trajectory
Field Name	AddBoundaryFunction
Data Type	List of OptimalControlFunction resources
Default Value	Empty List
Allowed Values	Any created OptimalControlFunction resource
Description	A list of user-defined OptimalControlFunction resources to be included in the optmization problem.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.17: User interface specification for Trajectory.PublishUpdateRate.

Resource/Command Name	Trajectory
Field Name	PublishUpdateRate
Data Type	Integer
Default Value	1
Allowed Values	`Integer` > 0
Description	Rate at which trajectory state is output to the publisher. Trajectory data is sent to the publisher (which updates plots and reports) every PublishUpdateRate calls to the cost/constraint functions. The output rate to the Optimal Control History file is independent of PublishUpdateRate.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.18: User interface specification for Trajectory.OptimizationMode.

Resource/Command Name	Trajectory
Field Name	OptimizationMode
Data Type	List of strings
Default Value	{Minimize}
Allowed Values	Feasible point, Minimize, Maximize
Description	The optimization mode (meaning to miminize, maximize, or find a feasible solution). For mesh-refinement iteration i, the optimization mode is OptimizationMode(i). If length(OptimizationMode) < mesh refinement iteration number, then the optimization mode is last(OptimizationMode).
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.19: User interface specification for Trajectory.MeshRefinementGuessMode.

Resource/Command Name	Trajectory
Field Name	MeshRefinementGuessMode
Data Type	String
Default Value	LastSolutionMostRecentMesh
Allowed Values	LastSolutionMostRecentMesh, BestSolutionMostRecentMesh, BestSolutionAnyMesh
Description	The initial-guess source for mesh refinement iterations after the first mesh refinement iteration. LastSolutionMostRecentMesh will use the solution attained by the previous mesh refinement iteration – whether that iteration converged to a feasible solution or not. BestSolutionMostRecentMesh uses the "best" previous solution from only the one immediately previous mesh refinement iteration as the initial guess for the current mesh refinement iteration. BestSolutionAnyMesh uses the "best" previous solution from any previous mesh refinement iteration as an initial guess for the current mesh refinement iteration. The following conditions are used to define "best": The following conditions are used to define "best": (1) If no feasible solutions have been found, then "best" refers to the previous solution with the smallest infeasibility. (2) If one feasible solution has been found, then that solution is "best," regardless of its optimality. (3) If multiple feasible solutions have been found, then the feasible solution with the smallest value of the merit function (i.e., the most optimal solution) is "best."
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.20: User interface specification for Trajectory.StateDimension.

Resource/Command Name	Trajectory
Field Name	StateDimension
Data Type	Integer
Default Value	7
Allowed Values	`Integer` > 0
Description	The number of states defining the problem for the trajectory.
Notes	Default value of 7 is selected because of 6 states defining position and velocity of point mass and 1 state defining mass of point mass. All Phases of a Trajectory default to Trajectory.StateDimension. However, if Phase.StateDimension is set, then the Phase value overrides the Trajectory value for that Phase. Not currently exposed to users because the default value is currently the only acceptable value.
Units	N/A
Field Couplings	Phase.StateDimension
Access	set
Interfaces	script

Table 5.21: User interface specification for Trajectory.ControlDimension.

Resource/Command Name	Trajectory
Field Name	ControlDimension
Data Type	Integer
Default Value	3
Allowed Values	`Integer` > 0
Description	The number of controls defining the problem for the trajectory.
Notes	Default value of 3 is selected for 3D control vector. All Phases of a Trajectory default to Trajectory.ControlDimension. However, if Phase.ControlDimension is set, then the Phase value overrides the Trajectory value for that Phase. Not currently exposed to users because the default value is currently the only acceptable value.
Units	N/A
Field Couplings	Phase.ControlDimension
Access	set
Interfaces	script

5.2 Phase

Table 5.22: User interface specification for Phase.Type.

Resource/Command Name	Phase
Field Name	Type
Data Type	String
Default Value	RadauPseudospectral
Allowed Values	RadauPseudospectral, HermiteSimpson, ImplicitRKOrder4, ImplicitRKOrder6, ImplicitRKOrder8
Description	The transcription algorithm for the phase.
Notes	GMAT instantiates a CSALT phase according to the type chosen here by the user.
Units	N/A
Field Couplings	PointsPerSubPhase, SubPhaseBoundaries
Access	set
Interfaces	script

Table 5.23: User interface specification for Phase.ThrustMode.

Resource/Command Name	Phase
Field Name	ThrustMode
Data Type	String
Default Value	Thrust
Allowed Values	Thrust, Coast
Description	Flag to model the Phase as a thrust or coast phase. In a coast phase, the control magnitude must be zero at all times. In a thrust phase, the control magnitude is allowed to vary. In other words, unless subject to additional constraints, a thrust phase may contain coasting segments.
Notes	N/A
Units	N/A
Field Couplings	DynamicsConfiguration
Access	set
Interfaces	script

Table 5.24: User interface specification for Phase.MaxRelativeErrorTolerance.

Resource/Command Name	Phase
Field Name	MaxRelativeErrorTolerance
Data Type	Real
Default Value	1e-05
Allowed Values	`Real` > 0.0
Description	The maximum allowable relative error in a mesh interval. The mesh refinement algorithm will continue to iterate until all mesh intervals in the phase have a relative error less than MaxRelativeErrorTolerance. The mesh refinment algorithm is different for different transcription types.
Notes	N/A
Units	N/A
Field Couplings	Type
Access	set
Interfaces	script

Table 5.25: User interface specification for Phase.SubPhaseBoundaries.

Resource/Command Name	Phase
Field Name	SubPhaseBoundaries
Data Type	Array of Reals
Default Value	[-1 1]
Allowed Values	Array of real numbers that is monotonically increasing. Allowed values depend on the Type field; see Description for more details.
Description	Defines sub-phases within a phase used as an initial guess for mesh refinement and to support different quadrature orders within a given phase to capture slow and fast dynamics within a single phase. When Type is set to RadauPseudospectral, SubPhaseBoundaries must start with -1 and end with 1, with internal values increasing monotonically. For other transcriptions, SubPhaseBoundaries must start with 0 and end with 1, with internal values increasing monotonically. The default value assumes the default Type = RadauPseudospectral.
Notes	Passed through to CSALT MeshIntervalFractions
Units	N/A
Field Couplings	Type, PointsPerSubPhase
Access	set
Interfaces	script

Table 5.26: User interface specification for Phase.PointsPerSubPhase.

Resource/Command Name	Phase
Field Name	PointsPerSubPhase
Data Type	Array of Integers
Default Value	[5]
Allowed Values	Array of integers that has one less entry than SubPhaseBoundaries. Allowed values depend on the Type field; see Desciption for more details.
Description	Models sub-phases within a phase. For RadauPseudospectral, this allows for different quadrature orders for each sub-phase. The first entry in PointsPerSubPhase is the number of points for the first sub-phase in SubPhaseBoundaries. For example, if SubPhaseBoundaries = [-1 0.5 1], and PointsPerSubPhase = [5 3], then the phase is modeled using two sub-phases. The first subphase is from non-dimensional time -1 to 0.5 and is modeled with 5 points, and the second sub-phase is from non-dimensional time 0.5 to 1.0 and is modeled with 3 points. Note: Mesh refinement will change these values; the values provided are guesses for the discretization. In general, set the number of points as low as possible to obtain convergence on the initial mesh and allow the mesh refinement algorithm to determine a finer mesh. On the other hand, for an ImplictRKOrder* method, the order of the quadrature is fixed. In this case, the value of PointsPerSubPhase sets the number of implicit integration steps. Using the earlier example, if Phase.Type = ImplicitRKOrder6, there would be 5 IRK steps in the first subphase, each of order 6, and 3 IRK steps in the second subphase, each of order 6. The default value assumes the default Type = RadauPseudospectral.
Notes	Passed through to CSALT MeshIntervalNumPoints
Units	N/A
Field Couplings	Type, SubPhaseBoundaries
Access	set
Interfaces	script

Table 5.27: User interface specification for Phase.GuessSource.

