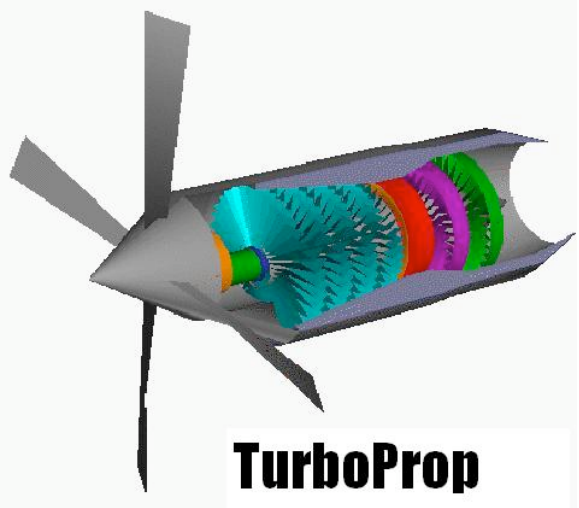

Image credit: NASA Glenn Research Center.

TurboProp Version 4.0
21 May 2009

Keric Hill and Brandon Jones
Project Geryon
Colorado Center for Astrodynamics Research
University of Colorado, Boulder

Geryon, a warrior in Greek mythology with three bodies, was only defeated when Hercules shot him with a poison arrow. The aim of the Project Geryon is to conquer the Three-body Problem and use its unique astrodynamics to create new possibilities for the exploration of space. (http://ccar.colorado.edu/geryon)

Contents

1 TurboProp

TurboProp is a package of functions that can be used to quickly propagate precise orbital or surface trajectories in the convenience of the MATLAB and Python programming environments. To speed up the computations, the orbit propagators are initial value ODE solvers coded in the C programming language. The orbit propagators can be called as MATLAB functions through a MEX interface or Python via the SWIG module, so the user gets both the superior execution speed of C code along with the ease of matrix computations and plotting that come with a high level, interpreted language.

The orbit propagators, or ODE solvers, in TurboProp include a fixed-step fourth-order Runge-Kutta solver, a variable step fourth/fifth-order Runge-Kutta solver, a variable-step seventh/eighth-order Runge-Kutta solver, and two symplectic Runge-Kutta integrators of 4th and 6th order. The surface station/vehicle propagator is a MATLAB function that generates the coordinates of the station/vehicle in inertial space.

TurboProp was originally written to propagate orbits in the Circular Restricted Three-body Problem and convert them to the JPL ephemeris. Earth Orientation and a drag model have been added that will allow users to propagate orbits near Earth. Included in this release are the JPL planetary ephemerides DE403 and DE405, the lunar gravity models GLGM-2, LP100K, and LP150Q, and the Earth gravity models GGM02C, JGM-3 and WGS-84.

If you wish to keep the functions available for older versions of TurboProp, please rename those functions so they won't conflict with the new ones in Version 4.0.

Section 2 contains installation instructions for configuring TurboProp to work on your machine. Section 3 provides some details on the interface with Python. Section 4 describes the integrators in detail. Section 5 explains the ODE functions (or derivative functions) that are available with this version. Section 6 explains how to use auxiliary MATLAB functions included with TurboProp.

The following are changes made since v.3.3:

- TurboProp is now compatible with the Python scripting language (given dependencies described later).
- The spherical harmonics formulation has been changed from the classical form in Vallado and McClain (2001) to the cartesian representation described in Gottlieb (1993). Although (slightly) not as computationally efficient, it removes the singularity at the poles. See the `_grav` section for more details.
- Introduced a derivative function and the associated code to model the gravity field based on a collection of mass concentrations (mascon).
- Removed dynamic allocation from all spherical harmonic code to increase code reliability and reduce computation time.
- While changing the spherical harmonics model and adding the mascon model, the interfaces to the `Gravity/` directory functions were streamlined.
- Added two new integrators: one 4th and one 6th order, symplectic Runge-Kutta.
- Added derivative functions that allow compatibility with the unscented Kalman filter. These derivative functions are added on an as needed basis by the developers.
- A graphical user interface (GUI) was added to TurboProp to assist in learning the software.
- Improved execution speed and added vector input/output capabilities to `elorb()` and `randv()`.
- Removed calls to `fgetpos()` and `fsetpos()` in the JPL ephemeris software. This improves portability and reliability of the software.
- Added the `FillExtras.c` and `FillExtras.h` files to centralize population of the extras structure in MATLAB mex files.
- Check for integrator failure after derivative function shutdown is complete.
- Allow for `options` array of 3 values for debugging (depends on compile mode and the integrator used).
- Add a check to see if the integration step size is a NaN or smaller than the machine precision.
- Added an `attitude` field to the extras structure for some users.

- MATLAB time conversion software updated for the new leap second.
- Slight change to the eclipse fraction if-test in _SRP derivative functions. This prevents the derivative function from generating a NaN when the satellite orbit radius is less than the radius of the primary body.
- Some general code clean-up/removal.

2 Installation

Before installing this new version of TurboProp, you may wish to rename your older-version TurboProp functions if you wish to retain the capability to use them.

You will need to download the zip file TurboProp4.0.zip containing TurboProp v.4.0 into the directory of your choosing. Uncompress the zip file and run the follow the software specific installation instruction, which follow in Sections 2.1 and 2.2.

When running the installation script, you will be prompted whether to compile in debug mode. When the software is compiled in debug mode, the rk45 and rk78 integrators can generate an output file that includes the time of each evaluation of the derivative function and the number of derivative function calls. The generation of the file is determined by the input options. This capability allows the user to profile the performance of the integrator and assists with debugging. However, this creates some computation overhead that is not desired for most applications. Thus, this capability is only enabled when the software is compiled in debug mode. In order to switch between debug mode and the non-debug mode, the installation script must be rerun with the appropriate option selected.

The installation scripts create the following files and directories in the TurboProp install directory:

<**InstallDir**>		
	Data/	Ephemeris, gravity, and EOP data files
	Derivatives/	C derivative functions.
	Ephemeris/	JPL ephemeris utilities.
	Gravity/	Gravity field utilities.
	Help/	MATLAB help files.
	Integrators/	Integrators, deriv_mex code.
	MATLAB/	MATLAB functions.
	PyUtils/	Python functions.
	findptrmaker.m	Function to find ODE functions.
	HelpFileMaker.m	Function to create help files.
	InstallScript.m	Script to install TurboProp.
	InstallScript.py	Script to install TurboProp.
	TurboPath.m	Script to add TurboProp to the MATLAB path.
	TurboTypes.h	C variable type definitions file.
	__init__.py	Definition of the TurboProp package in Python.

The install scripts will also convert the ASCII ephemeris files into a more efficient set of binary

ephemeris files. The binary ephemeris file for DE403 is called `DE.403` and the binary file for DE405 is called `DE.405`. The binary ephemeris files need to be installed on each machine as well, since the binary data is stored differently on different computers.

The LP100K gravity field coefficients file is called `jgl100k1.sha` and the LP150Q coefficients file is called `jgl150q.sha`, while `GLGM2.sha` contains the GLGM-2 gravity field. The LP150Q file has the permanent tide correction already applied to J_2 and $C_{2,2}$. The Grace Gravity Model GGM02C coefficients file also has the permanent tide corrections already applied and is called `GGM02C.sha`. The JGM-3 gravity file is called `JGM3.sha`, and the WGS-84 files is `WGS84.sha`. IERS data from April 2007 is in the `finals.data` file.

After the installation script completes, you may run `testephem.m` to make sure that the JPL ephemeris files were stored properly and can be read accurately. `testephem.m` is in the `Ephemeris/` directory, and will make two plots of the errors. If any of the blue error points are outside of the red dashed lines, there may be a problem in your files. The command window output will also tell you if there are any erroneous points. The DE403 and DE405 errors may appear to be different magnitudes, but that is just because the DE405 truth file (`testpo.405`) does not have as many digits of precision as the DE403 truth file (`testpo.403`).

After verifying that the ephemeris files are working properly with `testephem.m`, you can erase all of the ASCII ephemeris files to save hard drive space. These files are named:

```
ascp1960.405  ascp1950.403
ascp1980.405  ascp1975.403
ascp2000.405  ascp2000.403
ascp2020.405  ascp2025.403
ascp2040.405
```

If you experience any difficulties, or if you wish to have integration test scripts to verify that the integrators are working properly, contact Brandon Jones.

2.1 MATLAB Installation

In MATLAB, run `InstallScript.m` and it will compile the neccesary C code into MEX functions. If this is your first time compiling a MEX function, MATLAB may ask you which compiler you wish to use. You can select the C compiler that comes with MATLAB, which usually looks like:

```
[1] Lcc C version 2.4
```

Beyond the compiler used, one can further customize the compilation of the C software by editing the mexopts.sh file. For example, external libraries can be defined and included at compile time. See the MATLAB documentation for more information.

When installing TurboProp on Linux boxes, you may get a warning that the version of `gcc` on your machine is more recent than the versions used to test MEX compilation. Usually this will not cause any problems, but contact Brandon Jones if errors occur.

The new MEX functions will have a platform specific extension, according to the table below:

System Type	MEX File Extension
Sun Solaris	`.mexsol`
HP-UX	`.mexhpux`
Linux on PC	`.mexglx`
Linux on AMD Opteron	`.mexa64`
Macintosh	`.mexmac`
Windows	`.mexw32` or `.dll`

The MEX functions should be reinstalled on each machine on which you wish to use them, even if they use the same operating system. They are not usually compatible when copied and pasted from one machine to another.

In Windows, the installation script will automatically add the proper directories to the end of the MATLAB path for you, so you can call the TurboProp functions just like MATLAB's own built-in functions. UNIX, Macintosh, or Linux users will have to save the TurboProp directories in the path definition file or startup file themselves to make it permanent. Make sure that the `Help/` directory is last. Another option is to run the `TurboPath.m` script at every startup to add the TurboProp paths to MATLAB for that session. All ephemeris files are visible in the MATLAB path, so to retrieve the file name and path from any directory, use the function `which` in MATLAB. For example, to find the DE405 file, you can type:

```
extras.ephemfile = which('DE.405');
```

`extras.ephemfile` will now contain a string with the entire path to the DE405 file.

