EMTGv9 State Representations

Noble Hatten [*]

Revision Date	Author	Description of Change
April 28, 2022	Noble Hatten	Initial creation

[*]Aerospace Engineer, NASA Goddard Space Flight Center, Flight Dynamics and Mission Design Branch Code 595

Contents

List of Acronyms

EMTG Evolutionary Mission Trajectory Generator

GSAD Ghosh Sparse Automatic Differentiation

1 Overview

Evolutionary Mission Trajectory Generator (EMTG) allows the user to specify a state at a boundary in multiple state representations. These state representations are described in the StateRepresentation section of the Software Design Document. Though EMTG changes are described first, all PyEMTG changes must also be made before attempting to test the EMTG changes.

2 How to Add a State Representation to EMTG

Adding a state representation to EMTG consists of multiple steps and requires additions to both the EMTG C++ code and the PyEMTG Python code.

2.1 Preliminaries

Any new state representation requires a string name. This name will be used many times throughout EMTG and PyEMTG. The convention for a name is Pascal case; that is, the first letter of each compound word in the string is capitalized, including the first. For example, "IncomingBplane".

In addition, each individual element of the state representation must have string name. By convention, these strings are all caps. For example, for IncomingBplane, we have "VINF", "RHA", "DHA", "BRADIUS", "BTHETA", and "TA". The ordering of the string names is also meaningful and must be consistent across all uses.

2.2 State Transformation Equations

Any new state representation must be transformable to and from a Cartesian state representation. Thus, the user must derive and code (and verify and document!) equations to transform from the new state representation to Cartesian, and from Cartesian to the new state representation. The Jacobians of both these transformations are also required. The gravitational parameter of the central body to which the state representation is referenced is available for use in the transformations.

If another parameter is required, a larger change would be required to EMTG and PyEMTG than is currently discussed in this document.

2.3 EMTG Updates

2.3.1 New Files

A C++ source and header file must be created and placed in `src/Astrodynamics/StateRepresentations`. The convention is to name the source and header files the name of the new state representation appended with "StateRepresentation". For example, "OutgoingBplaneStateRepresentation.cpp".

The easiest way to create the new files is to copy/paste the source/header files for an existing state representation and use them as a starting point. Following that paradigm, the following changes must be made to the new header file:

- Change the name of the include-guard macro to reflect the name of the new file.
- Change the name of the class to be the name of the new file.
- Add any class properties deemed useful. Whether or not it is beneficial to add class properties may not be known until changes are made to the source file.

Note that all specific state representations are derived from the `StateRepresentationBase` class, so it may be beneficial to browse `StateRepresentationBase`.

The following changes must be made to the new source file:

- Change the name of the included header file to reflect the name of the header file for the new state representation.
- Replace all instances of the old state representation class name with the name state representation class name.
- In the constructor:
  - Set the `name` property to the string name of the new state representation.
  - Set the `stateNames` property to the string names of the individual state representation element names (in order!).
- In the `convertFromRepresentationToCartesian` method:
  - The purpose of this method is to convert from the new state representation to a Cartesian state representation. The Cartesian state must be placed in the `StateVectorCartesian` property.

  - In the `if (needG) {}` block, the Jacobian of the conversion must be calculated and placed in the `RepresentationToCartesianTransitionMatrix` property.

- In the `convertFromCartesianToRepresentation` method:
  - The purpose of this method is to convert from a Cartesian state representation to the new state representation. The state in the new state representation must be placed in the `StateVectorThisRepresentation` property.
  - In the `if (needG) {}` block, the Jacobian of the conversion must be calculated and placed in the `CartesianToRepresentationTransitionMatrix` property.

All `double` variables should be declared as `doubleType` in order to maintain compatibility with Ghosh Sparse Automatic Differentiation (GSAD).

2.3.2 EMTG Enums

In `src/Core/EMTG_enums.h`, there is an enum called `StateRepresentation`. The string name of the new state representation must be added to the end of this enum. Similarly, the string name *as a string* must be added to the end of the variable `StateRepresentationStrings`.