Resource/Command Name	Phase
Field Name	GuessSource
Data Type	Guess Resource
Default Value	No default
Allowed Values	OptimalControlGuess Resource
Description	The Guess source for the phase. If no guess is provided, the Trajectory's Guess resource is used.
Notes	N/A
Units	N/A
Field Couplings	Trajectory.Guess
Access	set
Interfaces	script

Table 5.28: User interface specification for Phase.StateLowerBound.

Resource/Command Name	Phase
Field Name	StateLowerBound
Data Type	Array of Reals
Default Value	No default
Allowed Values	Array of real numbers with same length as number of state decision parameters (StateDimension)
Description	The lower bound on state variables at all mesh and stage points within a phase. Bounds must be consistent with the state type and should be specified with respect to the central body of the DynamicsConfiguration with `J2000Eq` axes.
Notes	Passed through to CSALT
Units	Units of the State Vector. Typically [km km km km/s km/s km/s and kg] if using Cartesian state, and modified accordingly for other state types.
Field Couplings	StateDimension, Trajectory.StateDimension
Access	set
Interfaces	script

Table 5.29: User interface specification for Phase.StateUpperBound.

Resource/Command Name	Phase
Field Name	StateUpperBound
Data Type	Array of Reals
Default Value	No default
Allowed Values	Array of real numbers with same length as number of state decision parameters (StateDimension)
Description	The upper bound on state variables at all mesh and stage points within a phase. Bounds must be consistent with the state type and should be specified with respect to the central body of the DynamicsConfiguration with `J2000Eq` axes.
Notes	Passed through to CSALT
Units	Units of the State Vector. Typically [km km km km/s km/s km/s and kg] if using Cartesian state, and modified accordingly for other state types.
Field Couplings	StateDimension, Trajectory.StateDimension
Access	set
Interfaces	script

Table 5.30: User interface specification for Phase.ControlLowerBound.

Resource/Command Name	Phase
Field Name	ControlLowerBound
Data Type	Array of Reals
Default Value	No default
Allowed Values	Array of real numbers with same length as number of control decision parameters (ControlDimension)
Description	The non-dimensional lower bound on control variables at all mesh and stage points within a phase. In standard non-dimensionalization, 0, -1 is minimum control, and +1 is maximum control.
Notes	Passed through to CSALT
Units	non-dimensional
Field Couplings	ControlDimension, Trajectory.ControlDimension
Access	set
Interfaces	script

Table 5.31: User interface specification for Phase.ControlUpperBound.

Resource/Command Name	Phase
Field Name	ControlUpperBound
Data Type	Array of Reals
Default Value	No default
Allowed Values	Array of real numbers with same length as number of control decision parameters (ControlDimension)
Description	The non-dimensional upper bound on control variables at all mesh and stage points within a phase. In standard non-dimensionalization, 0, -1 is minimum control, and +1 is maximum control.
Notes	Passed through to CSALT
Units	non-dimensional
Field Couplings	ControlDimension, Trajectory.ControlDimension
Access	set
Interfaces	script

Table 5.32: User interface specification for Phase.EpochFormat.

Resource/Command Name	Phase
Field Name	EpochFormat
Data Type	String
Default Value	UTCGregorian
Allowed Values	Valid GMAT epoch type. E.g., UTCModJulian, TDB-Gregorian
Description	The epoch format used to specify all epoch fields for the phase.
Notes	N/A
Units	N/A
Field Couplings	InitialEpoch, FinalEpoch, EpochLowerBound, EpochUpperBound
Access	set
Interfaces	script

Table 5.33: User interface specification for Phase.InitialEpoch.

Resource/Command Name	Phase
Field Name	InitialEpoch
Data Type	GMAT Epoch
Default Value	No default
Allowed Values	Valid GMAT Epoch
Description	Initial guess for the initial epoch of the phase. This value is modified during optimization.
Notes	N/A
Units	Time.
Field Couplings	EpochFormat
Access	set
Interfaces	script

Table 5.34: User interface specification for Phase.FinalEpoch.

Resource/Command Name	Phase
Field Name	FinalEpoch
Data Type	GMAT Epoch
Default Value	No default
Allowed Values	Valid GMAT Epoch
Description	Initial guess for the final epoch of the phase. This value is modified during optimization.
Notes	N/A
Units	Time
Field Couplings	EpochFormat
Access	set
Interfaces	script

Table 5.35: User interface specification for Phase.EpochLowerBound.

Resource/Command Name	Phase
Field Name	EpochLowerBound
Data Type	GMAT Epoch
Default Value	No default
Allowed Values	Valid GMAT Epoch
Description	The lower bound on time variables at all mesh and stage points within a phase.
Notes	N/A
Units	Time
Field Couplings	EpochFormat
Access	set
Interfaces	script

Table 5.36: User interface specification for Phase.EpochUpperBound.

Resource/Command Name	Phase
Field Name	EpochUpperBound
Data Type	GMAT Epoch
Default Value	No default
Allowed Values	Valid GMAT Epoch
Description	The upper bound on time variables at all mesh and stage points within a phase.
Notes	N/A
Units	Time
Field Couplings	EpochFormat
Access	set
Interfaces	script

Table 5.37: User interface specification for Phase.DynamicsConfiguration.

Resource/Command Name	Phase
Field Name	DynamicsConfiguration
Data Type	DynamicsConfiguration Resource
Default Value	No default
Allowed Values	Valid DynamicsConfiguration Resource
Description	The grouping of spacecraft, force models, and maneuver models to be used for the phase.
Notes	N/A
Units	N/A
Field Couplings	StateLowerBound,StateUpperBound
Access	set
Interfaces	script

Table 5.38: User interface specification for Phase.MaxControlMagnitude.

Resource/Command Name	Phase
Field Name	MaxControlMagnitude
Data Type	Real
Default Value	1
Allowed Values	$0 <$ `Real` $<$ `Inf`
Description	The maximum magnitude of the non-dimensional control vector. Warning: Setting this value to greater than 1.0 corresponds to control authority greater than what the controller can physically provide and can result in non-physical thrust profiles and trajectories. The upper bound should be set to ≤ 1 except for when using homotopy or troubleshooting.
Notes	N/A
Units	Dimensionless
Field Couplings	None
Access	set
Interfaces	script

Table 5.39: User interface specification for Phase.MinControlMagnitude.

Resource/Command Name	Phase
Field Name	MinControlMagnitude
Data Type	Real
Default Value	0
Allowed Values	$0 <$ `Real` $<$ `Inf`
Description	The minimum magnitude of the non-dimensional control vector. Warning: Setting this value to greater than 1.0 corresponds to control authority greater than what the controller can physically provide and can result in non-physical thrust profiles and trajectories. The upper bound should be set to ≤ 1 except for when using homotopy or troubleshooting.
Notes	N/A
Units	Dimensionless
Field Couplings	None
Access	set
Interfaces	script

Table 5.40: User interface specification for Phase.OverrideColorInGraphics.