2.2 Python Installation

The Python installation is similar to the MATLAB directions provided previously. Run the `InstallScript.py` from the Python or shell command line to compile the necessary C code into Python libraries. Note there are some dependencies, which are summarized below.

- Simplified Wrapper and Interface Generator (SWIG), `http://www.swig.org`
- The compiler used to compile you native version of Python

Note SWIG is commonly distributed on Linux systems and the compiler is required to install Python. If either dependency is not met, the installation script will return an error.

During installation, you may get warnings associated with the SWIG generated files (`*_wrap.c`). These are typical with some installations and have not been known to cause any problems with the generated Python libraries. Additionally, some warnings may be returned for files in the `Gravity/` directory. These can mostly be ignored. If you have any questions, e-mail Brandon Jones.

After the software is installed, the location of the TurboProp software must be added to the `PYTHONPATH` environment variable for future use in Python software. Additionally, the TurboProp directory itself must be specified in the user's environment in the `TURBOPROP_DIR` variable. The exact values of these variables will be specified at the end of the installation script.

Unlike MATLAB, Python does not provide a convenient `which` command for loading the path to the TurboProp data files. However, the TurboProp Python libraries provide the absolute path to the `Data` directory. For example:

```
import os
import TurboProp

extras['ephemfile'] = os.path.join( TurboProp.DataDir, 'DE.405' )
```

will insert the absolute location of the `DE.405` file into the `extras` dictionary under the `ephemfile` key.

3 Python Interface Definition

Before moving on to the description of the integrators, it is prudent to define the TurboProp interface when using the Python language. If the user only plans on using the MATLAB capabilities, skip to Section 4.

This section assumes some familiarity with the Python language. It is not intended to provide a tutorial or describe the language. The user is directed to the many references on the web, starting with the Python website (`http://www.python.org`).

TurboProp in Python was designed to exploit the object oriented capabilities and the modular definition of Python packages. Once the software has been installed (and the environment variables are set), the basic TurboProp interface is provided by loading the TurboProp package, i.e. :

```
import TurboProp
```

This loads some basic definitions to make the interface with Python easier. Specifically, two variables are included in this top level package: `Dir` and `DataDir`. The `Dir` variable includes the absolute path to the TurboProp directory, while the `DataDir` includes the absolute path to the `Data/` directory.

Loading the main TurboProp module does not load the complete package, which would require more time at initialization and loads potentially unnecessary software for a given application. There are two packages within TurboProp that may prove useful to the user, and they are accessed by

```
import TurboProp.Integrators
import TurboProp.PyUtils
```

The `Integrators` package contains all of the integrators and the derivative function evaluator. To access a specific integration module, for example the `rk4fix` integrator, use the import command

```
import TurboProp.Integrators.rk4fix
```

All of the `rk4fix` integration capabilities will now be available in Python. The same format is used for all integrators included in TurboProp. For more information on the integrators and their interface,

see Section 4. As dictated by the Python language, the user can reassign the name of the imported integration module to another variable. For example,

```
import TurboProp.Integrators.rk4fix as rk4fix
```

Currently, there are no modules of use for the user in the `PyUtils` package. Current modules are used by the software during installation or by the `Integrators` package. The `PyUtils` package will include the Python equivalent to the MATLAB utilities described in Section 6 as they are developed and released.

Throughout this document, Fortran indexing (indices start with 1) is used to match the syntax of MATLAB. However, Python uses C indexing (indices start with 0). To improve readability, the MATLAB notation is used throughout the document when talking about variables commonly used in both languages. For Python users, subtract the indices by 1. Additionally, the user should make the appropriate changes to syntax (`y0[0]=1.0` versus `y0(1)=1.0;`).

4 Integrators

Orbit integrators, or propagators, are a way of solving the ordinary differential equation (ODE)

$$\dot{\mathbf{X}}(t) = \mathbf{F}\left(\mathbf{X}(t), t\right). \tag{1}$$

where $\mathbf{X}$ is a state vector, t is time, and $\dot{\mathbf{X}}(t)$ is the derivative of $\mathbf{X}$ with respect to time. With an initial state $\mathbf{X}(t_0)$, and a method of computing $\mathbf{F}\left(\mathbf{X}(t_0), t_0\right)$, future states $\mathbf{X}(t)$ can be obtained. The method of computing the future states is called an initial value solver for the ODE.

There are eight ODE solvers in TurboProp. They are `rk4fix`, `rk4fix_dstm`, `rk45`, `rk78_dstm`, `rk78`, `rk78_dstm`, `rk4sym`, and `rk6sym`. The interface with these solvers varies with the calling software to exploit advantages of the given language. A description of these integrators can be called from MATLAB by typing `help solvername`, where `solvername` is the name of the TurboProp integrator. Python help information can be accessed by importing the integrator module from the Python interpreter and using the `help()` function.

The `_dstm` solvers are useful for sequential processors in orbit determination. For example, when using a Kalman filter it is much faster to perform one function call to integrate a whole trajectory with automatic initialization of the state transition matrix (STM). The slower alternative would be to call the integrator in chunks, manually reinitializing the STM after each time. Although the integrators are very fast, there is a performance penalty associated with the function call. The interface between the scripting language and the C code requires some extra computations, but those computations are performed only once per function call. Note, the normal integrators can be used with the proper derivative function to generate the STM for batch processors like the least-squares filter.

While the ODE solvers are used to generate trajectories for orbiting spacecraft, the MATLAB function `Surfer.m` is used to propagate trajectories in inertial space for ground stations or vehicles. `SurferIC.m` is useful for generating initial conditions for `Surfer.m` using inputs of latitude, longitude, height, heading, and speed. For information on `Surfer.m`, see Section 6. This capability is not yet implemented in Python.

4.1 rk4fix

rk4fix is a fixed-step, fourth-order Runge-Kutta integrator. Since rk4fix is a fixed-step integrator, the intervals in tspan must be evenly divisible by the time step. The only exception is the last interval, which may be any length of time. rk4fix_mex requires that the options vector have two values. options(1) contains the time step size, or the step size used in the integration. options(2) contains a tolerance used to verify that the time step evenly divides the time span intervals (except the last). If your time step evenly divides the time span, but numerical issues make it so that it appears it does not, use a larger value for options(2). A good value is normally 1e-6.

4.2 rk4fix_dstm

After storing the output of the state vector in Y, rk4fix_dstm will reinitialize the state transition matrix (STM) portion of the state vector to an identity matrix. Otherwise, it is the same as rk4fix.

4.3 rk45

rk45 is a variable step Runge-Kutta integrator that compares the results of 4th and 5th order integrations to adjust the step size. Like the ode45 function in MATLAB, the integration scheme described in Dormand and Prince (1980) is utilized. However the proprietary interpolation routine used by MATLAB to determine the state at user specified nodes is not available, so results may differ slightly when compared to ode45. rk45 is a variable-step integrator, meaning that the integration step size is adjusted to keep the difference between the fourth- and fifth-order Runge-Kutta integrators less than a certain tolerance. However, the user still must input an initial guess or starting step size that will be used to begin the integration. rk45 requires that the options vector have two values. options(1) contains the initial guess at the time step size, or the step size used in the integration. options(2) contains a tolerance used when comparing the seventh- and eighth-order Runge-Kutta integrations. The smaller this tolerance is, the smaller the time steps must be so that the fourth- and fifth-order Runge-Kutta integrations agree more closely to each other. When TurboProp is compiled in debug mode, a third option may be provided (options(3)) that determines if a file will be generated that includes each evaluation time of the derivative function, and the total number of evaluations. To enable this capability, set options(3) equal to 1 (1.0 in Python). Otherwise, a value of 0 or no value will prevent the file from being generated. The file will be created in the current working directory, and named rk45out.txt.

4.4 rk45_dstm

After storing the output of the state vector in Y, rk45_dstm will reinitialize the state transition matrix (STM) portion of the state vector to an identity matrix. Otherwise, it is the same as rk45.

4.5 rk78

rk78 is a variable step Runge-Kutta integrator that compares the results of 7th and 8th order integrations to adjust the step size. This integrator algorithm was modified by Jeff Parker using the code

ode78 (v1.11) in MATLAB written by Marc Compere in 2001. rk78 is a variable-step integrator, meaning that the integration step size is adjusted to keep the difference between the seventh- and eighth-order Runge-Kutta integrators less than a certain tolerance. However, the user still must input an initial guess or starting step size that will be used to begin the integration. rk78 requires that the options vector have two values. options(1) contains the initial guess at the time step size, or the step size used in the integration. options(2) contains a tolerance used when comparing the seventh- and eighth-order Runge-Kutta integrations. The smaller this tolerance is, the smaller the time steps must be so that the seventh- and eighth-order Runge-Kutta integrations agree more closely to each other. When TurboProp is compiled in debug mode, a third option may be provided (options(3)) that determines if a file will be generated that includes each evaluation time of the derivative function, and the total number of evaluations. To enable this capability, set options(3) equal to 1 (1.0 in Python). Otherwise, a value of 0 or no value will prevent the file from being generated. The file will be created in the current working directory, and named rk78out.txt.

4.6 rk78_dstm

After storing the output of the state vector in Y, rk78_dstm will reinitialize the state transition matrix (STM) portion of the state vector to an identity matrix. Otherwise, it is the same as rk78.

4.7 rk4sym

The rk4sym is a fixed-step, symplectic, fourth order Runge-Kutta integrator. The integration scheme utilizes the 2-stage Gauss-Legendre scheme describe in Leimkuhler and Reich (2004). Since rk4sym is a fixed-step integrator, the intervals in tspan must be evenly divisible by the time step. The only exception is the last interval, which may be any length of time. rk4sym requires that the options vector have two values. options(1) continas the time step size, or the step size used in the integration. Since the integration scheme is implicit, a fixed point iteration is required to estimate the step using the time step. The convergence tolerance is set in options(2).

Unlike the explicit Runge-Kutta integrators previously described, symplectic integrators preserve the Hamiltonian of the system. Thus, a small time step is not required to preserve accuracy. Of course, these time steps vary with the system propagated. More information on symplectic integrators can also be found in Hairer et al. (2002).