2.3.3 StateRepresentationFactory

In `src/Astrodynamics/StateRepresentaton/StateRepresentationFactory.cpp`:

- Add an `#include` statement for the header file for the new state representation class.
- Add an `else if {}` block for the new state representation, modeled after the blocks for the existing state representations.

2.3.4 FreePointBoundary

In `src/Mission/Journey/Phase/BoundaryEvents/FreePointBoundary.cpp`:

- In the `calcbounds_event_left_side()` method, in the `switch (this->myStateRepresentationEnum) {}` block, add a `case` block for the new state representation, based on the existing `case` blocks. In a given `push_back()`, the arguments are the string name of the state element, the scale factor for the state element, and whether or not the state element is an angle. Note that the string names for each state element do not have to be the same as the state names used in the new state representation's `stateNames` property. For example, IncomingBplane and OutgoingBplane have the same `stateNames` values, but have different string names in `FreePointBoundary->statesToRepresent`. `FreePointBoundary->statesToRepresent` sets the names for the states in the decision vector descriptions (i.e., what is written in the XF-file.csv).

2.3.5 PeriapseBoundary

In `src/Mission/Journey/Phase/BoundaryEvents/PeriapseBoundary.cpp`:

- In the `initialize()` method, in the `//do we need the rdotv constraint? if{}` block, add the enum for the new state representation to the list of conditions *if* the new state representation has a state that can be directly constrained to ensure a periapse. For example, classical orbital elements are included in the list of conditions because true anomaly is part of that state representation, and constraining true anomaly to be 0 is sufficient to ensure a periapse.
- In the `initialize()` method, in the `//do we need the distance constraint? if{}` block, add the enum for the new state representation to the list of conditions *if* the new state representation has a state that can be directly constrained to ensure a distance at periapse. For example, spherical state representations are included in the list of conditions because r is part of the state representation.
- In the `calcbounds_event_left_side()` method, in the `switch (this->myStateRepresentationEnum) {}` block, add a `case` block for the new state representation, based on the existing `case` block. In a given `push_back()`, the arguments are the lower bound and upper bound on each state element. The values selected are dependent on the state representation. However, there are important points:
  - Angle bounds are in degrees.
  - For a state representation that directly constrains a periapse (e.g., with true anomaly), lower and upper bounds must be set to ensure a periapse. For example, for classical orbital elements, the upper and lower bounds on true anomaly are set to `math::SMALL` and `-math::SMALL`, respectively.
  - For a state representation that directly constrains periapse distance, lower and upper bounds must be set to ensure that distance. For example, for a spherical representation, the upper and lower bounds on r must be set to `RadiusBounds[1]` and `RadiusBounds[0]`, respectively.
  - For a state representation that directly encodes v_∞, the upper and lower bounds on that state must be set to `VelocityMagnitudeBounds[1]` and `VelocityMagnitudeBounds[0]`, respectively.

2.3.6 Documentation

A new state representation must be accompanied by the following documentation:

- Doxygen-compatible comments in source code.
- A description of the new state representation in the StateRepresentation section of the EMTG Software Design Document.
- A mathematical specification for the new state representation in `docs/`.

2.3.7 Testing

Verification that EMTG is acting appropriately includes:

- Verifying the results of the state transformations.
- Executing the missiontestbed when using the new state representation and ensuring that:
  - The elements in the .emtgopt file and the XFfile are as expected
  - The derivatives in the Gout file match their automatic differentiation counterparts.
  - There are no elements in the missing entries file.

2.4 CMAKE Updates

The new source and header files added to EMTG must be added to the appropriate variables in `src/Astrodynamics/StateRepresentation/CMakeLists.txt`. Specifically,

```
${CMAKE_CURRENT_SOURCE_DIR}/<NewFileName>.h
```

must be added to `STATEREPRESENTATION_HEADERS` and

```
${CMAKE_CURRENT_SOURCE_DIR}/<NewFileName>.cpp
```

must be added to `STATEREPRESENTATION_SOURCE`. This is a straightforward matter of following the conventions of the lists already present in the CMakeLists file.