Resource/Command Name	Phase
Field Name	OverrideColorInGraphics
Data Type	Boolean
Default Value	false
Allowed Values	true, false
Description	Flag to override the default color defined on the spacecraft and to use the color defined by OrbitColor in graphics during optimization. This field allows phases to appear using different colors in the graphics.
Notes	N/A
Units	N/A
Field Couplings	OrbitColor
Access	set
Interfaces	script

Table 5.41: User interface specification for Phase.BuiltInCost.

Resource/Command Name	Phase
Field Name	BuiltInCost
Data Type	String
Default Value	No default
Allowed Values	RMAGFinal, TotalMassFinal, AbsoluteEpochFinal
Description	The cost function.
Notes	N/A
Units	N/A
Field Couplings	Trajectory.OptimizationMode
Access	set
Interfaces	script

Table 5.42: User interface specification for Phase.BuiltInBoundaryConstraints.

Resource/Command Name	Phase
Field Name	BuiltInBoundaryConstraints
Data Type	StringArray
Default Value	No default
Allowed Values	N/A
Description	Boundary constraints for the phase. These boundary constraints do not support analytic Jacobians.
Notes	This interface is intended to support rapid implementation but is being replaced with assignment-based constraints. As of this writing, there are no constraints implemented via this interface.
Units	N/A
Field Couplings	None
Access	set
Interfaces	script

Table 5.43: User interface specification for Phase.OrbitColor.

Resource/Command Name	Phase
Field Name	OrbitColor
Data Type	ColorType
Default Value	Red
Allowed Values	Valid predefined color name or RGB triplet value between 0 and 255
Description	The color for the phase in graphics if OverrideColorInGraphics is true.
Notes	N/A
Units	N/A
Field Couplings	OverrideColorInGraphics
Access	set
Interfaces	script

Table 5.44: User interface specification for Phase.StateDimension.

Resource/Command Name	Phase
Field Name	StateDimension
Data Type	Integer
Default Value	7
Allowed Values	`Integer` > 0
Description	The number of states defining the problem for the phase.
Notes	Default value of 7 is selected because of 6 states defining position and velocity of point mass and 1 state defining mass of point mass. All Phases of a Trajectory default to Trajectory.StateDimension. However, if Phase.StateDimension is set, then the Phase value overrides the Trajectory value for that Phase. Not currently exposed to users because the default value is currently the only acceptable value.
Units	N/A
Field Couplings	Trajectory.StateDimension
Access	set
Interfaces	script

Table 5.45: User interface specification for Phase.ControlDimension.

Resource/Command Name	Phase
Field Name	ControlDimension
Data Type	Integer
Default Value	3
Allowed Values	`Integer` > 0
Description	The number of control variables defining the problem for the phase.
Notes	Default value of 3 is selected for 3D control vector. All Phases of a Trajectory default to Trajectory.ControlDimension. However, if Phase.ControlDimension is set, then the Phase value overrides the Trajectory value for that Phase. Not currently exposed to users because the default value is currently the only acceptable value.
Units	N/A
Field Couplings	Trajectory.ControlDimension
Access	set
Interfaces	script

5.3 DynamicsConfiguration

Table 5.46: User interface specification for DynamicsConfiguration.ForceModels.

Resource/Command Name	DynamicsConfiguration
Field Name	ForceModels
Data Type	List of Force Model Resources
Default Value	No default
Allowed Values	List of Valid Force Model Resources
Description	Lists the force model for each spacecraft. Currently, only a single spacecraft is supported, so only a single force model should be provided.
Notes	N/A
Units	N/A
Field Couplings	Spacecraft
Access	set
Interfaces	script

Table 5.47: User interface specification for DynamicsConfiguration.Spacecraft.

Resource/Command Name	DynamicsConfiguration
Field Name	Spacecraft
Data Type	List of Spacecraft Resources
Default Value	No default
Allowed Values	List of Valid Spacecraft Resources
Description	Lists the spacecraft that are modeled.
Notes	Currently, only one spacecraft per phase is supported. If there are multiple spacecraft, then the ith spacecraft in the list is the spacecraft to which the ith elements in the DynamicsModels, FiniteBurns, and ImpulsiveBurns lists apply.
Units	N/A
Field Couplings	ForceModels, FiniteBurns, ImpulsiveBurns
Access	set
Interfaces	script

Table 5.48: User interface specification for DynamicsConfiguration.FiniteBurns.

Resource/Command Name	DynamicsConfiguration
Field Name	FiniteBurns
Data Type	List of Finite Burns
Default Value	No default
Allowed Values	List of Finite Burns (including EMTG models, if used)
Description	Lists the finite burn resources for each spacecraft.
Notes	Currently, finite burns MUST be EmtgInterface resources
Units	N/A
Field Couplings	Spacecraft
Access	set
Interfaces	script

Table 5.49: User interface specification for DynamicsConfiguration.ImpulsiveBurns.

Resource/Command Name	DynamicsConfiguration
Field Name	ImpulsiveBurns
Data Type	List of Impulsive Burns
Default Value	No default
Allowed Values	List of Impulsive Burns
Description	Lists the impulsive burn resources for each spacecraft.
Notes	Not currently implemented.
Units	N/A
Field Couplings	Spacecraft
Access	set
Interfaces	script

Table 5.50: User interface specification for DynamicsConfiguration.EMTGTankConfig.

Resource/Command Name	DynamicsConfiguration
Field Name	EMTGTankConfig
Data Type	List of Fuel Tank Hardware
Default Value	No default
Allowed Values	List of valid Fuel Tank Hardware resources
Description	Lists the fuel tanks that are used by the selected spacecraft to represent the depletion of fuel mass along a trajectory on the GMAT side of the GMAT/CSALT interface. The tank(s) selected must be attached to the spacecraft in use and the total mass of the spacecraft must be at least as much as the upper bound on mass in the phase objects using this spacecraft. This field is only required when the DynamicsConfiguration is using an EMTGSpacecraft for a Phase that is not a coast phase.
Notes	
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

5.4 OptimalControlGuess

Table 5.51: User interface specification for OptimalControlGuess.Type.

Resource/Command Name	OptimalControlGuess
Field Name	Type
Data Type	String
Default Value	No default
Allowed Values	GMATArray, CollocationGuessFile
Description	Defines the array guess type: GMATArray-based or file-based.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.52: User interface specification for OptimalControlGuess.TimeSystem.

Resource/Command Name	OptimalControlGuess
Field Name	TimeSystem
Data Type	String
Default Value	No default
Allowed Values	Any GMAT Modified Julian Time System
Description	The time system used in the data array specified by ArrayName. Only used if Type = GMATArray.
Notes	N/A
Units	N/A
Field Couplings	Type
Access	set
Interfaces	script

Table 5.53: User interface specification for OptimalControlGuess.CoordinateSystem.

Resource/Command Name	OptimalControlGuess
Field Name	CoordinateSystem
Data Type	CoordinateSystemResource
Default Value	EarthMJ2000Eq
Allowed Values	CoordinateSystem Resource
Description	The coordinate system used in the data array specified by ArrayName. Only used if Type = GMATArray.
Notes	N/A
Units	N/A
Field Couplings	Type
Access	set
Interfaces	script

Table 5.54: User interface specification for OptimalControlGuess.GuessArray.

Resource/Command Name	OptimalControlGuess
Field Name	GuessArray
Data Type	Array of Reals
Default Value	No default
Allowed Values	Each element of the array must be $-\texttt{Inf} < \texttt{Real} < \texttt{Inf}$
Description	The array containing the guess data.
Notes	The columns of the array are: time, states, controls. The rows are the different values of those variables. The times must increase monotonically as the row number increases. The array must have at least 5 rows.
Units	N/A
Field Couplings	Type
Access	set
Interfaces	script

Table 5.55: User interface specification for OptimalControlGuess.FileName.

