GMAT Boundary Function Python Prototype User Guide

Noble Hatten

December 17, 2018

Contents

Abstract

This document describes how to create a new GMAT boundary function using the Python prototype interface.

1 Fundamental Concepts

1.1 Phases

A *phase* is a segment of a trajectory.

1.2 Boundary Functions

A *boundary function* is a function that is dependent on the independent and/or dependent variables of one or more phases, evaluated at a finite number of points.

2 The `PhaseConfig` Class

The `PhaseConfig` class is used to define both phases and boundary functions. When creating a boundary function, a `PhaseConfig` object is required for every phase and the boundary function itself.

2.1 Relevant Attributes

2.1.1 `OrbitStateRep`

- Attribute type: variable (string)
- Description: A string that selects the variable representation of the orbit state from among preset options. Valid values and the associated orbit state vectors are:
  - 'Cartesian': $[x, y, z, v_x, v_y, v_z]$
  - 'SphericalAzFPA': $[r, RA, DEC, v, AZI, FPA]$
  - 'ClassicalKeplerian': $[a, e, i, \omega, \Omega, \nu]$

2.1.2 `Origin`

- Attribute type: variable (string)
- Description: A string that sets the origin of the coordinate system in which the orbit state is described. The string must be recognizable as a central body by SPICE and must be in the body ephemeris file specified by the user.

2.1.3 `Axes`

- Attribute type: variable (string)
- Description: A string that sets the coordinate axes in which the orbit state is described. The string must be recognizable as a coordinate system by SPICE and must be in the coordinate system kernel specified by the user.

2.1.4 `DecVecData`

- Attribute type: variable (list)
  - Each element of the list is itself a two-element list.
- Description: A list of lists that describes the contents of the state of the phase. Each inner list contains two elements:
  1. The name of the "substate" (string)
  2. The number of elements of the substate (integer greater than zero)

  The order in which the substates are listed is the order in which data is stored in the state vector.

  The substate name "OrbitState" is special in the sense that it is recognized internally. The orbit state must always be called `OrbitState` and must consistent of six elements. The six elements correspond to the variables defined by the value of `OrbitStateRep` (Section 2.1.1).

2.1.5 `Units`

- Attribute type: variable (list of strings)
- Description: A list of strings that describes the units of each element of the state vector and is used for scaling the problem. The elements are in the same order as the corresponding elements of the state vector itself, as defined by `OrbitStateRep` and `DecVecData`. More information on units and scaling is given in Section 3.

2.2 Example

The example in this section demonstrates how to create a `PhaseConfig` object and assign valid values to relevant attributes.

```
pConfig = PhaseConfig()
pConfig.OrbitStateRep = 'Cartesian'
pConfig.Origin = 'Sun'
pConfig.Axes = 'EclipJ2000'
pConfig.DecVecData = [['OrbitState', 6], ['TotalMass', 1]]
pConfig.Units = ["DU", "ANGU", "ANGU", "VU", "ANGU", "ANGU", "MU"]
```

3 The `ScaleUtility` Class

The `ScaleUtility` class is used to define the scaling of variables for both phases and boundary functions. When creating a boundary function, a `ScaleUtility` object is required for every phase and the boundary function itself.

A `ScaleUtility` object can hold multiple units. Each unit is defined by a string name and a unit factor f and shift δ. A variable or function of a given unit is then scaled according to

$$x_{scaled} = \frac{(x_{unscaled} - \delta)}{f} \tag{1}$$

3.1 Relevant Attributes

3.1.1 `SetUnit`

- Attribute type: method
- Description: Sets an existing unit's scale factor
- Argument(s):
  1. `unitName`: string; the name of the unit
  2. `unitValue`: real; the value of the scale factor
- Output(s): None

3.1.2 `SetShift`

- Attribute type: method
- Description: Sets an existing unit's shift factor
- Argument(s):
  1. `unitName`: string; the name of the unit
  2. `unitShift`: real; the value of the shift
- Output(s): None

3.1.3 `SetUnitAndShift`

- Attribute type: method
- Description: Sets an existing unit's scale factor and shift factor
- Argument(s):
  1. `unitName`: string; the name of the unit
  2. `unitValue`: real; the value of the scale factor

3. `unitShift`: real; the value of the shift

- Output(s): None

3.1.4 `AddUnitAndShift`

- Attribute type: method
- Description: Adds a key to the `ScaleUtility` object's dictionary of units and sets the unit's scale factor and shift factor
- Argument(s):
  1. `unitName`: string; the name of the unit
  2. `unitValue`: real; the value of the scale factor
  3. `unitShift`: real; the value of the shift
- Output(s): None

Some units are pre-added to the dictionary of every `ScaleUtility` object and do not need to be added: `DU` (distance unit), `TU` (time unit), `VU` (velocity unit), `MU` (mass unit), `ACCU` (acceleration unit), and `MFU` (mass flow rate unit). For these units, the preset values of f and δ are 1 and 0, respectively.