4.8 rk6sym

The rk6sym is a like the rk4sym integrator, except it uses the 3-stage Gauss-Legendre scheme in Leimkuhler and Reich (2004). Given the higher order solution, a larger time step than the rk4sym without a loss of accuracy.

4.9 MATLAB Integrator Call Syntax

The syntax for calling the TurboProp integrators was designed to be similar to the ODE solvers in MATLAB such as ode45. One of the formats for calling ode45 is:

```
[T,Y] = ode45(odefun,tspan,y0,options,extras)
```

The syntax for the TurboProp integrators is:

```
[T,Y] = rk4fix_mex(odefun,tspan,y0,options,extras)
[T,Y] = rk4fix_dstm_mex(odefun,tspan,y0,options,extras)
[T,Y] = rk45_mex(odefun,tspan,y0,options,extras)
[T,Y] = rk45_dstm_mex(odefun,tspan,y0,options,extras)
[T,Y] = rk78_mex(odefun,tspan,y0,options,extras)
[T,Y] = rk78_dstm_mex(odefun,tspan,y0,options,extras)
[T,Y] = rk4sym_mex(odefun,tspan,y0,options,extras)
[T,Y] = rk6sym_mex(odefun,tspan,y0,options,extras)
```

The inputs and outputs for these integrators are described in detail in Sections 4.11 through 4.17. Note the calling function is the name of the integrator, followed by the `_mex` tag to prevent a conflict with any similarly named software on the MATLAB path.

4.10 Python Integrator Call Syntax

As previously mentioned, the Python interface to TurboProp was designed to exploit the object-oriented capabilities of Python. To that end, each TurboProp integrator has two access options: a class interface and a function interface. The function interface is intentionally similar to the MATLAB interface. The integrator syntax (using the shorthand defined in Section 3) is:

```
[T,Y] = rk4fix.rk4fix(odefun,tspan,y0,options,extras)
[T,Y] = rk4fix_dstm.rk4fix_dstm(odefun,tspan,y0,options,extras)
[T,Y] = rk45.rk45(odefun,tspan,y0,options,extras)
[T,Y] = rk45_dstm.rk45_dstm(odefun,tspan,y0,options,extras)
[T,Y] = rk78.rk78(odefun,tspan,y0,options,extras)
[T,Y] = rk78_dstm.rk78_dstm(odefun,tspan,y0,options,extras)
[T,Y] = rk4sym.rk4sym(odefun,tspan,y0,options,extras)
[T,Y] = rk6sym.rk6sym(odefun,tspan,y0,options,extras)
```

This syntax may seem redundant given the function name is the same as the module. If the user wishes to prevent this redundancy, the function can be assigned to a variable, i.e.

```
from TurboProp.Integrators.rk4fix import rk4fix as rk4fix

[T,Y] = rk4fix(odefun,tspan,y0,options,extras)
```

The class definition of the interface is slightly different. For many applications, only the function definition is necessary. However, with each call to the TurboProp integration software, some initialization is performed based on the `odefcn` used. For example, ephemeris files may be loaded into memory. For repeated calls to the integrators, this becomes inefficient. The class interface is used to prevent this since these initialization tasks are only performed at the time of the class instantiation. Assuming the same `odefcn`, `options`, and `extras`, the integration function can be repeatedly called without

reinitializing the integrator.

Assuming the modules have been imported, the calls to instantiate the classes are:

```
Cl = rk4fix.RungeKutta4Fix(odefcn,options,extras,tspan=tspan,y0=y0)
Cl = rk4fix_dstm.RungeKutta4Fix_dstm(odefcn,options,extras,
                                     tspan=tspan,y0=y0)
Cl = rk45.RungeKutta45(odefcn,options,extras,tspan=tspan,y0=y0)
Cl = rk45_dstm.RungeKutta45_dstm(odefcn,options,extras,
                                 tspan=tspan,y0=y0)
Cl = rk78.RungeKutta78(odefcn,options,extras,tspan=tspan,y0=y0)
Cl = rk78_dstm.RungeKutta78_dstm(odefcn,options,extras,
                                 tspan=tspan,y0=y0)
Cl = rk4sym.RungeKutta4Sym(odefcn,options,extras,tspan=tspan,y0=y0)
Cl = rk6sym.RungeKutta6Sym(odefcn,options,extras,tspan=tspan,y0=y0)
```

As indicated above (congruent with all Python interface documentation), the `tspan` and `y0` inputs are optional at the time of class instantiation. They may be useful for any derivative functions requiring knowledge of the initial state or the possible integration times.

After instantiation, the integration is performed by

```
[T,Y] = Cl( tspan, y0 )
```

This command may be repeatedly called for different `tspan` and `y0` values.

4.11 `odefun` (Input)

`odefun` is a string with the name of the ODE function, or the derivative function. As an example assignment in MATLAB, you could type `odefun = 'CRTBP';` Section 5 explains the ODE functions included in TurboProp. Unfortunately, users can not easily create their own ODE functions to use with the TurboProp integrators. Advanced programmers would have to code up their own derivative functions in C and compile them with their integrators. For more information, please contact Brandon Jones.

4.12 `tspan` (Input)

`tspan` is a vector (either a row or a column) with values of time. In Python, it is an array of floats. The first value in `tspan` has to be the time associated with the initial state given in `y0`. States will be output in `Y` for all of the values in `tspan`. Derivative functions may have required units for `tspan`, so verify that you are using the correct time scale. The `DE`-type derivative functions now require that `tspan` be in seconds, where it used to be days. The fixed-step integrators also require you to create `tspan` so that the integration time step will evenly divide each interval in `tspan`, except the last.

4.13 y0 (Input)

y0 is a vector (either a row or a column) with the state parameters. In Python, y0 is an array of floats. y0 should describe the state at the time given in tspan(1). When the state vector includes a state transition matrix (STM), it should be added on to the end of the state vector in column order. As an example assignment in MATLAB with a six-element state and a STM, you could type

```
y0 = [x y z dx dy dz];                          (this assigns the state)
y0(6+1:6+36) = reshape(eye(6),36,1);            (this adds on the STM)
```

In Python, the assignment is similar, but uses the syntax for assigning variables to an array. However, for definition of the STM, you could use

```
for i in xrange(0,6):
     for j in xrange(0,6):
          if i == j :  y0.append(1.0)
          else:  y0.append(0.0
```

4.14 options (Input)

options is, in MATLAB, a vector (either a row or a column) of numbers that are used by the integrator. In Python, it is an array of floats. options can include integration tolerances, step sizes, and other parameters used by the integration function. See sections 4.1 through 4.6 for details on what to include in the options vector.

4.15 extras (Input)

In MATLAB, extras is a structure data type passed on to the ODE function, or derivative function. This variable is used to pass on constants that are needed to compute the derivative of the state, but that aren't estimated. The fields in extras are described in the following sections. If a field is included in extras that is not needed by the ODE function, the ODE function will ignore that field.

In Python, extras is a dictionary with keys set to the fields of the extras structure. For example,

```
extras['gravfile'] = os.path.join( TurboProp.DataDir, 'GGM02C.sha' )
```

Otherwise, the definitions are the same. For the following subsections, this difference will not be repeated. Any reference to extras.* is equivalent to extras['*'].

4.15.1 extras.mu

extras.mu is a scalar double containing a gravitational parameter. For the CRTBP-type ODE functions, this is the three-body gravitational parameter, or mass ratio. For the TwoBody-type ODE functions, this is the gravitational parameter of the central body.

extras.mu is optional for the DE-type ODE functions, and can be used to assign a customized value for the central body's gravitational parameter (in km^3/s^2).

4.15.2 extras.planets

extras.planets is a vector of 13 values (converted to integers) used in the DE-type ODE functions. This vector indicates which planets will be included in the model. A one indicates that the gravitational influence of a planet is used in the propagation and a zero (or any other integer) indicates that it is not. Each parameter in extras.planets looks like this:

extras.planets(1)	1 = Mercury is used, 0 = Mercury is not
extras.planets(2)	1 = Venus is used, 0 = Venus is not
extras.planets(3)	1 = Earth-Moon Barycenter is used, 0 = it is not
extras.planets(4)	1 = Mars is used, 0 = Mars is not
extras.planets(5)	1 = Jupiter is used, 0 = Jupiter is not
extras.planets(6)	1 = Saturn is used, 0 = Saturn is not
extras.planets(7)	1 = Uranus is used, 0 = Uranus is not
extras.planets(8)	1 = Neptune is used, 0 = Neptune is not
extras.planets(9)	1 = Pluto is used, 0 = Pluto is not
extras.planets(10)	Any value (will be automatically set to 0)
extras.planets(11)	1 = Sun is used, 0 = Sun is not
extras.planets(12)	1 = Earth is used, 0 = Earth is not
extras.planets(13)	1 = Moon is used, 0 = Moon is not

The Earth-Moon Barycenter models the combined mass of the Earth and Moon at a single point (the Barycenter) and should not be used when the Earth or the Moon are used separately. The following example shows what users would type in MATLAB if they wished to include the Sun, Venus, Earth, the Moon, and Jupiter in the propagation:

```
extras.planets = [0 1 0 0 1 0 0 0 0 0 1 1 1];
```

4.15.3 extras.radius

extras.radius is a scalar double used in the CRTBP_grav-type ODE functions. It contains the radius of the secondary body in length units (LU).

extras.radius is optional for the DE_grav-type ODE functions, and can be used to assign a customized value for the central body's reference radius (in km). This is the radius used in the spherical harmonic equations, but it is not used for eclipse calculations.

4.15.4 extras.center

extras.center is a scalar double (converted to an integer in C) that specifies the planet that is the center of the coordinate frame used in the integration, according to the list below:

```
  extras.center = 0:  Mercury Centered Inertial
  extras.center = 1:  Venus Centered Inertial
  extras.center = 2:  Earth-Moon Barycenter Centered Inertial
  extras.center = 3:  Mars Centered Inertial
  extras.center = 4:  Jupiter Centered Inertial
  extras.center = 5:  Saturn Centered Inertial
  extras.center = 6:  Uranus Centered Inertial
  extras.center = 7:  Neptune Centered Inertial
  extras.center = 8:  Pluto Centered Inertial
 extras.center = 10:  Sun Centered Inertial
 extras.center = 11:  Earth Centered Inertial
 extras.center = 12:  Moon Centered Inertial
```

`extras.center` cannot be any other value.