Resource/Command Name	OptimalControlGuess
Field Name	FileName
Data Type	File name and path.
Default Value	No default
Allowed Values	File consistent with Type defined in Type field.
Description	The file containing the guess data, formatted as an Optimal Control History file, relative to the directory in which the GMAT executable is located.
Notes	N/A
Units	N/A
Field Couplings	Type
Access	set
Interfaces	script

5.5 CustomLinkageConstraint

Table 5.56: User interface specification for CustomLinkageConstraint.ConstraintMode.

Resource/Command Name	CustomLinkageConstraint
Field Name	ConstraintMode
Data Type	String
Default Value	N/A
Allowed Values	Difference, Absolute
Description	Applies a constraint on the difference between parameters (specifed in using SetModelParameter) for two phases (Difference mode) or applies an absolute constraint on quantities for a single phase specified in the InitialPhase field (Absolute mode).
Notes	None
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.57: User interface specification for CustomLinkageConstraint.InitialPhase.

Resource/Command Name	CustomLinkageConstraint
Field Name	InitialPhase
Data Type	String
Default Value	N/A
Allowed Values	Phase resource
Description	The first phase in the linkage for a Difference-mode constraint. The only phase in the linkage for an Absolute-mode constraint.
Notes	None
Units	N/A
Field Couplings	ConstraintMode
Access	set
Interfaces	script

Table 5.58: User interface specification for CustomLinkageConstraint.InitialPhaseBoundaryType.

Resource/Command Name	CustomLinkageConstraint
Field Name	InitialPhaseBoundaryType
Data Type	String
Default Value	N/A
Allowed Values	Start, End
Description	The boundary of InitialPhase at which the constraint is calculated.
Notes	None
Units	N/A
Field Couplings	ConstraintMode
Access	set
Interfaces	script

Table 5.59: User interface specification for CustomLinkageConstraint.FinalPhase.

Resource/Command Name	CustomLinkageConstraint
Field Name	FinalPhase
Data Type	String
Default Value	N/A
Allowed Values	Phase resource
Description	The second phase in the linkage for a Difference-mode constraint.
Notes	None
Units	N/A
Field Couplings	ConstraintMode
Access	set
Interfaces	script

Table 5.60: User interface specification for CustomLinkageConstraint.FinalPhaseBoundaryType.

Resource/Command Name	CustomLinkageConstraint
Field Name	FinalPhaseBoundaryType
Data Type	String
Default Value	N/A
Allowed Values	Start, End
Description	The boundary of FinalPhase at which the constraint is calculated.
Notes	None
Units	N/A
Field Couplings	ConstraintMode
Access	set
Interfaces	script

Table 5.61: User interface specification for CustomLinkageConstraint.SetModelParameter().

Resource/Command Name	CustomLinkageConstraint
Field Name	SetModelParameter()
Data Type	Tuple
Default Value	N/A
Allowed Values	See Table 4.4
Description	SetModelParameter() is an overloaded function for setting model data. In general, the argument is a tuple. The first element of the tuple is a string describing the parameter to be set, and the second element of the tuple is the value of the parameter.
Notes	None
Units	See Table 4.4
Field Couplings	ConstraintMode
Access	set
Interfaces	script

5.6 EMTGSpacecraft

Table 5.62: User interface specification for EMTGSpacecraft.SpacecraftFile.

Resource/Command Name	EMTGSpacecraft
Field Name	SpacecraftFile
Data Type	String
Default Value	N/A
Allowed Values	Full file path (can be relative to ../data/emtg) AND name of *.emtg_spacecraftopt file
Description	A valid EMTG spacecraft file
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.63: User interface specification for EMTGSpacecraft.SpacecraftStage.

Resource/Command Name	EMTGSpacecraft
Field Name	SpacecraftStage
Data Type	IntegerArray
Default Value	[1]
Allowed Values	Integers that are less than or equal to the number of stages contained in the EMTG options file that is set using the "EMTGSpacecraftFile" field on EMTGInterface Resource
Description	The EMTG spacecraft stage for each phase, where each position in the array corresponds to the phase index on the trajectory. (I.e., the second element in this array corresponds to the second phase's stage). If there are more phases than listed stages that use this EmtgInterface object, the phases after the last stage will continue to use the last stage in the list. The elements of EMTGSpacecraftStage are 1-indexed.
Notes	The array of stages does NOT skip coast phases. I.e., if the phase structure is coast-thrust-thrust, then [2,1] will result in stage 2 applying to the coast phase (and not being used) and stage 1 being applied to both thrust phases. [2,2,1] results in the first thrust phase using stage 2 and the second thrust phase using stage 1. If the length of the array is longer than the number of phases, then trailing elements of the array are not used. I.e., if there are i phases, then only the first i elements of EMTGSpacecraftStage are used.
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.64: User interface specification for EMTGSpacecraft.DutyCycle.

Resource/Command Name	EMTGSpacecraft
Field Name	DutyCycle
Data Type	Real
Default Value	1
Allowed Values	$0 \leq$ `Real` ≤ 1
Description	Field to set the averaged duty cycle factor of the propulsion system.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

5.7 OptimalControlFunction

Table 5.65: User interface specification for OptimalControlFunction.Function.

Resource/Command Name	OptimalControlFunction
Field Name	Function
Data Type	String
Default Value	Expression
Allowed Values	"Expression", "PatchedConicLaunch", "IntegratedFlyby", "PatchedConicFlyby", "CelestialBodyRendezvous"
Description	Sets type of function of the OptimalControlFunction
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

5.8 OptimalControlFunction for Function Field = Expression

Table 5.66: User interface specification for OptimalControlFunction.Type for OptimalControlFunction.Function = Expression.

Resource/Command Name	OptimalControlFunction:Expression
Field Name	Type
Data Type	String
Default Value	AlgebraicConstraint
Allowed Values	AlgebraicConstraint
Description	Sets the "meaning" of the OptimalControlFunction. I.e., is the function to be interpreted as a constraint or a cost function, etc.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.67: User interface specification for OptimalControlFunction.PhaseList for OptimalControlFunction.Function = Expression.

Resource/Command Name	OptimalControlFunction:Expression
Field Name	PhaseList
Data Type	GMAT List
Default Value	No default
Allowed Values	Name of a single existing Phase resource and either Initial or Final. E.g., `PhaseList` = {`Phase1.Initial`} or `PhaseList` = {`Phase2.Final`}.
Description	Sets on which phase and at which boundary of that phase the OptimalControlFunction Expression is to be evaluated.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.68: User interface specification for OptimalControlFunction.SetModelParameter for OptimalControlFunction.Function = Expression.

Resource/Command Name	OptimalControlFunction:Expression
Field Name	SetModelParameter
Data Type	Tuple
Default Value	No default
Allowed Values	('LowerBounds', <Real>); ('UpperBounds', <Real>); ('ScaleFactor', <Real>); ('Expression', <GMAT Equation as a string>)
Description	Used to set parameters of an OptimalControlFunction. If OptimalControlFunction.Type = AlgebraicConstraint, then the tuple can be used to set the bounds on the constraint ('LowerBounds' or 'UpperBounds'); to set the scale factor for the constraint ('ScaleFactor'); and to set the constrained expression itself ('Expression').
Notes	N/A
Units	Default GMAT units for the expression being set
Field Couplings	Type, Function
Access	set
Interfaces	script

5.9 OptimalControlFunction for Function Field = PatchedConicLaunch

Table 5.69: User interface specification for OptimalControlFunction.Type for OptimalControlFunction.Function = PatchedConicLaunch.