3.2 Example

The example in this section demonstrates how to create a `ScaleUtility` object for a phase and a `ScaleUtility` object for a function and use the methods described to assign values to pre-existing units and create new units.

```
# Set scaling values
MU = 1000.0 # kg
ANGU = 1.0 # rad

muSun = 1.32712440018e+11 # km^3/s^2
DUSun = 149597870.691 # km
TUSun = 2.0*np.pi*np.sqrt(DUSun**3/muSun) # sec
VUSun = DUSun/TUSun # km/s

muEarth = 398600.4415 # km^3/s^2
DUEarth = 6378.1363 # km
TUEarth = 2.0*np.pi*np.sqrt(DUEarth**3/muEarth) # sec
VUEarth = DUEarth/TUEarth # km/s
VUSquaredEarth = VUEarth * VUEarth # km^2/s^2

# Set scaler for a phase
pScaler = ScaleUtility()
pScaler.SetUnit("DU",DUSun)
```

```
pScaler.SetUnit("TU",TUSun)
pScaler.SetUnit("VU",VUSun)
pScaler.SetUnit("MU",MU)
pScaler.SetUnit("MFU",MFUSun)
pScaler.AddUnitAndShift("ANGU",ANGU,0.0)

# Set scaler for a function
fScaler = ScaleUtility()
fScaler.AddUnitAndShift("VUSquared",VUSquaredEarth,0.0)
```

4 Creating and Configuring a Function Class

A user creates a function by creating a class for that function. The class is derived from the `GMATFunction` class, so the beginning of the class to hold a function called `VelocitySquaredFunction` might look like:

```
import numpy as np
import GMATFunction

class VelocitySquaredFunction(GMATFunction.GMATFunction):

    def __init__(self):
        """
        Constructor
        """

        # Call parent class constructor
        super().__init__()
        return
```

4.1 Configuring the Function Object

When creating an object from a user-created function class, the user is required to set several attributes:

- The number of points at which the state is evaluated in order to calculate the function.
- The number of elements of the function (e.g., 1 for a scalar function, 6 for a vector function of six elements).
- The configuration objects of all phases and the function itself.
- The scaling objects of all phases and the function itself.
- The lower and upper bounds on the function.
- Optionally, any custom settings for the function.

Other attributes would, normally, be set by CSALT. However, for prototyping purposes, the attributes are set by the user:

- The time data, state data, control data, and parameter data for each phase that is used to calculate the function.

4.1.1 Relevant Attributes

`SetNumPoints`

- Attribute type: method (derived from `AlgebraicFunction`)
- Description: Sets the number of points on which the function is dependent
- Argument(s):
  1. `numPoints`: integer; the number of points on which the function is dependent
- Output(s): None

`SetNumFunctions`

- Attribute type: method (derived from `AlgebraicFunction`)
- Description: Sets the dimension of the function vector
- Argument(s):
  1. `numFunctions`: integer; the dimension of the function vector
- Output(s): None

`SetPhaseConfig`

- Attribute type: method (derived from `GMATFunction`)
- Description: Sets the `PhaseConfig` object for a point on which the function is dependent. `SetPhaseConfig` must be called once for each point on which the function is dependent.
- Argument(s):
  1. `pointIdx`: integer; the index of the point whose phase configuration is to be set, indexed from zero
  2. `phaseConfig`: `phaseConfig` object; the `PhaseConfig` object for point `pointIdx`
- Output(s): None

`SetFunConfig`

- Attribute type: method (derived from `GMATFunction`)
- Description: Sets the `PhaseConfig` object for the function
- Argument(s):
  1. `funConfig`: `phaseConfig` object; the `PhaseConfig` object for the function
- Output(s): None