4.15.5 extras.ephemfile

`extras.ephemfile` For the `DE`-type ODE functions, the name of the binary planetary ephemeris file has to be sent to the TurboProp integrator. This can be done in the following way:

```
extras.ephemfile = which('DE.405');
```

4.15.6 extras.degord

`extras.degord` is a vector with two doubles (converted to integers in C) that is used by the `grav`-type ODE functions. `extras.degord(1)` is the maximum degree to be used in the spherical harmonic gravity model. `extras.degord(2)` is the maximum order to be used in the spherical harmonic gravity model. For example, if you wish to use a 50×50 gravity field, type the following code in MATLAB:

```
extras.degord = [50 50];
```

The effective value of `extras.degord(2)` will be `extras.degord(1)` if it is larger than `extras.degord(1)`. The values in `extras.degord` cannot be larger than the gravity field you supply in the gravity file.

4.15.7 extras.degordstm

`extras.degordstm` is a vector with two doubles (converted to integers in C) that is used by the `grav_stm`-type ODE functions. `extras.degordstm(1)` sets the maximum degree of the spherical harmonic gravity model used in the variational equations. `extras.degord(2)` sets the maximum order of the spherical harmonic gravity model used in the variational equations. For example, if you wish to use a 20×20 gravity field in the variational equations used to compute the state transition matrix, type the following code in MATLAB:

```
extras.degordstm = [20 20];
```

If you wish to neglect all the aspherical gravity effects in the state transition matrix and just include the point mass contributions, use code like the following in MATLAB:

```
extras.degordstm = [0 0];
```

If the values in extras.degordstm are greater than those in extras.degord, than the value of extras.degord will be used instead of using extras.degordstm. Likewise, the effective value of extras.degordstm(2) will be extras.degordstm(1) if it is larger than extras.degordstm(1).

4.15.8 extras.gravfile

For the grav-type ODE functions, the name of the gravity field (spherical harmonic coefficients or point masses) file has to be sent to the TurboProp integrator. This can be done in the following way:

```
extras.gravfile = which('jgl150q.sha');
```

4.15.9 extras.A_m

extras.A_m is a scalar double containing the area-to-mass ratio of the spacecraft. The units depend on the ODE function used. It is m^2/kg for the DE-type functions. extras.A_m is used in the solar radiation pressure (SRP) calculations. The area is considered constant and is the cross-sectional area of the spacecraft facing the Sun. If extras.A_m is zero, the SRP model is turned off.

4.15.10 extras.reftime

For the DE-type ODE functions, the Julian Ephemeris Date (JED) is used to query the JPL DE ephemerides for planetary positions, so the time scale of integration is in days. To convert from a UTC calendar date to JED, use the function UTC2JED.m described in Section 6, or JD2JED.m. Since JED numbers are so large, some digits of precision are lost. This can be remedied by sending a reference JED in the extras vector. The time span in tspan should be relative to that reference date. In other words, if the desired time span for the integration (in JED) was:

```
[2454003.5007 2454008.5007]
```

The user would input a time span of

```
tspan = 86400*[0 5];
```

and set the reference date extras.reftime = 2454003.5007. The scale factor of 86,400 is required because the time unit in DE-type ODE functions is now seconds instead of days.

4.15.11 extras.EOP

Earth Orientation Parameters (EOP) are contained in extras.EOP. This is used to convert from an inertial coordinate system to an Earth- or Planet-fixed coordinate system. For the TwoBody_grav-type functions, extras.EOP is a vector containing two doubles:

extras.EOP(1): θ_0, Sidereal Time at time zero, t_0.

extras.EOP(2): $\dot{\theta}$, Rotation rate of the central body.

These parameters are used to find the angle between the x-axis of the inertial frame and the x-axis of the surface-fixed frame. This angle is measured as a counter-clockwise rotation about the positive z-axis (right hand rule), the same as Greenwich Mean Sidereal Time. The equation used to compute $\theta(t)$, the sidereal time at time t, is shown below:

$$\theta(t) = \theta_0 + \dot{\theta}t \tag{2}$$

Both quantities in `extras.EOP` use units of radians to describe angles, while `extras.EOP(2)` uses whatever time unit is used in the state vector, gravitational parameter, and time span vector. It is up to the user to make sure all the units are consistent.

For the `DE_grav`-type functions where the central body is Earth, `extras.EOP` is a vector used to convert from an Earth-centered Inertial frame (GCRF) to the Earth-fixed frame (ITRF). This is done in the `Orient()` function within the C code in the MEX functions and also in the `EarthFrame.m` function in MATLAB. They both use the 1976 Precession, the 1980 Nutation (with corrections), Earth rotation, and Polar Motion. For details, see Section 5.1.6. The `extras.EOP` array can be filled automatically using the `EOPfind.m` function and a data file from IERS. The file included with TurboProp, `finals.data`, can be updated by downloading a new one from any of these locations:

```
ftp://maia.usno.navy.mil/ser7/finals.data
ftp://maia.usno.navy.mil/ser7/finals.all
http://maia.usno.navy.mil/ser7/finals.data
http://maia.usno.navy.mil/ser7/finals.all
```

For these Earth-centered integrations in the JPL ephemeris system, `extras.EOP` is a vector containing fifteen doubles:

`extras.EOP(1)`:	Integer portion of the EOP reference time, t_{ref}, in JD$_{TT}$.
`extras.EOP(2)`:	Fractional portion of the EOP reference time, t_{ref}, in JD$_{TT}$.
`extras.EOP(3)`:	N_{leap}, Number of leap seconds at EOP reference time.
`extras.EOP(4)`:	$x_p(t_{ref})$, bias term (radians)
`extras.EOP(5)`:	$\dot{x}_p$, rate term (radians/day)
`extras.EOP(6)`:	$y_p(t_{ref})$, bias term (radians)
`extras.EOP(7)`:	$\dot{y}_p$, rate term (radians/day)
`extras.EOP(8)`:	$[UT1 - UTC](t_{ref})$, bias term (seconds)
`extras.EOP(9)`:	$\frac{d}{dt}[UT1 - UTC]$, rate term (seconds/day)
`extras.EOP(10)`:	$LOD(t_{ref})$, bias term (days)
`extras.EOP(11)`:	$\frac{d}{dt}LOD$, rate term (days/day)
`extras.EOP(12)`:	$\partial\Delta\Psi_{1980}(t_{ref})$, bias term (radians)
`extras.EOP(13)`:	$\frac{d}{dt}\partial\Delta\Psi_{1980}$, rate term (radians/day)
`extras.EOP(14)`:	$\partial\Delta\epsilon_{1980}(t_{ref})$, bias term (radians)
`extras.EOP(15)`:	$\frac{d}{dt}\partial\Delta\epsilon_{1980}$, rate term (radians/day)

To allow for sufficient numerical precision in some applications, the reference time was split into two variables. Thus, the Julian Date in Terrestrial Time (JD$_{TT}$) for 1 Jan 2007, 16:38:35 UTC, would be:

```
extras.EOP(1) = 2454110
extras.EOP(2) = 0.194215092592593
```

`EOPfind.m` will automatically compute these times when it creates the `extras.EOP` vector. Otherwise, Terrestrial Time conversions can be performed using `UTC2TT.m` or `JED2TT.m`. Since integration in the `DE`-type derivative functions is performed using JED as the time scale, the `Orient()` function will automatically convert from JED to JD$_{TT}$.

The polar motion parameters x_p and y_p are computed for time t (in JD$_{TT}$) using a linear interpolation.

$$x_p(t) = x_p(t_{ref}) + \dot{x}_p(t - t_{ref}) \quad (3)$$
$$y_p(t) = y_p(t_{ref}) + \dot{y}_p(t - t_{ref}) \quad (4)$$

Earth rotation parameters are computed for time t (in JD$_{TT}$) using a linear interpolation.

$$[UT1 - UTC](t) = [UT1 - UTC](t_{ref}) + \frac{d}{dt}[UT1 - UTC](t - t_{ref}) \quad (5)$$
$$LOD(t) = LOD(t_{ref}) + \frac{d}{dt}LOD(t - t_{ref}) \quad (6)$$

IERS precession and nutation correction parameters are computed for time t (in JD$_{TT}$) using a linear interpolation.

$$\partial\Delta\Psi_{1980}(t) = \partial\Delta\Psi_{1980}(t_{ref}) + \frac{d}{dt}\partial\Delta\Psi_{1980}(t - t_{ref}) \quad (7)$$
$$\partial\Delta\epsilon_{1980}(t) = \partial\Delta\epsilon_{1980}(t_{ref}) + \frac{d}{dt}\partial\Delta\epsilon_{1980}(t - t_{ref}) \quad (8)$$

Using this linear interpolation, the resulting EOP are only accurate for about one day, so new EOP will need to be computed and the integration restarted about every 24 hours. Section 5.1.6 describes these parameters and how they are used in detail.

4.15.12 `extras.atmos`

`extras.atmos` contains three parameters used to compute the atmospheric density for drag computations:

`extras.atmos(1)`: ρ_0, the atmospheric reference density, or nominal density.
`extras.atmos(2)`: r_0, the atmospheric reference radius, or base radius.
`extras.atmos(3)`: H, the atmospheric normalizing factor, or scale height.

The density model used is an exponential atmosphere model, and the density, ρ, at radius r from the center of the body is

$$\rho(r) = \rho_0 e^{\left(\frac{r_0 - r}{H}\right)}. \quad (9)$$

For the `TwoBody`-type derivative functions, the units in ρ_0, r_0, and H must agree with the units used in the integration (and `extras.mu`, `extras.A_m`). For example, if the state vector had units of km and km/s, `extras.mu` had units of km^3/s^2, and `extras.A_m` had units of km^2/kg, the units for ρ_0 should be kg/km^3 and the units for r_0 and H should be km.

For the `DE`-type derivative functions, the units for ρ_0 must be kg/m^3, the units for r_0 must be km, and the units for H must be km.

Sample values for these parameters can be found in Vallado and McClain (2001), Table 8-4, p. 534 (2nd ed. 1st Prnt)

4.15.13 extras.attitude

extras.attitude contains three (or four parameters) used to represent the vehicle attitude. This field has only been used for select applications, so their use has not been fully defined. The initialization software allows the extras.attitude vector to be of length 3 or 4 to accommodate both Euler angles (or modified Rodriguez parameters) or quaternions.