Resource/Command Name	OptimalControlFunction:PatchedConicLaunch
Field Name	Type
Data Type	String
Default Value	AlgebraicConstraint
Allowed Values	AlgebraicConstraint
Description	When the user instantiates a PatchedConicLaunch OptimalControlFunction, it must be of type AlgebraicConstraint.
Notes	N/A
Units	N/A
Field Couplings	Function
Access	set
Interfaces	script

Table 5.70: User interface specification for OptimalControlFunction.PhaseList for OptimalControlFunction.Function = PatchedConicLaunch.

Resource/Command Name	OptimalControlFunction:PatchedConicLaunch
Field Name	PhaseList
Data Type	GMAT List
Default Value	Empty list
Allowed Values	Name of a single existing Phase resource and either Initial or Final. E.g., `PhaseList` = {`Phase1.Initial`} or `PhaseList` = {`Phase2.Final`}.
Description	Sets on which phase and at which boundary of that phase the PatchedConicLaunch is to be evaluated.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.71: User interface specification for OptimalControlFunction.SetModelParameter('CentralBody', 'body name') for OptimalControlFunction.Function = PatchedConicLaunch.

Resource/Command Name	OptimalControlFunction:PatchedConicLaunch
Field Name	SetModelParameter('CentralBody', 'body name')
Data Type	Tuple
Default Value	N/A
Allowed Values	The first element of the tuple must be a string: CentralBody. The second element of the tuple is a string whose value is a celestial body that an be interpreted by GMAT and whose ephemeris is accessible.
Description	Sets the central body of the patched-conic launch.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.72: User interface specification for OptimalControlFunction.SetModelParameter('EMTGLaunchVehicleOptionsFile', 'launch vehicle options file') for OptimalControlFunction.Function = PatchedConicLaunch.

Resource/Command Name	OptimalControlFunction:PatchedConicLaunch
Field Name	SetModelParameter('EMTGLaunchVehicleOptionsFile', 'launch vehicle options file')
Data Type	Tuple
Default Value	N/A
Allowed Values	The first element of the tuple must be a string: EMTGLaunchVehicleOptionsFile. The second element of the tuple is a string containing the name (including path relative to the GMAT executable directory) of an EMTG launch vehicle options file.
Description	Sets the EMTG launch vehicle file in which the launch vehicle characteristics are described.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.73: User interface specification for OptimalControlFunction.SetModelParameter('VehicleName', 'name of launch vehicle') for OptimalControlFunction.Function = PatchedConicLaunch.

Resource/Command Name	OptimalControlFunction:PatchedConicLaunch
Field Name	SetModelParameter('VehicleName', 'name of launch vehicle')
Data Type	Tuple
Default Value	N/A
Allowed Values	The first element of the tuple must be a string: VehicleName. The second element of the tuple is the name of a launch vehicle described in the specified EMTG launch vehicle options file.
Description	Sets the specific name of the launch vehicle within the provided EMTG launch vehicle options file.
Notes	N/A
Units	N/A
Field Couplings	SetModelParameter('EMTGLaunchVehicleOptionsFile', 'launch vehicle options file')
Access	set
Interfaces	script

5.10 OptimalControlFunction for Function Field = CelestialBodyRendezvous

Table 5.74: User interface specification for OptimalControlFunction.Type for OptimalControlFunction.Function = CelestialBodyRendezvous.

Resource/Command Name	OptimalControlFunction:CelestialBodyRendezvous
Field Name	Type
Data Type	String
Default Value	AlgebraicConstraint
Allowed Values	AlgebraicConstraint
Description	When the user instantiates a CelestialBodyRendezvous OptimalControlFunction, it must be of type AlgebraicConstraint.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.75: User interface specification for OptimalControlFunction.PhaseList for OptimalControlFunction.Function = CelestialBodyRendezvous.

Resource/Command Name	OptimalControlFunction:CelestialBodyRendezvous
Field Name	PhaseList
Data Type	GMAT List
Default Value	Empty list
Allowed Values	Name of a single existing Phase resource and either Initial or Final. E.g., `PhaseList` = {`Phase1.Initial`} or `PhaseList` = {`Phase2.Final`}.
Description	Sets on which phase and at which boundary of that phase the CelestialBodyRendezvous is to be evaluated.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.76: User interface specification for OptimalControlFunction.SetModelParameter('CelestialBody', 'body name') for OptimalControlFunction.Function = CelestialBodyRendezvous.

Resource/Command Name	OptimalControlFunction:CelestialBodyRendezvous
Field Name	SetModelParameter('CelestialBody', 'body name')
Data Type	Tuple
Default Value	N/A
Allowed Values	The first element of the tuple must be a string: CelestialBody. The second element of the tuple is a string whose value is a celestial body that an be interpreted by GMAT and whose ephemeris is accessible.
Description	Sets the celestial body with which the spacecraft is to rendezvous.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

5.11 OptimalControlFunction for Function Field = IntegratedFlyby

Table 5.77: User interface specification for OptimalControlFunction.Type for OptimalControlFunction.Function = IntegratedFlyby.

Resource/Command Name	OptimalControlFunction:IntegratedFlyby
Field Name	Type
Data Type	String
Default Value	AlgebraicConstraint
Allowed Values	AlgebraicConstraint
Description	When the user instantiates an IntegratedFlyby OptimalControlFunction, it must be of type AlgebraicConstraint.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.78: User interface specification for OptimalControlFunction.PhaseList for OptimalControlFunction.Function = IntegratedFlyby.

Resource/Command Name	OptimalControlFunction:IntegratedFlyby
Field Name	PhaseList
Data Type	GMAT List
Default Value	empty list
Allowed Values	List must consist of exactly 2 existing Phase resources and either Initial and Final on both. E.g., `PhaseList` = {`Phase1.Final, Phase2.Initial`}.
Description	Sets the phase preceeding the flyby event and the phase following the flyby event.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.79: User interface specification for OptimalControlFunction.SetModelParameter('CelestialBody', 'body name') for OptimalControlFunction.Function = IntegratedFlyby.

Resource/Command Name	OptimalControlFunction:IntegratedFlyby
Field Name	SetModelParameter('CelestialBody', 'body name')
Data Type	Tuple
Default Value	N/A
Allowed Values	The first element of the tuple must be a string: CelestialBody. The second element of the tuple is a string whose value is a celestial body that an be interpreted by GMAT and whose ephemeris is accessible.
Description	Sets the celestial body that is the central body of the integrated flyby.
Notes	N/A
Units	N/A
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.80: User interface specification for OptimalControlFunction.SetModelParameter('PeriapsisRadiusLowerBound', value) for OptimalControlFunction.Function = IntegratedFlyby.

Resource/Command Name	OptimalControlFunction:IntegratedFlyby
Field Name	SetModelParameter('PeriapsisRadiusLowerBound', value)
Data Type	Tuple
Default Value	6978
Allowed Values	First element of tuple must be a string: PeriapsisRadiusLowerBound. The second element of the tuple is a $0 <$ `Real` $<$ `Inf`.
Description	Lower bound of periapse of flyby with respect to the central body's center of mass.
Notes	N/A
Units	km
Field Couplings	N/A
Access	set
Interfaces	script

Table 5.81: User interface specification for OptimalControlFunction.SetModelParameter('PeriapsisRadiusUpperBound', value) for OptimalControlFunction.Function = IntegratedFlyby.