2.5 PyEMTG Updates

2.5.1 The GUI

The Journey Options panel and the Physics Options panel must both be updated. For the Journey Options panel:

- In `PyEMTG/JourneyOptionsPanel/OrbitElementsPanel.py`, in the `OrbitElementsPanel` constructor, add the enum number and string name of the new state representation to `elements_representation`
- In `PyEMTG/JourneyOptionsPanel/ArrivalElementsPanel.py`, in the `ArrivalElementsPanel.update()` method, add an `elif` block for the new state representation based on the existing blocks for existing representations.

- In `PyEMTG/JourneyOptionsPanel/DepartureElementsPanel.py`, in the `DepartureElementsPanel.update` method, add an `elif` block for the new state representation based on the existing blocks for existing representations.
- In `PyEMTG/JourneyOptionsPanel/ArrivalElementsPanelToAEI.py`, in the `ArrivalElementsPanelToAEI.u` method, add an `elif` block for the new state representation based on the existing blocks for existing representations.
- In `PyEMTG/JourneyOptionsPanel/ArrivalElementsPanelToEnd.py`, in the `ArrivalElementsPanelToEnd.u` method, add an `elif` block for the new state representation based on the existing blocks for existing representations.

For the Physics Options panel:

- In `PyEMTG/PhysicsOptionsPanel.py`, add the string name of the new state representation to the `StateRepresentationChoices` assignment.

2.5.2 StateConverter

The `StateConverter` class is used to take a state in one representation and convert it to another representation. Currently, this is done by having a method for every possible permutation of input/output state representations, with the Cartesian representation used as a intermediary. Thus, when a new representation is added, new methods are required of the form:

```
def NewRepresentationtoCartesian(self, stateNewRepresentation, mu):
"""
Parameters
----------
stateNewRepresentation : 6-element list of floats
State in new representation: [state0Name, state1Name state2Name,
state3Name, state4Name, state5Name]
mu : float
Gravitational parameter of central body

Returns
-------
6-element list of floats
State in Cartesian: [x, y, z, vx, vy, vz]
"""
...Math...
return stateCartesian

def CartesiantoNewRepresentation(self, stateCartesian, mu):
"""
```

```
Parameters
----------
stateCartesian : 6-element list of floats
State in Cartesian: [x, y, z, vx, vy, vz]
mu : float
Gravitational parameter of central body

Returns
-------
6-element list of floats
State in new representation: [state0Name, state1Name state2Name,
state3Name, state4Name, state5Name]
"""
...Math...
return stateNewRepresentation
```

For transformations to other representations, a template like the following is used:

```
def NewRepresentationtoCOE(self, stateNewRepresentation, mu):
stateCartesian = self.NewRepresentationToCartesian(stateNewRepresentation, mu)
return self.CartesiantoCOE(stateCartesian, mu)
```

Currently, the required permutations are:

- `NewRepresentationtoCartesian()`
- `NewRepresentationtoCOE()`
- `NewRepresentationtoSphericalAZFPA()`
- `NewRepresentationtoSphericalRADEC()`
- `NewRepresentationtoIncomingBplane()`
- `NewRepresentationtoOutgoingBplane()`
- `NewRepresentationtoIncomingBplaneRpTA()`
- `NewRepresentationtoOutgoingBplaneRpTA()`
- `CartesiantoNewRepresentation()`
- `COEtoNewRepresentation()`
- `SphericalAZFPAtoNewRepresentation()`
- `SphericalRADECtoNewRepresentation()`
- `IncomingBplanetoNewRepresentation()`

- `OutgoingBplanetoNewRepresentation()`
- `IncomingBplaneRpTAtoNewRepresentation()`
- `OutgoingBplaneRpTAtoNewRepresentation()`

The `convertDecisionVector()` method must also be updated. This method is used to automatically convert elements on an EMTG decision vector from one state representation to another. This is used, for example, in the HighFidelityConverter and when using the GUI to change the free point state parameterization. Specific changes required are described below. As always, use existing code as a template!