4.16 T (Output)

T is a vector of times (an array in Python), which for all the integrators in the current version of TurboProp should be identical to tspan.

4.17 Y (Output)

Y is a matrix. Each row in Y gives the value of the state vector (in row form) at the corresponding time in T. For example, Y(3,:) will have the state vector integrated to time T(3), which is the same as tspan(3). If the state transition matrix (STM) is part of the state vector, it can be extracted in the following way (when the state is a six-vector):

```
STM = reshape(Y(i,6+1:6+36),6,6);
```

In this example, i is the row number. The STM for time tspan(i) would be stored in the variable STM as a 6×6 matrix.

In Python, Y is a list of lists where each list corresponds to a state at the corresponding time in the T array.

5 ODE Functions

Details on the state vectors and extras vectors for all the ODE functions are given later in this section. The current ODE functions in TurboProp are:

TwoBody	CRTBP	DE	PointMasses
TwoBody_stm	CRTBP_stm	DE_stm	
TwoBody_grav	CRTBP_2D	DE_SRP	
TwoBody_grav_stm	CRTBP_stm_2D	DE_SRP_stm	
TwoBody_drag_grav	CRTBP_grav	DE_grav	
TwoBody_drag_grav_stm	CRTBP_grav_stm	DE_grav_stm	
TwoBody_grav_ukf		DE_SRP_grav	
TwoBody_drag_grav_ukf		DE_SRP_grav_stm	
		DE_drag_grav	
		DE_drag_grav_stm	
		DE_drag_SRP_grav	
		DE_drag_SRP_grav_stm	

The list of accepted ODE functions can be obtained in MATLAB by typing help odefun.

5.1 Mathematical Theory for ODE Functions

The mathematical models and requirements used in the ODE functions, along with the abbreviations used in the function names, are described in detail in the sections that follow.

5.1.1 Two-body Problem, "`TwoBody_`"

The two-body problem, where a spacecraft orbits a single, massive central body. The acceleration due to the central body is

$$\ddot{\mathbf{r}}_{2body} = -\frac{\mu \mathbf{r}}{r^3}, \tag{10}$$

where $\mathbf{r}$ is the position vector of the spacecraft with respect to the planet' center of mass. r is the magnitude of that vector.

5.1.2 Circular Restricted Three-body Problem, "`CRTBP_`"

In the circular-restricted three-body model, there are two massive bodies in orbit about their mutual barycenter, as depicted in Figure 1. To simplify the model, each body orbits the barycenter in the same plane in perfectly circular orbits. A spacecraft with infinitesimal mass experiences forces due to the gravitational influence of both bodies simultaneously, and the two bodies are approximated as point masses. The more massive body is labeled P_1 and the other is P_2. The coordinate frame has its origin at the barycenter and rotates with the two bodies so that P_1 and P_2 are always on the x-axis, with the positive x-direction going from P_1 to P_2. The positive y-axis is parallel to the velocity vector of P_2. The three-body gravitational parameter is called μ:

$$\mu = \frac{m_2}{m_1 + m_2}. \tag{11}$$

where m_1 is the mass of P_1 and m_2 is the mass of P_2. The mass of the primaries is nondimensionalized so that the mass of P_2 is defined to be μ and the mass of P_1 is 1 - μ. One nondimensional length unit (LU) is equal to the distance between the two primaries, so the distance along the x-axis from the origin to P_1 is -μ LU and from the origin to P_2 is 1 - μ LU. The time unit (TU) is defined such that P_2 orbits around P_1 in 2π TU.

The equations of motion for the CRTBP are (Murray and Dermott, 1999)

$$\begin{aligned} \ddot{x} - 2\dot{y} &= x - (1-\mu)\frac{x+\mu}{r_1^3} - \mu\frac{x+\mu-1}{r_2^3} \\ \ddot{y} + 2\dot{x} &= \left(1 - \frac{1-\mu}{r_1^3} - \frac{\mu}{r_2^3}\right) y \\ \ddot{z} &= \left(\frac{\mu-1}{r_1^3} - \frac{\mu}{r_2^3}\right) z, \end{aligned} \tag{12}$$

$$\text{where} \quad r_1 = \sqrt{(x+\mu)^2 + y^2 + z^2} \quad \text{and} \quad r_2 = \sqrt{(x+\mu-1)^2 + y^2 + z^2}. \tag{13}$$

The functions `CRTBP_DE.m` and `rot_inert.m` can be used to convert from an inertial frame to the rotating, non-dimensional frame used in the CRTBP ODE functions. See Section 6 for more details on those functions.

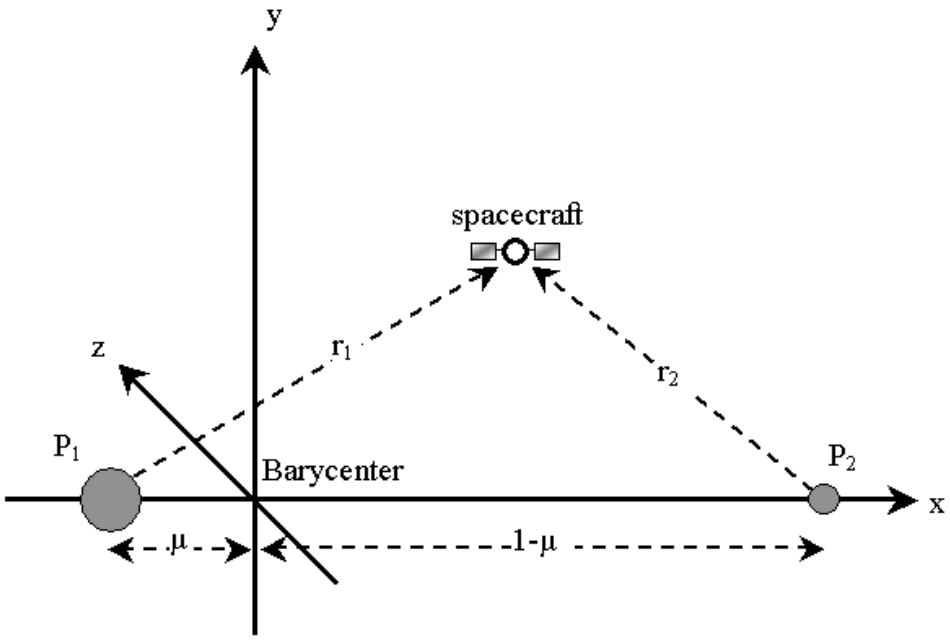

Figure 1: Diagram of the Circular Restricted Three-Body Problem with a rotating, nondimensional coordinate frame.

The origin of the coordinate frame for the `CRTBP_grav`-type functions is not at the three-body barycenter like the other `CRTBP` models, but the origin is at the center of the secondary, with the x-axis pointed toward the primary and the z-axis perpendicular to the orbit plane of the primary and secondary. Note that this reference frame is rotated 180° about the z-axis from the other `CRTBP` models. These new axes are called $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ and the coordinates can be converted like this:

$$\begin{aligned} \tilde{x} &= -x - (1-\mu) \\ \tilde{y} &= -y \\ \tilde{z} &= z \\ \dot{\tilde{x}} &= -\dot{x} \\ \dot{\tilde{y}} &= -\dot{y} \\ \dot{\tilde{z}} &= \dot{z}. \end{aligned} \tag{14}$$

(15)

The converted equations of motion are

$$\begin{aligned} \ddot{\tilde{x}} - 2\dot{\tilde{y}} &= \tilde{x} - (1-\mu) - (1-\mu)\frac{\tilde{x}-1}{r_1^3} - \mu\frac{\tilde{x}}{r_2^3} \\ \ddot{\tilde{y}} + 2\dot{\tilde{x}} &= \left(1 - \frac{1-\mu}{r_1^3} - \frac{\mu}{r_2^3}\right)\tilde{y} \\ \ddot{\tilde{z}} &= \left(\frac{\mu-1}{r_1^3} - \frac{\mu}{r_2^3}\right)\tilde{z}, \end{aligned} \tag{16}$$

$$\text{where} \qquad r_1 = \sqrt{(1-\tilde{x})^2 + \tilde{y}^2 + \tilde{z}^2} \qquad \text{and} \qquad r_2 = \sqrt{\tilde{x}^2 + \tilde{y}^2 + \tilde{z}^2}. \tag{17}$$

5.1.3 JPL Ephemeris System, "DE_"

The JPL planetary ephemeris model. `DE` ODE functions are in the ICRF inertial reference frame centered on a user-specified planetary body, or the Sun. The reference plane of the ICRF is the mean equator of J2000, and the reference vector is the dynamic equinox of J2000. The positions and velocities of all the planets are obtained from JPL DE ephemerides: DE403 (Standish et al., 1995) or DE405 (Standish, 1998). These ephemerides contain lunar orientation angles that can be used to rotate from the inertial Moon-centered coordinates to Moon-fixed coordinates. The ephemerides also contain Earth Nutation angles for converting between inertial and fixed coordinates. The binary ephemeris file for DE403 is called `ephem.403` and the binary file for DE405 is called `ephem.405`.

The code used to create the ephemeris files and to perform the interpolation was adapted from work by CCAR graduate student Greg Lehr and code by Mark Hoffman (Hoffman, 1998) at

`ftp://ssd.jpl.nasa.gov/pub/eph/export/C-versions/hoffman/`

The ephemeris data files were obtained from

`ftp://ssd.jpl.nasa.gov/pub/eph/export/`

In the `DE`-type functions, the acceleration due to the central body is

$$\ddot{\mathbf{r}}_{2body} = -\frac{\mu \mathbf{r}}{r^3}, \tag{18}$$

where $\mathbf{r}$ is the position vector of the spacecraft with respect to the planet's center of mass. r is the magnitude of that vector. μ is the gravitational parameter of the body.