Resource/Command Name	OptimalControlFunction:IntegratedFlyby
Field Name	SetModelParameter('PeriapsisRadiusUpperBound', value)
Data Type	Tuple
Default Value	1000000
Allowed Values	First element of tuple must be a string: PeriapsisRadiusUpperBound. The second element of the tuple is a $0 <$ `Real` $<$ `Inf`.
Description	Upper bound of periapse of flyby with respect to the central body's center of mass.
Notes	N/A
Units	km
Field Couplings	N/A
Access	set
Interfaces	script

CHAPTER

SIX

EMTG SPACECRAFT FILE SPECIFICATION

This section describes the contents of an EMTG spacecraft file. The descriptions are given in the tables in this section and are organized by the spacecraft file block, the line number within a block, and the variable number within a line. It is highly recommended that this file specification be used in conjunction with *Format of EMTG Spacecraft File* and the example .emtg_spacecraft files provided in gmat/data/emtg/.

Several symbols and acronyms are used in the tables in this section: P is power; P_0 is base power delivered by a solar array; r_s is the distance from the spacecraft to the Sun; Isp is specific impulse; CSI is constant specific impulse; and VSI is variable specific impulse.

6.1 Spacecraft Block

This section describes the contents of the Spacecraft Block of an EMTG spacecraft file.

Table 6.1: EMTG spacecraft file specification for Spacecraft Block, line 1, entry 1.

Line number	1
Line name	name
Variable number	1
Variable description	N/A
Data type	string
Units	N/A
Value restrictions	none
Other	N/A

Table 6.2: EMTG spacecraft file specification for Spacecraft Block, line 2, entry 1.

Line number	2
Line name	EnableGlobalElectricPropellantTankConstraint
Variable number	1
Variable description	N/A
Data type	integer
Units	N/A
Value restrictions	0 or 1
Other	boolean integer

Table 6.3: EMTG spacecraft file specification for Spacecraft Block, line 3, entry 1.

Line number	3
Line name	EnableGlobalChemicalPropellantTankConstraint
Variable number	1
Variable description	N/A
Data type	integer
Units	N/A
Value restrictions	0 or 1
Other	boolean integer

Table 6.4: EMTG spacecraft file specification for Spacecraft Block, line 4, entry 1.

Line number	4
Line name	EnableGlobalDryMassConstraint
Variable number	1
Variable description	N/A
Data type	integer
Units	N/A
Value restrictions	0 or 1
Other	boolean integer

Table 6.5: EMTG spacecraft file specification for Spacecraft Block, line 5, entry 1.

Line number	5
Line name	GlobalElectricPropellantTankCapacity
Variable number	1
Variable description	N/A
Data type	real
Units	kg
Value restrictions	≥ 0
Other	N/A

Table 6.6: EMTG spacecraft file specification for Spacecraft Block, line 6, entry 1.

Line number	6
Line name	GlobalFuelTankCapacity
Variable number	1
Variable description	N/A
Data type	real
Units	kg
Value restrictions	≥ 0
Other	N/A

Table 6.7: EMTG spacecraft file specification for Spacecraft Block, line 7, entry 1.

Line number	7
Line name	GlobalOxidizerTankCapacity
Variable number	1
Variable description	N/A
Data type	real
Units	kg
Value restrictions	≥ 0
Other	N/A

Table 6.8: EMTG spacecraft file specification for Spacecraft Block, line 8, entry 1.

Line number	8
Line name	GlobalDryMassBounds
Variable number	1
Variable description	Lower bound on global dry mass
Data type	real
Units	kg
Value restrictions	≥ 0; $\leq$ GlobalDryMassBounds[2]
Other	N/A

Table 6.9: EMTG spacecraft file specification for Spacecraft Block, line 8, entry 2.

Line number	8
Line name	GlobalDryMassBounds
Variable number	2
Variable description	Upper bound on global dry mass
Data type	real
Units	kg
Value restrictions	$\geq$ GlobalDryMassBounds[1]
Other	N/A

6.2 Stage Block

This section describes the contents of the Stage Block of an EMTG spacecraft file.

Table 6.10: EMTG spacecraft file specification for Stage Block, line 1, entry 1.

Line number	1
Line name	name
Variable number	1
Variable description	Name of the stage
Data type	string
Units	N/A
Value restrictions	none
Other	N/A

Table 6.11: EMTG spacecraft file specification for Stage Block, line 2, entry 1.

Line number	2
Line name	BaseDryMass
Variable number	1
Variable description	Dry mass of the propulsion system base
Data type	real
Units	kg
Value restrictions	≥ 0
Other	N/A

Table 6.12: EMTG spacecraft file specification for Stage Block, line 3, entry 1.

Line number	3
Line name	AdapterMass
Variable number	1
Variable description	Mass of the spacecraft-to-power-system adapter
Data type	real
Units	kg
Value restrictions	≥ 0
Other	N/A

Table 6.13: EMTG spacecraft file specification for Stage Block, line 4, entry 1.

Line number	4
Line name	EnableElectricPropellantTankConstraint
Variable number	1
Variable description	Is the constraint on the electric propellant tank capacity enabled?
Data type	integer
Units	N/A
Value restrictions	0 or 1
Other	boolean integer

Table 6.14: EMTG spacecraft file specification for Stage Block, line 5, entry 1.

Line number	5
Line name	EnableChemicalPropellantTankConstraint
Variable number	1
Variable description	Is the constraint on the chemical propellant tank capacity enabled?
Data type	integer
Units	N/A
Value restrictions	0 or 1
Other	boolean integer

Table 6.15: EMTG spacecraft file specification for Stage Block, line 6, entry 1.

Line number	6
Line name	EnableDryMassConstraint
Variable number	1
Variable description	Is the constraint on dry mass enabled?
Data type	integer
Units	N/A
Value restrictions	0 or 1
Other	boolean integer

Table 6.16: EMTG spacecraft file specification for Stage Block, line 7, entry 1.

Line number	7
Line name	ElectricPropellantTankCapacity
Variable number	1
Variable description	Maximum electric propellant mass
Data type	real
Units	kg
Value restrictions	≥ 0
Other	Only enforced if EnableElectricPropellantTankConstraint == 1

Table 6.17: EMTG spacecraft file specification for Stage Block, line 8, entry 1.

Line number	8
Line name	ChemicalFuelTankCapacity
Variable number	1
Variable description	Maximum chemical fuel mass
Data type	real
Units	kg
Value restrictions	≥ 0
Other	Only enforced if EnableChemicalPropellantTankConstraint == 1

Table 6.18: EMTG spacecraft file specification for Stage Block, line 9, entry 1.

Line number	9
Line name	ChemicalOxidizerTankCapacity
Variable number	1
Variable description	Maximum oxidizer mass
Data type	real
Units	kg
Value restrictions	≥ 0
Other	Only enforced if EnableChemicalPropellantTankConstraint == 1

Table 6.19: EMTG spacecraft file specification for Stage Block, line 10, entry 1.

Line number	10
Line name	ThrottleLogic
Variable number	1
Variable description	How should the number of thrusters be selected?
Data type	integer
Units	N/A
Value restrictions	1 or 2
Other	1: minimum number of thrusters 2: maximum number of thrusters

Table 6.20: EMTG spacecraft file specification for Stage Block, line 11, entry 1.

Line number	11
Line name	ThrottleSharpness
Variable number	1
Variable description	Determines how smooth or sharp the transition between throttle levels is
Data type	real
Units	N/A
Value restrictions	> 0
Other	The Heaviside step function is analytically approximated using $H = \frac{1}{1-\exp(-2kx)}$, $k =$ ThrottleSharpness.