`SetPhaseScaler`

- Attribute type: method (derived from `GMATFunction`)
- Description: Sets the `ScaleUtility` object for a point on which the function is dependent. `SetPhaseScaler` must be called once for each point on which the function is dependent.
- Argument(s):
  1. `pointIdx`: integer; the index of the point whose scaling is to be set, indexed from zero
  2. `scaler`: `ScaleUtility` object; the `ScaleUtility` object for point `pointIdx`
- Output(s): None

`SetFunScaler`

- Attribute type: method (derived from `GMATFunction`)
- Description: Sets the `ScaleUtility` object for the function
- Argument(s):
  1. `scaler`: `ScaleUtility` object; the `ScaleUtility` object for the function
- Output(s): None

`SetFunBounds`

- Attribute type: method (derived from `GMATFunction`)
- Description: Sets lower and upper bounds on the function. An equality constraint is imposed by setting the lower bound and upper bound of a function equal to one another.

- Argument(s):
  1. `funLBUnscaled`: 1D numpy array of reals; array of unscaled lower bounds on the function. The elements of the array are organized in the same order as the elements of the function vector itself.
  2. `funUBUnscaled`: 1D numpy array of reals; array of unscaled upper bounds on the function. The elements of the array are organized in the same order as the elements of the function vector itself.
- Output(s): None

`SetData`

- Attribute type: method (derived from `AlgebraicFunction`)
- Description: Sets function data, where data can be time, state, control, or parameter data.
- Argument(s):
  1. `varType`: string; the type of data to be set. Valid values are:
     - "State"
     - "Control"
     - "Time"
     - "Param"
  2. `pointIdx`: integer; the index of the point whose data is to be set, indexed from zero
  3. `theData`: real (scaler or 1D numpy array); the data to be set. The dimension of `theData` is equal to the dimension of either the state vector, the control vector, time, or the parameter vector, depending on the value of `varType`.
- Output(s): None

`SetCustomSetting`

- Attribute type: method (derived from `GMATFunction`)
- Description: Sets a value for a user-defined setting for a function.
- Argument(s):
  1. `key`: string; name for the custom setting
  2. `value`: any data type; the value for the custom setting
- Output(s): None

4.1.2 Example

Altogether, setting the necessary data for a scalar function `vFun` (an instance of the class `VelocitySquaredFunction`) that depends on a single phase could look like:

```
import numpy as np

vFun = VelocitySquaredFunction() # create the function object
vFun.SetNumPoints(1) # set the number of points on which the function depends
vFun.SetNumFunctions(1) # set the dimension of the function vector

pConfig = PhaseConfig()
# ... Set pConfig as required ...
vFun.SetPhaseConfig(0,pConfig) # set the configuration object for the zeroth point

fConfig = PhaseConfig()
# ... Set fConfig as required ...
vFun.SetFunConfig(fConfig) # set the configuration object for the function

pScaler = ScaleUtility()
# ... Set pScaler as required ...
vFun.SetPhaseScaler(0,pScaler) # set the scaling object for the zeroth point

fScaler = ScaleUtility()
# ... Set fScaler as required ...
vFun.SetFunScaler(0,fScaler) # set the scaling object for the function

funLB = np.array([7.5]) # unscaled lower bound on function
funUB = np.array([20000.0]) # unscaled upper bound on function
vFun.SetFunBounds(funLB, funUB) # set the lower and upper bounds on the function

vFun.SetCustomSetting('TargetVelocitySquared', 1000.5) # add a custom setting

# Simulate CSALT populating the function with data
time = 30000.0
vFun.SetData("Time",0,time) # set the time

# State data as [RMAG, RA, DEC, VMAG, AZI, FPA] w.r.t. the Sun
stateData = np.array([150.0e6/DUSun, 1.2/ANGU,
    -0.1/ANGU, 36.0/VUSun, 0.7/ANGU, 0.2/ANGU, 3000.0/MU])
vFun.SetData("State",0, stateData) # set SCALED state data
```

4.2 The `EvaluateFunction` Method

The user is required to write a method for their function class called `EvaluateFunction` that evaluates the function and returns its *scaled* value:

`EvaluateFunction`

- Attribute type: method
- Description: Returns the scaled function value
- Argument(s): None
- Output(s):
  1. `funcValue`[1]: 1D numpy array of reals; the scaled function values

The `EvaluateFunction` uses attributes of the base `GMATFunction` class to access data (time, state, control, parameters, etc.).