The acceleration due to other planetary bodies, or 'third bodies' (Tapley et al., 2004) is

$$\ddot{\mathbf{r}}_{3body} = -\mu_{3body}\left(\frac{\mathbf{r}_{3,sat}}{r_{3,sat}^3} + \frac{\mathbf{r}_{\oplus,3}}{r_{\oplus,3}^3}\right), \tag{19}$$

where μ_{3body} is the gravitational parameter of the third-body, $\mathbf{r}_{3,sat}$ is the vector from the third-body to the satellite, $\mathbf{r}_{\oplus,3}$ is the vector from the central body to the third-body, and $r_{3,sat}$ and $r_{\oplus,3}$ are the vector magnitudes.

By default, the gravitational parameters are taken from the JPL ephemeris, as shown in Table 1.

The user can provide `extras.mu` as an optional input to the `DE`-type ODE functions, and its value will be used as the gravitational parameter for the body that is the center of integration. The gravitational parameters for the other planets will remain at their default values. The value in `extras.mu` will also be used instead of the gravitational parameter in a gravity file, if the ODE function requires it.

Table 1: DEFAULT GRAVITATIONAL PARAMETERS FROM JPL EPHEMERIDES.

Planet	DE403	DE405
Mercury	$2.20320804864179\times10^{4}$	$2.20320804864179\times10^{4}$
Venus	$3.2485859882646\times10^{5}$	$3.2485859882646\times10^{5}$
Earth+Moon	$4.03503234715983\times10^{5}$	$4.03503233479087\times10^{5}$
Mars	$4.28283142580671\times10^{4}$	$4.28283142580671\times10^{4}$
Jupiter	$1.26712767857796\times10^{8}$	$1.26712767857796\times10^{8}$
Saturn	$3.79406260611373\times10^{7}$	$3.79406260611373\times10^{7}$
Uranus	$5.79454900707188\times10^{6}$	$5.79454900707188\times10^{6}$
Neptune	$6.83653406387926\times10^{6}$	$6.83653406387926\times10^{6}$
Pluto	$9.81600887707005\times10^{2}$	$9.81600887707005\times10^{2}$
Sun	$1.32712440017987\times10^{11}$	$1.32712440017987\times10^{11}$
Earth	$3.98600435608103\times10^{5}$	$3.98600432896939\times10^{5}$
Moon	$4.90279910787977\times10^{3}$	$4.90280058214776\times10^{3}$

Units are km^3/s^2

5.1.4 Mass Concentration Gravity Field, "`PointMasses_`"

The mass concentration model is a gravity field where the total acceleration is determined by a collection of point masses, each with a given mass and location. The net acceleration is

$$\ddot{\mathbf{r}}_{pointmass} = -G\sum_i \frac{m_i \bar{\mathbf{r}}_i}{r_i^3}, \tag{20}$$

where m_i is the mass of i-th point and $\bar{\mathbf{r}}_i$ is the vector from the point mass to the satellite. r_i is the magnitude of the $\bar{\mathbf{r}}_i$ vector. Note, this is essentially the sum of the two-body accelerations due to each point mass. In TurboProp,

$$G = 6.67428 \times 10^{-20} \quad \mathrm{km}^3/(\mathrm{kg}\cdot\mathrm{sec}^2). \tag{21}$$

This value was selected based on Mohr et al. (2007). Note, based on the units of this value, all input values must be in km and seconds.

The definition of the point masses, sometimes referred to as mascons, is provided to the integrator in the `extras.gravfile` option. The file format is rather straightforward, with one header line and a collection of data lines. The format of the header and data are provided in Tables 2 and 3, respectively. The given mass for each point is represented as a fraction of the total mass of the system. The total mass is derived from the gravitation parameter given in the header line. Locations of the point masses are defined in the planet fixed frame in spherical coordinates, and rotated based on the `extras.EOP` over time. An example file (`example.ptm`) is provided in the `Data/` directory.

Table 2: HEADER LINE FORMAT FOR POINT MASS FILES

C format string: `"%le, %d"`		
Parameter Description	Format	Units
Gravitation Parameter of the "planet"	Scientific notation	km^3/s^2
Number of point masses	Integer	N/A

Table 3: DATA LINE FORMAT FOR POINT MASS FILES

C format string: `"%le, %le, %le, %le"`		
Parameter Description	Format	Units
Fraction of total mass for given point	Scientific Notation	N/A (fraction of whole)
Point Radius	Scientific Notation	km
Point Latitude	Scientific Notation	degrees
Point Longitude	Scientific Notation	degrees

5.1.5 Spherical Harmonic Gravity Field, "_grav"

A spherical harmonic gravity field model is used for one of the planetary bodies. Current ODE functions are only compatible with Earth and lunar gravity fields. In other words, Earth-fixed and lunar-fixed orientations are the only ones currently available. Two lunar gravity fields are from Konopliv et al. (2001). The lunar LP100K gravity field coefficients file is called `jgl100k1.sha` and the lunar LP150Q coefficients file is called `jgl150q.sha`. `jgl150q.sha` has the permanent tide correction already applied to J_2 and $C_{2,2}$. The GLGM-2 gravity field is from Lemoine et al. (1997) and is called `GLGM2.sha`. The gravity model for the Earth comes from the GRACE mission (Tapley et al., 2005) and is called `GGM02C.sha`. The JGM-3 gravity field (Tapley et al., 1996) is called `JGM3.sha`, while the WGS-84 gravity field (NGA, 2000) is named `WGS84.sha`.

The gravity field format was based after the file format for the lunar gravity field `jgl100k1.sha` found at

`http://pds-geosciences.wustl.edu/missions/lunarp/shadr.html`

The first line is a header and contains eight parameters, three of which are used and the rest discarded. The parameters are shown in Table 4.

Note that these parameters are comma-delimited. If `extras.mu` was not input by the user, the gravitational parameter and reference radius from this header line will be used for all gravity calculations for the central body. The user can provide `extras.mu` and `extras.radius` to override the parameters in the gravity header line.

After the header line, "data lines" contain the gravity field coefficients. These coefficients are comma-delimited also, and are described in Table 5.

Table 4: HEADER LINE FORMAT FOR GRAVITY FIELD FILES

C format string: `" %le, %le, %le, %d, %d, %d, %le, %le"`		
Parameter Description	Format	Units
Reference radius of the planet	Scientific notation	km
Gravitational parameter	Scientific notation	km^3/s^2
Gravitational parameter uncertainty	Scientific notation	km^3/s^2 (discarded)
Degree of model field	Integer	N/A (discarded)
Order of model field	Integer	N/A (discarded)
Normalization	Integer	0 = unnormalized 1 = normalized
Reference longitude, normally 0	Scientific notation	degrees (discarded)
Reference latitude, normally 0	Scientific notation	degrees (discarded)

Table 5: DATA LINE FORMAT FOR GRAVITY FIELD FILES

C format string: `" %d, %d, %le, %le, %le, %le"`	
Parameter Description	Format
Degree index n	Integer
Order index m	Integer
Coefficient $C_{n,m}$	Scientific Notation
Coefficient $S_{n,m}$	Scientific Notation
Uncertainty in coefficient $C_{n,m}$	Scientific Notation
Uncertainty in coefficient $S_{n,m}$	Scientific Notation

The Earth and the Moon are the only planets supported in `DE_grav`-type derivative functions, so `extras.center` should only be 11 or 12 for those functions.

The gravity field is modeled using U, an aspherical potential function excluding the two-body term (Gottlieb (1993)),

$$U(x,y,z) = \frac{\mu}{r}\sum_{n=2}^{\infty}\sum_{m=0}^{n}\left(\frac{R}{r}\right)^n \bar{A}_{n,m}\left[\frac{z}{r}\right]\{\bar{C}_{n,m}\bar{E}_m + \bar{S}_{n,m}\bar{F}_m\} \tag{22}$$

where

$$r = \sqrt{x^2+y^2+z^2} \tag{23a}$$

$$\bar{E}_m = \bar{E}_1\bar{E}_{m-1} - \bar{F}_1\bar{F}_{m-1}, \quad \bar{E}_0 = 1, \quad \bar{E}_1 = \frac{x}{r} \tag{23b}$$

$$\bar{F}_m = \bar{F}_1\bar{E}_{m-1} + \bar{E}_1\bar{F}_{m-1}, \quad \bar{F}_0 = 0, \quad \bar{F}_1 = \frac{y}{r}. \tag{23c}$$

Note this form is defined in terms of the Cartesian, not spherical, coordinates. Thus, this formulation does not contain the singularity at the poles present in the classical formulation presented in Vallado and McClain (2001), and elsewhere. R is the radius of the central body, μ is the gravitational parameter of the central body, and n and m are the degree and order of the spherical harmonic model respectively. $\bar{C}_{n,m}$ and $\bar{S}_{n,m}$ are normalized coefficients describing the magnitude of the spherical harmonics, and are specific to the gravity field model being used. $\bar{A}_{n,m}\left[\alpha = \frac{z}{r}\right]$ is a normalized, derived Legendre function computed using the following recursion:

$$\begin{aligned}
\bar{A}_{0,0}[\alpha] &= 1 \\
\bar{A}_{1,0}[\alpha] &= \alpha\sqrt{3} \\
\bar{A}_{1,1}[\alpha] &= \sqrt{3} \\
\bar{A}_{n,n}[\alpha] &= \sqrt{\frac{2n+1}{2n}}\bar{A}_{n-1,n-1}[\alpha] \qquad n > 1 \\
\bar{A}_{n,m}[\alpha] &= \alpha B_{n,m}\bar{A}_{n-1,m}[\alpha] - \frac{B_{n,m}}{B_{n-1,m}}\bar{A}_{n-2,m}[\alpha] \qquad m < n
\end{aligned} \tag{24}$$

where

$$B_{n,m} = \sqrt{\frac{(2n+1)(2n-1)}{(n+m)(n-m)}}. \tag{25}$$

Since the $B_{n,m}$ coefficients are not a function of α, they are precomputed at software initialization and stored in memory for future use. This formulation of the derived Legendre functions is adapted from personal correspondence with Dr. Bob Schutz of the University of Texas Center for Space Research and from his book Satellite Dynamics: Theory and Application current being written with G. Giacaglia. This adaptation to the formulation in Schutz's book was based on Lundberg and Schutz (1988), and is consistent with Gottlieb (1993).