Table 6.21: EMTG spacecraft file specification for Stage Block, line 12, entry 1.

Line number	12
Line name	PowerSystem
Variable number	1
Variable description	Name of power system for stage
Data type	string
Units	N/A
Value restrictions	Must be a valid name in the stage's Power Library block
Other	N/A

Table 6.22: EMTG spacecraft file specification for Stage Block, line 13, entry 1.

Line number	13
Line name	ElectricPropulsionSystem
Variable number	1
Variable description	Name of electric propulsion system for stage
Data type	string
Units	N/A
Value restrictions	Must be a valid name in the stage's Propulsion Library block
Other	N/A

Table 6.23: EMTG spacecraft file specification for Stage Block, line 14, entry 1.

Line number	14
Line name	ChemicalPropulsionSystem
Variable number	1
Variable description	Name of chemical propulsion system for stage
Data type	string
Units	N/A
Value restrictions	Must be a valid name in the stage's Propulsion Library block
Other	N/A

6.3 Power Library Block

This section describes the contents of the Power Library Block of an EMTG spacecraft file.

Multiple power systems may be added to the power library by adding additional lines that contain all elements specified in line 1.

Table 6.24: EMTG spacecraft file specification for Power Library Block, line 1, entry 1.

Line number	1
Line name	N/A
Variable number	1
Variable description	name for this power system being specified by that line
Data type	string
Units	N/A
Value restrictions	none
Other	N/A

Table 6.25: EMTG spacecraft file specification for Power Library Block, line 1, entry 2.

Line number	1
Line name	N/A
Variable number	2
Variable description	power supply type
Data type	integer
Units	N/A
Value restrictions	[0, 1]
Other	0: constant; 1: solar

Table 6.26: EMTG spacecraft file specification for Power Library Block, line 1, entry 3.

Line number	1
Line name	N/A
Variable number	3
Variable description	power supply curve type
Data type	integer
Units	N/A
Value restrictions	[0, 1]
Other	0: Sauer: $P_{Generated} = \frac{P_0}{r_s^2}\left(\frac{\gamma_0+\gamma_1/r_s+\gamma_2/r_s^2}{1+\gamma_3 r_s+\gamma_4 r_s^2}\right)$ 1: polynomial: $P_{Generated} = \frac{P_0}{r_s^2}\sum_{i=0}^{6} r_s^i$

Table 6.27: EMTG spacecraft file specification for Power Library Block, line 1, entry 4.

Line number	1
Line name	N/A
Variable number	4
Variable description	bus power type
Data type	integer
Units	N/A
Value restrictions	[0, 1]
Other	0: type A quadratic: P_Bus = bus_power[0] + bus_power[1] / r_s^1 + bus_power[2] / r_s^2 1: type B conditional: If P_Provided > bus_power[0], then P_Bus = bus_power[0]. Else, P_Bus = bus_power[0] + bus_power[1] * (bus_power[2] - P_provided)

Table 6.28: EMTG spacecraft file specification for Power Library Block, line 1, entry 5.

Line number	1
Line name	N/A
Variable number	5
Variable description	P0
Data type	real
Units	kW
Value restrictions	≥ 0
Other	Base power

Table 6.29: EMTG spacecraft file specification for Power Library Block, line 1, entry 6.

Line number	1
Line name	N/A
Variable number	6
Variable description	mass per kW
Data type	real
Units	kg
Value restrictions	≥ 0
Other	N/A

Table 6.30: EMTG spacecraft file specification for Power Library Block, line 1, entry 7.

Line number	1
Line name	N/A
Variable number	7
Variable description	decay rate
Data type	real
Units	1/years
Value restrictions	≥ 0
Other	rate of power decay over time: $P_{generated} = P_{generated} \exp(-\texttt{decay\_rate} \cdot t)$, with t in years

Table 6.31: EMTG spacecraft file specification for Power Library Block, line 1, entry 8.

Line number	1
Line name	N/A
Variable number	8
Variable description	gamma[0]
Data type	real
Units	assumes r_s is in km and P is in kW
Value restrictions	N/A
Other	Power delivered by solar array. If power supply curve type == 0: numerator coefficient of r_s^0. If power supply curve type == 1: coefficient of r_s^{-2}

Table 6.32: EMTG spacecraft file specification for Power Library Block, line 1, entry 9.

Line number	1
Line name	N/A
Variable number	9
Variable description	gamma[1]
Data type	real
Units	assumes r_s is in au and P is in kW
Value restrictions	N/A
Other	Power delivered by solar array. If power supply curve type == 0: numerator coefficient of r_s^1. If power supply curve type == 1: coefficient of r_s^{-1}

Table 6.33: EMTG spacecraft file specification for Power Library Block, line 1, entry 10.

Line number	1
Line name	N/A
Variable number	10
Variable description	gamma[2]
Data type	real
Units	assumes r_s is in au and P is in kW
Value restrictions	N/A
Other	Power delivered by solar array. If power supply curve type == 0: numerator coefficient of r_s^2. If power supply curve type == 1: coefficient of r_s^0

Table 6.34: EMTG spacecraft file specification for Power Library Block, line 1, entry 11.

Line number	1
Line name	N/A
Variable number	11
Variable description	gamma[3]
Data type	real
Units	assumes r_s is in au and P is in kW
Value restrictions	N/A
Other	Power delivered by solar array. If power supply curve type == 0: denominator coefficient of r_s^1. If power supply curve type == 1: coefficient of r_s^1

Table 6.35: EMTG spacecraft file specification for Power Library Block, line 1, entry 12.

Line number	1
Line name	N/A
Variable number	12
Variable description	gamma[4]
Data type	real
Units	assumes r_s is in au and P is in kW
Value restrictions	N/A
Other	Power delivered by solar array. If power supply curve type == 0: denominator coefficient of r_s^2. If power supply curve type == 1: coefficient of r_s^2

Table 6.36: EMTG spacecraft file specification for Power Library Block, line 1, entry 13.

Line number	1
Line name	N/A
Variable number	13
Variable description	gamma[5]
Data type	real
Units	assumes r_s is in au and P is in kW
Value restrictions	N/A
Other	Power delivered by solar array. If power supply curve type == 1: coefficient of r_s^3

Table 6.37: EMTG spacecraft file specification for Power Library Block, line 1, entry 14.

Line number	1
Line name	N/A
Variable number	14
Variable description	gamma[6]
Data type	real
Units	assumes r_s is in au and P is in kW
Value restrictions	N/A
Other	Power delivered by solar array. If power supply curve type == 1: coefficient of r_s^4

Table 6.38: EMTG spacecraft file specification for Power Library Block, line 1, entry 15.

Line number	1
Line name	N/A
Variable number	15
Variable description	bus power[0]
Data type	real
Units	N/A
Value restrictions	N/A
Other	If bus power type == 0: Power required by spacecraft bus: coefficient of r_s^0. If bus power type == 1, see note on bus power type.

Table 6.39: EMTG spacecraft file specification for Power Library Block, line 1, entry 16.

Line number	1
Line name	N/A
Variable number	16
Variable description	bus power[1]
Data type	real
Units	N/A
Value restrictions	N/A
Other	If bus power type == 0: Power required by spacecraft bus: coefficient of r_s^1. If bus power type == 1, see note on bus power type.

Table 6.40: EMTG spacecraft file specification for Power Library Block, line 1, entry 17.

Line number	1
Line name	N/A
Variable number	17
Variable description	bus power[2]
Data type	real
Units	N/A
Value restrictions	N/A
Other	If bus power type == 0: Power required by spacecraft bus: coefficient of r_s^2. If bus power type == 1, see note on bus power type.