- Add a list containing the names of the states of the state parameterization as they appear in the EMTG decision vector. I.e., the same as the names written in `FreePointBoundary->statesToRepresent`
- Add an `elif` block to the `if 'left state x' in description:` block. The purpose of this `if` block is to determine, based on decision vector description strings, whether a state in a particular representation has been encountered. Thus, the `elif` condition must uniquely identify that a state in the new state representation has been encountered. The `elif` condition should start with `left state`, then include a string that is unique to the representation. For example, COE uses `elif 'left state SMA' in description`.
  - In the case that the text used to uniquely identify the state representation is not in the zeroth element of the state, then this must be taken into account when stepping through the decision vector. See the use of the variable `delta` in the `elif 'left state TAin' in description` block. This `delta` may be reused for future state representations. Its value is set equal to the value that needs to be added to the current `Xindex` in order to access the zeroth element of the state. For example, if the unique text is located in the fifth element of the state (zero-indexed), then `delta = -5`. Note that the B plane state representation blocks are newer than the other state representation blocks and contain more maintainable code, particularly when use of `delta` is needed. As a result, it is recommended to start with one of these blocks when writing the new code block.
  - After the `elif` block is created, an interior `if` block based on `desiredStateRep` is used to call the appropriate method to transform the state to the correct representation. If `desiredStateRep` is the same as the current state rep, then no transformation is used, and `Xindex` is simply incremented. It is highly recommended that a copy/paste of an existing case be used as the basis for anew case.
- In each of the previously existing outer `if/elif` blocks, inner `elif` blocks for state conversion if `desiredStateRep` is the new state representation must be added.

2.5.3 Documentation

All new PyEMTG code must be commented using comments compatible with Sphinx automatic documentation generation.

2.5.4 Testing

Verification that PyEMTG is acting appropriately includes:

- Verifying the results of the state transformations in `StateConverter`.
- Verifying that the proper labels and fields appear in the GUI.
- Verifying that switching state representations in the GUI and saving results in expected changes to decision variable values in the .emtgopt file.

2.6 Journey Options Updates

In OptionsOverhaul/list_of_journeyoptions.csv, update the rows whose "name" is:

- departure_elements_state_representation. Increase the value in the "upperBound" column by 1. Update the "comment" and "description" columns to reference the new state representation.
- arrival_elements_state_representation. Increase the value in the "upperBound" column by 1. Update the "comment" and "description" columns to reference the new state representation.
- **NOTE: Code currently restricts probe entry element representations. DO NOT do this line until further checking is done.** Probe_AEI_elements_state_representation. Increase the value in the "upperBound" column by 1. Update the "comment" and "description" columns to reference the new state representation.
- **NOTE: Code currently restricts probe entry element representations. DO NOT do this line until further checking is done.** Probe_End_elements_reference_epoch. Increase the value in the "upperBound" column by 1. Update the "comment" and "description" columns to reference the new state representation.

2.7 Mission Options Updates

In OptionsOverhaul/list_of_missionoptions.csv, update the rows whose "name" is:

- PeriapseBoundaryStateRepresentation. Increase the value in the "upperBound" column by 1. Update the "comment" and "description" columns to reference the new state representation.

2.8 Execute Options Overhaul Script

After performing the tasks in Sections 2.6 and 2.7, execute the Python script in PyEMTG/OptionsOverhaul/make_F
This creates new Python scripts and C++ code, so EMTG will need to be rebuilt after executing this script. Note that the `EMTG_path` variable must be edited to match the user's repository location.

2.9 Test System Updates

Relevant regression tests must be created and placed in testatron/tests/state_representation_tests/. These new tests plus all existing unit tests must be executed and results confirmed prior to acceptance of the new state representation.