The acceleration in body-fixed coordinates, $\ddot{\mathbf{r}}$, is found by taking the gradient of the potential in cartesian coordinates. As derived in Gottlieb (1993), the following equations summarize the algorithm:

$$a_1 = \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n \sum_{m=1}^{n} m\bar{A}_{n,m}\left(C_{n,m}\bar{E}_{m-1} + S_{n,m}\bar{F}_{m-1}\right) \tag{26a}$$

$$a_2 = \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n \sum_{m=1}^{n} m\bar{A}_{n,m}\left(S_{n,m}\bar{E}_{m-1} - C_{n,m}\bar{F}_{m-1}\right) \tag{26b}$$

$$a_3 = \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n \sum_{m=0}^{n} \bar{A}_{n,m+1}\Gamma_{n,m}\left(C_{n,m}\bar{E}_{m} + S_{n,m}\bar{F}_{m}\right) \tag{26c}$$

$$a_4 = \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n \sum_{m=0}^{n} (n+m+1)\bar{A}_{n,m}\left(C_{n,m}\bar{E}_{m} + S_{n,m}\bar{F}_{m}\right) \tag{26d}$$

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^2}\left(\left(\frac{z}{r}a_3 + a_4\right)\frac{\mathbf{r}}{r} - \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}\right) \tag{26e}$$

where

$$\Gamma_{n,m} = \Pi_{n,m+1}/\Pi_{n,m} = \sqrt{\frac{(n+m+1)(n-m)(2-\delta_{0m})}{2}} \tag{27}$$

since

$$\Pi_{n,m} = \sqrt{\frac{(n+m)!}{(n-m)!(2-\delta_{0m})(2n+1)}}. \tag{28}$$

In the software, $\Gamma_{n,m}$ is precomputed and stored at initialization.

The equations of motion are integrated in the inertial frame, so the spacecraft state vector must be converted from the inertial frame to the body-fixed frame before computing the accelerations. After computing the body-fixed accelerations, they must be converted back into the inertial frame. The conversion from an inertial to a fixed frame is described in detail for both the Moon and the Earth in Section 5.1.6.

5.1.6 Inertial-Fixed Conversions

Lunar Orientation The conversion between inertial and fixed position for the Moon is computed using the Lunar Librations from the DE403 (or DE405) ephemeris. The DE403 ephemeris is best for the Moon, and using Julian Ephemeris Date (JED) is technically more correct than using Terrestrial Time (TT) when interpolating the Lunar Librations. The Lunar Librations are three Euler angles, ϕ, ψ, and θ, according to Newhall and Williams (1997). ϕ is the rotation angle along the Earth's mean equator of J2000 from the equinox to the ascending node of the lunar equator. θ is the inclination of the lunar equator with respect to the Earth's mean equator. ψ is the angle along the lunar equator from the node to a longitude of zero. Figure 2 from Newhall and Williams shows the angles graphically. Make sure not to confuse the ϕ mentioned here with the ϕ that represents latitude. The conversion for velocity is included in the discussion, even though it is not used to compute acceleration due to the gravity field.

Using these angles from the JPL ephemeris, the conversion from the Moon-Centered inertial position and velocity (based on ICRF) to Moon-Centered Moon-Fixed (MCMF) would be

$$\begin{aligned} \mathbf{r}_{MCMF} &= \gamma \mathbf{r}_{ICRF} \\ \mathbf{v}_{MCMF} &= \gamma \mathbf{v}_{ICRF} + \dot{\gamma} r_{ICRF}. \end{aligned} \tag{29}$$

The rotation matrices $ROT1$, $ROT2$, and $ROT3$ are

$$ROT1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{30}$$

$$ROT2(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{31}$$

$$ROT3(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{32}$$

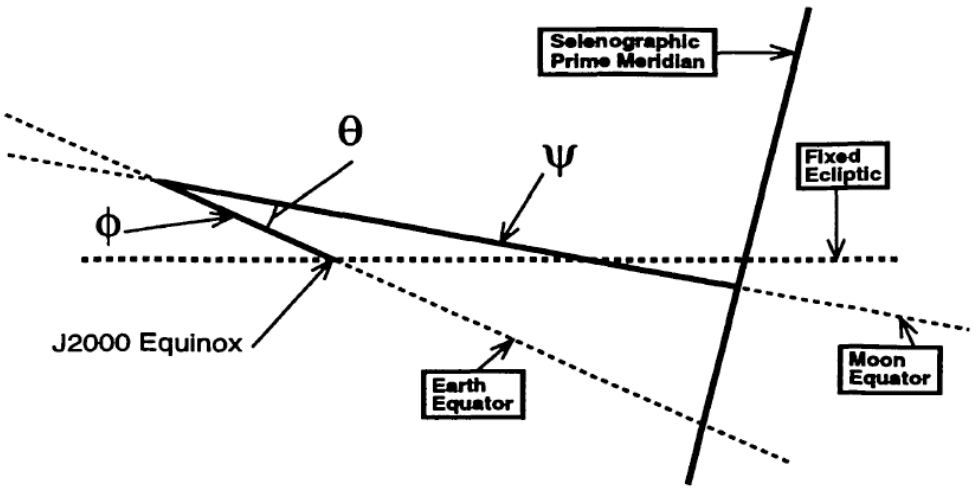

Figure 2: The equatorial system in which the Libration Eular angles are defined. (The value of ϕ would be negative as shown, credit: Newhall and Williams (1997).)

Using these rotation matrices, γ for the Moon is

$$\gamma = ROT3(\psi)ROT1(\theta)ROT3(\phi). \tag{33}$$

Multiplying the three rotation matrices together gives:

$$\gamma = \begin{bmatrix} C(\psi)C(\phi) - S(\psi)C(\theta)S(\phi) & C(\psi)S(\phi) + S(\psi)C(\theta)C(\phi) & S(\psi)S(\theta) \\ -S(\psi)C(\phi) - C(\psi)C(\theta)S(\phi) & -S(\psi)S(\phi) + C(\psi)C(\theta)C(\phi) & C(\psi)S(\theta) \\ S(\theta)S(\phi) & -S(\theta)C(\phi) & C(\theta) \end{bmatrix}, \tag{34}$$

where C and S represent sine and cosine functions, respectively.

The GEONS Mathematical Specification by Long and Lee (2006) gives the technique used to find $\dot{\gamma}$ in Eqn. 3.2-91a, although different angles are used here. The time derivative of γ is taken, assuming $\frac{d\phi}{dt} = 0$ and $\frac{d\theta}{dt} = 0$:

$$\dot{\gamma} = \dot{\psi} \begin{bmatrix} -S(\psi)C(\phi) - C(\psi)C(\theta)S(\phi) & -S(\psi)S(\phi) + C(\psi)C(\theta)C(\phi) & C(\psi)S(\theta) \\ -C(\psi)C(\phi) + S(\psi)C(\theta)S(\phi) & -C(\psi)S(\phi) - S(\psi)C(\theta)C(\phi) & -S(\psi)S(\theta) \\ 0 & 0 & 0 \end{bmatrix}. \tag{35}$$

Long and Lee (2006) give $\dot{\psi}$ a value of 0.2299708331 radians per day.

The conversion from Moon-Centered Moon-Fixed (MCMF) to Moon-Centered inertial (based on ICRF) would be:

$$\begin{aligned} \mathbf{r}_{ICRF} &= [\gamma]^T \mathbf{r}_{MCMF} \\ \mathbf{v}_{ICRF} &= [\gamma]^T \mathbf{v}_{MCMF} + [\dot{\gamma}]^T \mathbf{r}_{MCMF}. \end{aligned} \tag{36}$$

Earth Orientation The conversion from an Earth-centered inertial frame (GCRF) to an Earth-centered, Earth-fixed frame (ITRF) is performed using the 1976 IAU Precession, 1980 IAU Nutation (with IERS corrections), Earth rotation parameters, and Polar Motion. The conversion for velocity is included in the discussion, even though it is not used to compute acceleration due to the gravity field.

1976 IAU Precession To perform the conversion from inertial to fixed coordinates for the Earth, a position vector in the inertial frame, $\mathbf{r}_{GCRF}$, must first be rotated into a Mean of Date (MOD) inertial frame. This is done in TurboProp using the 1976 IAU Precession model given in Eqn. 3-56 in Vallado and McClain (2001).

$$\begin{aligned} \zeta &= 2306.2181" T_{TT} + 0.30188 T_{TT}^2 + 0.017998 T_{TT}^3 \\ \Theta &= 2004.3109" T_{TT} - 0.42665 T_{TT}^2 - 0.041833 T_{TT}^3 \\ z &= 2306.2181" T_{TT} + 1.09468 T_{TT}^2 + 0.018203 T_{TT}^3 \end{aligned} \tag{37}$$

The units must be changed to radians. T_{TT} is the number of Julian centuries from J2000 in Terrestrial Time (TT).

$$T_{TT} = \frac{JD_{TT} - 2451545.0}{36525} \tag{38}$$

The equations for ζ, Θ, and z are nominally written using T_{JED}, but T_{TT} can be used with no discernible change in results.

The conversion for position and velocity from GCRF to MOD is (Vallado and McClain (2001) Eqn. 3-57):

$$\begin{aligned} \mathbf{r}_{MOD} &= ROT3(-z)ROT2(\Theta)ROT3(-\zeta)\mathbf{r}_{GCRF} \\ \mathbf{v}_{MOD} &= ROT3(-z)ROT2(\Theta)ROT3(-\zeta)\mathbf{v}_{GCRF}. \end{aligned} \tag{39}$$

Note that this MOD frame is different than the MOD frame obtained when converting from the Mean J2000 inertial frame.

The three rotation matrices shown above are combined and called $[PRE]$:

$$[PRE] = ROT3(-z)ROT2(\Theta)ROT3(-\zeta). \tag{40}$$

1980 IAU Theory of Nutation + corrections Next, a conversion is performed from inertial MOD, $\mathbf{r}_{MOD}$, to inertial True of Date (TOD) coordinates, $\mathbf{r}_{TOD}$. This is done in TurboProp using the 1980 IAU Theory of Nutation.