6.4 Propulsion Library Block

This section describes the contents of the Propulsion Library Block of an EMTG spacecraft file.

Multiple propulsion systems may be added to the propulsion library by adding additional lines that contain all elements specified in line 1.

Table 6.41: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 1.

Line number	1
Line name	N/A
Variable number	1
Variable description	name for this power system being specified by that line
Data type	string
Units	N/A
Value restrictions	none
Other	N/A

Table 6.42: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 2.

Line number	1
Line name	N/A
Variable number	2
Variable description	thruster mode
Data type	integer
Units	N/A
Value restrictions	[0,1,3,4,5]
Other	0: constant thrust and Isp 1: fixed efficiency CSI 3: polynomial 1D 4: stepped H thrust 1D 5: stepped L mdot 1D.

Table 6.43: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 3.

Line number	1
Line name	N/A
Variable number	3
Variable description	throttle table file
Data type	string
Units	N/A
Value restrictions	"none" or path to external *.ThrottleTable file, relative to GMAT executable location
Other	If an external throttle table is to be used, the relevant file is specified here. Otherwise, use "none". If an external table is provided, it overrides local thrust coefficients and power coefficients

Table 6.44: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 4.

Line number	1
Line name	N/A
Variable number	4
Variable description	mass per string
Data type	real
Units	kg
Value restrictions	≥ 0
Other	mass of a single thruster string

Table 6.45: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 5.

Line number	1
Line name	N/A
Variable number	5
Variable description	number of strings
Data type	integer
Units	N/A
Value restrictions	≥ 1
Other	the number of thruster strings in operation

Table 6.46: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 6.

Line number	1
Line name	N/A
Variable number	6
Variable description	Minimum power
Data type	real
Units	kW
Value restrictions	≥ 0
Other	Minimum power to thruster

Table 6.47: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 7.

Line number	1
Line name	N/A
Variable number	7
Variable description	Maximum power
Data type	real
Units	kW
Value restrictions	$\geq$ Minimum power (variable 6)
Other	Maximum power to thruster

Table 6.48: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 8.

Line number	1
Line name	N/A
Variable number	8
Variable description	constrant thrust
Data type	real
Units	N
Value restrictions	≥ 0
Other	If a constant thrust is desired, its value is set here

Table 6.49: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 9.

Line number	1
Line name	N/A
Variable number	9
Variable description	constant Isp
Data type	real
Units	s
Value restrictions	≥ 0
Other	If a constant Isp is desired, its value is set here

Table 6.50: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 10.

Line number	1
Line name	N/A
Variable number	10
Variable description	minimum Isp / monoprop Isp
Data type	real
Units	s
Value restrictions	≥ 0
Other	Isp for a monoprop system, if used

Table 6.51: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 11.

Line number	1
Line name	N/A
Variable number	11
Variable description	fixed efficiency
Data type	real
Units	N/A
Value restrictions	$0 \leq \texttt{real} \leq 1$
Other	Fixed efficiency if operating in fixed efficiency mode

Table 6.52: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 12.

Line number	1
Line name	N/A
Variable number	12
Variable description	mixture ratio
Data type	real
Units	N/A
Value restrictions	$0 \leq$ `real` ≤ 1
Other	Mixture ratio for chemical thruster

Table 6.53: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 13.

Line number	1
Line name	N/A
Variable number	13
Variable description	thrust scale factor
Data type	real
Units	N/A
Value restrictions	≥ 0
Other	Constant factor by which to scale thrust

Table 6.54: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 14.

Line number	1
Line name	N/A
Variable number	14
Variable description	thrust coefficient[0]
Data type	real
Units	assumes thrust is in mN
Value restrictions	N/A
Other	Thrust coefficient of P^0. Only used if throttle table file is “none”.

Table 6.55: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 15.

Line number	1
Line name	N/A
Variable number	15
Variable description	thrust coefficient[1]
Data type	real
Units	assumes thrust is in mN
Value restrictions	N/A
Other	Thrust coefficient of P^1. Only used if throttle table file is “none”.

Table 6.56: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 16.

Line number	1
Line name	N/A
Variable number	16
Variable description	thrust coefficient[2]
Data type	real
Units	assumes thrust is in mN
Value restrictions	N/A
Other	Thrust coefficient of P^2. Only used if throttle table file is “none”.

Table 6.57: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 17.

Line number	1
Line name	N/A
Variable number	17
Variable description	thrust coefficient[3]
Data type	real
Units	assumes thrust is in mN
Value restrictions	N/A
Other	Thrust coefficient of P^3. Only used if throttle table file is “none”.

Table 6.58: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 18.

Line number	1
Line name	N/A
Variable number	18
Variable description	thrust coefficient[4]
Data type	real
Units	assumes thrust is in mN
Value restrictions	N/A
Other	Thrust coefficient of P^4. Only used if throttle table file is “none”.

Table 6.59: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 19.

Line number	1
Line name	N/A
Variable number	19
Variable description	thrust coefficient[5]
Data type	real
Units	assumes thrust is in mN
Value restrictions	N/A
Other	Thrust coefficient of P^5. Only used if throttle table file is “none”.

Table 6.60: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 20.

Line number	1
Line name	N/A
Variable number	20
Variable description	thrust coefficient[6]
Data type	real
Units	assumes thrust is in mN
Value restrictions	N/A
Other	Thrust coefficient of P^6. Only used if throttle table file is “none”.

Table 6.61: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 21.

Line number	1
Line name	N/A
Variable number	21
Variable description	mass flow coefficient[0]
Data type	real
Units	asumes mass flow is in mg/s
Value restrictions	N/A
Other	Mass flow coefficient of P^0. Only used if throttle table file is “none”.

Table 6.62: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 22.

Line number	1
Line name	N/A
Variable number	22
Variable description	mass flow coefficient[1]
Data type	real
Units	asumes mass flow is in mg/s
Value restrictions	N/A
Other	Mass flow coefficient of P^1. Only used if throttle table file is “none”.

Table 6.63: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 23.

Line number	1
Line name	N/A
Variable number	23
Variable description	mass flow coefficient[2]
Data type	real
Units	asumes mass flow is in mg/s
Value restrictions	N/A
Other	Mass flow coefficient of P^2. Only used if throttle table file is “none”.

Table 6.64: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 24.

Line number	1
Line name	N/A
Variable number	24
Variable description	mass flow coefficient[3]
Data type	real
Units	asumes mass flow is in mg/s
Value restrictions	N/A
Other	Mass flow coefficient of P^3. Only used if throttle table file is “none”.

Table 6.65: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 25.

Line number	1
Line name	N/A
Variable number	25
Variable description	mass flow coefficient[4]
Data type	real
Units	asumes mass flow is in mg/s
Value restrictions	N/A
Other	Mass flow coefficient of P^4. Only used if throttle table file is “none”.

Table 6.66: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 26.

Line number	1
Line name	N/A
Variable number	26
Variable description	mass flow coefficient[5]
Data type	real
Units	asumes mass flow is in mg/s
Value restrictions	N/A
Other	Mass flow coefficient of P^5. Only used if throttle table file is “none”.

Table 6.67: EMTG spacecraft file specification for Propulsion Library Block, line 1, entry 27.

Line number	1
Line name	N/A
Variable number	27
Variable description	mass flow coefficient[6]
Data type	real
Units	asumes mass flow is in mg/s
Value restrictions	N/A
Other	Mass flow coefficient of P^6. Only used if throttle table file is “none”.