4.2.1 Relevant Attributes

The contents of this section are relevant attributes of classes from which the function class derives or of objects that are attributes of the function class.

`GetUnscaledStateForFun`

- Attribute type: method (derived from `GMATFunction`)
- Description: Returns the state, unscaled and in the format desired based on the `FunConfig` and `FunScaler` objects
- Argument(s):
  1. `pointIdx`: integer; the index of the point at which the unscaled state is to be retrieved, indexed from zero
  2. `returnElements = ['All']`: list of strings; which substates are to be returned. Stings must correspond to elements specified in `PhaseConfig.DecVecData`. Passing the default list `['All']` returns all elements of the state, in the order specified in `PhaseConfig.DecVecData`.
- Output(s):
  1. `unscaledState`: 1D numpy array of reals; the unscaled state in the format desired by the function

The elements of the state are returned in the order in which they are listed in the input argument.

[1] The name of the return variable is set by the user and is unimportant.

`GetTime`

- Attribute type: method (derived from `AlgebraicFunction`)
- Description: Returns the time
- Argument(s):
    1. `pointIdx`: integer; the index of the point at which the time is to be retrieved, indexed from zero
- Output(s):
    1. `timeData`: real; the time

`GetControl`

- Attribute type: method (derived from `AlgebraicFunction`)
- Description: Returns the control
- Argument(s):
    1. `pointIdx`: integer; the index of the point at which the control vector is to be retrieved, indexed from zero
- Output(s):
    1. `controlData`: 1D numpy array; the control vector

`GetParams`

- Attribute type: method (derived from `AlgebraicFunction`)
- Description: Returns the parameters
- Argument(s):
    1. `pointIdx`: integer; the index of the point at which the parameter vector is to be retrieved, indexed from zero
- Output(s):
    1. `paramData`: 1D numpy array; the parameter vector

`GetCustomSetting`

- Attribute type: method (derived from `GMATFunction`)
- Description: Returns the value of a custom setting that was previously set using the `SetCustomSetting` method
- Argument(s):
  1. `key`: string; a key for the custom setting dictionary
- Output(s):
  1. `value`: any variable type; value of the custom setting associated with `key`

`ScaleVector` This method returns the scaled value of its input. The method accepts two arguments. The first argument is a numpy array of one or more entries to scale. The second argument is a `PhaseConfig.Units` attribute that describes how to scale the input.

4.2.2 Example

The `EvaluateFunction` method for the `VelocitySquaredFunction` class could look like:

```
def EvaluateFunction(self):
    # get the state at point 0
    # only the orbit state is required, so only 'OrbitState' is requested
    state = self.GetUnscaledStateForFun(0, returnElements=['OrbitState'])

    # get the time at point 0
    # (not used, but this example shows how to get it)
    t = self.GetTime(0)

    # the target value of the square of the velocity is set using a custom setting
    target = self.GetCustomSetting('TargetVelocitySquared')

    # calculate the function value (unscaled)
    funcValue = np.linalg.norm(v)**2 - target

    # scale the function value
    # funcValue needs to be passed in as an array because len() is used on it
    funcValue = self.funScaler.ScaleVector([funcValue], self.funConfig.Units)

    return funcValue
```

5 The `helperData` Class

Because the prototype is not integrated with GMAT, certain capabilities are implemented manually rather than through GMAT. One relevant example is accessing data related to celestial bodies, such as gravitational parameters and states as functions of time. These capabilities are necessary for converting between different state representations. In order to mimic this capability, the `helperData` class is used to hold data. Specifically, `helperData` holds two dictionaries as class variables: `mu` and `kernels`:

`mu`

- Attribute type: variable (dictionary)
- Description: Holds names of celestial bodies as keys and corresponding gravitational parameters as values

`kernels`

- Attribute type: variable (dictionary)
- Description: Holds generic names of NAIF SPICE kernels as keys and system-specific filepaths as values. There are three generic names that are used by other routines within the boundary function prototype:
  1. `deKernel`: A bsp file (e.g., de430.bsp) that hold celestial body ephemerides
  2. `naifTlsKernel`: A tls file (e.g., naif0012.tls) that holds leap seconds
  3. `pckKernel`: A pck file (e.g., pck00010.tpc) that holds celestial body orientation parameters

A user is free to edit the contents of these dictionaries to reflect files and filepaths on their own system, as well as add contents if other celestial bodies are desired.