First, compute the mean obliquity (Vallado and McClain (2001) Eqn. 3-52):

$$\bar{\epsilon} = 23.439291^\circ - 0.0130042 T_{TT} - 1.64 \times 10^{-7} T_{TT}^2 + 5.04 \times 10^{-7} T_{TT}^3. \tag{41}$$

The nutation in longitude, $\Delta\Psi$, and the nutation in obliquity, $\Delta\epsilon$ are normally computed from a series of 106 terms. However, TurboProp uses the interpolated nutation parameters from the JPL DE ephemerides instead of computing all 106 terms in the trigonometric series. Due to errors in 1976 IAU Precession and 1980 IAU Nutation, the International Earth Rotation Service (IERS) publishes corrections to the nutation parameters called $\partial\Delta\Psi_{1980}$ and $\partial\Delta\epsilon_{1980}$. These corrections also account for

differences between the GCRF frame used in TurboProp and the mean J2000 frame normally used with the 1976 IAU Precession and 1980 IAU Nutation models. These nutation correction parameters are computed from the `extras.EOP` vector (see Section 4.15.11).

$$\begin{aligned} \Delta\Psi_{1980} &= \Delta\Psi + \partial\Delta\Psi_{1980} \\ \Delta\epsilon_{1980} &= \Delta\epsilon + \partial\Delta\epsilon_{1980} \\ \epsilon &= \bar{\epsilon} + \Delta\epsilon_{1980} \end{aligned} \tag{42}$$

The conversion from MOD to TOD is:

$$\begin{aligned} \mathbf{r}_{TOD} &= ROT1(-\epsilon)ROT3(-\Delta\Psi_{1980})ROT1(\bar{\epsilon})\mathbf{r}_{MOD} \\ \mathbf{v}_{TOD} &= ROT1(-\epsilon)ROT3(-\Delta\Psi_{1980})ROT1(\bar{\epsilon})\mathbf{v}_{MOD}. \end{aligned} \tag{43}$$

The rotation $ROT1(-\epsilon)ROT3(-\Delta\Psi_{1980})ROT1(\bar{\epsilon})$ is called $[NUT]$.

Earth Rotation or Sidereal Time This conversion accounts for the instantaneous orientation of the Earth about its rotation axis. First, Greenwich Mean Sidereal Time, θ_{GMST}, must be computed (Vallado and McClain (2001) Eqn. 3-44).

$$\theta_{GMST} = \theta_{GMST0h} + \omega_{\oplus prec}UT1 \tag{44}$$

The $UT1$ in the equation is the time past midnight in UT1. θ_{GMST0h} is the Greenwich Mean Sidereal Time at midnight, UT1,0h, or 00:00:00.0 UT1 on the date of interest. θ_{GMST0h} is computed using the equation (Vallado and McClain (2001) Eqn. 3-43)

$$\theta_{GMST0h} = 100.4606184^\circ + 36000.77005361T_{UT1,0h} + .00038793T^2_{UT1,0h} - 2.6\times10^{-8}T^3_{UT1,0h}. \tag{45}$$

The units must be changed to radians. $T_{UT1,0h}$ is the number of Julian Centuries elapsed from J2000 to 00:00:00.0 UT1 on the date of interest.

$$T_{UT1,0h} = \frac{JD_{UT1,0h} - 2451545.0}{36525} \tag{46}$$

$JD_{UT1,0h}$ is the Julian Date at 00:00:00.0 UT1 for the date of interest.

JD_{UT1} is computed from JD_{TT} in the following way:

$$JD_{UT1} = JD_{TT} + (-32.184 + [UT1 - UTC] - N_{leap})/86400. \tag{47}$$

where -32.184 is the conversion from TT to TAI, N_{leap} is the number of leap seconds needed to convert from TAI to UTC, and $[UT1 - UTC]$ is the conversion from UTC to UT1. N_{leap} and $[UT1 - UTC]$ are computed from the EOP parameters in `extras.EOP` (see Section 4.15.11).

$\omega_{\oplus prec}$ is the Earth's mean angular rotation, and can be computed using Eqn. 3-38 in Vallado and McClain (2001), which gives revolutions per day.

$$\omega_{\oplus prec} = 1.002737909350795 + 5.9006\times10^{-11}T_{TT} - 5.9\times10^{-15}T^2_{TT} \tag{48}$$

The units must be converted to radians, and T_{TT} is the number of Julian centuries (TT) elapsed since J2000. The equation is nominally written using T_{UT1}, but T_{TT} can be used with no discernible change in results.

After computing θ_{GMST}, it is necessary to compute the Greenwich Apparent Sidereal Time, θ_{AST}, using the equation of the equinoxes (Vallado and McClain (2001) Eqn. 3-65).

$$EQ_{equinox} = \Delta\Psi \cos(\bar{\epsilon}) + 0.00264\text{"} \sin(\Omega_{\leftmoon}) + 0.000063 \sin(2\Omega_{\leftmoon}) \quad (49)$$

$$\theta_{AST} = \theta_{GMST} + EQ_{equinox} \quad (50)$$

$\Omega_{\leftmoon}$ can be found using Eqn. 3-54 in Vallado and McClain (2001).

$$\Omega_{\leftmoon} = 125.04455501^\circ - 1934.1361851 T_{TT} + .0020756 T_{TT}^2 + 2.139 \times 10^{-6} T_{TT}^3 - 1.650 \times 10^{-8} T_{TT}^4 \quad (51)$$

The conversion from the TOD to a Pseudo-Earth-fixed (PEF) is (Vallado and McClain (2001) Eqn. 3-70)

$$\mathbf{r}_{PEF} = ROT3(\theta_{AST}) \mathbf{r}_{TOD}. \quad (52)$$

$ROT3(\theta_{AST})$ will be called $[ST]$. For the velocity, the rotation rate of the Earth must be computed using the excess Length Of Day (LOD) (Vallado and McClain (2001) Eqn. 3-69).

$$\omega_{\oplus} = 7.29211514670698 \times 10^{-5} \, (1 - LOD) \quad (53)$$

LOD is a parameter in the `extras.EOP` vector (see Section 4.15.11) and has units of days. For units of seconds, then the term in the equation above must be changed to $LOD/86400$. Using $\omega_{\oplus}$, the conversion matrix $[\dot{ST}]$ is

$$[\dot{ST}] = \begin{bmatrix} -\omega_{\oplus} \sin(\theta_{AST}) & \omega_{\oplus} \cos(\theta_{AST}) & 0 \\ -\omega_{\oplus} \cos(\theta_{AST}) & -\omega_{\oplus} \sin(\theta_{AST}) & 0 \\ 0 & 0 & 0 \end{bmatrix}. \quad (54)$$

Finally, the velocity is transformed from TOD to PEF.

$$\mathbf{v}_{PEF} = [ST] \mathbf{v}_{TOD} + [\dot{ST}] \mathbf{r}_{TOD} \quad (55)$$

Polar Motion To convert from PEF to the Earth-fixed International Terrestrial Reference Frame (ITRF), a polar motion rotation must be applied. x_p and y_p are the coordinates describing the motion of the Earth's rotation axis with respect to the Earth's crust. These polar motion parameters are computed from the `extras.EOP` vector (see Section 4.15.11). Using x_p and y_p, the conversion from PEF to ITRF is (Vallado and McClain (2001) Eqn. 3-68)

$$\mathbf{r}_{ITRF} = \begin{bmatrix} 1 & 0 & x_p \\ 0 & 1 & -y_p \\ -x_p & y_p & 1 \end{bmatrix} \mathbf{r}_{PEF}. \quad (56)$$

The matrix above will be called $[PM]$, and the velocity conversion is

$$\mathbf{v}_{ITRF} = [PM] \mathbf{v}_{PEF}. \quad (57)$$

To summarize the conversion from GCRF to ITRF:

$$\begin{aligned} \mathbf{r}_{ITRF} &= [PM][ST][NUT][PREC]\mathbf{r}_{GCRF} \\ \mathbf{v}_{ITRF} &= [PM][\dot{ST}][NUT][PREC]\mathbf{r}_{GCRF} + [PM][ST][NUT][PREC]\mathbf{v}_{GCRF}. \end{aligned} \tag{58}$$

Let γ be $[PM][ST][NUT][PREC]$ and let $\dot{\gamma}$ be $[PM][\dot{ST}][NUT][PREC]$, and the notation is shortened and the conversion is

$$\begin{aligned} \mathbf{r}_{ITRF} &= [\gamma]\mathbf{r}_{GCRF} \\ \mathbf{v}_{ITRF} &= [\dot{\gamma}]\mathbf{r}_{GCRF} + [\gamma]\mathbf{v}_{GCRF}. \end{aligned} \tag{59}$$

Converting back to the inertial frame (GCRF) would be

$$\begin{aligned} \mathbf{r}_{GCRF} &= [\gamma]^T\mathbf{r}_{ITRF} \\ \mathbf{v}_{GCRF} &= [\dot{\gamma}]^T\mathbf{r}_{ITRF} + [\gamma]^T\mathbf{v}_{ITRF}. \end{aligned} \tag{60}$$

5.1.7 Solar Radiation Pressure, "_SRP"

A solar radiation pressure model is included in the force model for the satellite. This is based on a constant area, constant reflectance model. The reflectance is also included in the spacecraft state vector. The shadow model determines an approximate value of the percentage of the Sun's face visible at the spacecraft location. The radii of the bodies used in the shadow model are taken from Vallado and McClain (2001) or the JPL ephemeris. The value of solar radiation pressure is adjusted based on distance from the Sun.

The acceleration due to SRP is

$$\ddot{\mathbf{r}}_{SRP} = p_{SR}\, c_R \frac{A_\odot}{m} \frac{\mathbf{r}_{\odot sat}}{|\mathbf{r}_{\odot sat}|}. \tag{61}$$

c_R is the reflectivity of the spacecraft. $A_\odot$ is the cross-sectional area of the spacecraft facing the Sun in square meters. m is the mass of the spacecraft in kilograms. $\mathbf{r}_{\odot sat}$ is the vector from the center of the Sun to the spacecraft. p_{SR} is the pressure of solar radiation in Pa. This is about 4.53×10^{-6} Pa at one AU, and is found by dividing the solar constant (flux in W/m^2) by the speed of light (p. 544 Vallado and McClain (2001)).

$$p_{SR,AU} = \frac{1358 \quad W/m^2}{299,792,458 m/s} \tag{62}$$

For varying distances from the Sun, p_{SR} would be

$$p_{SR} = p_{SR,AU} \frac{(149,597,870\, km)^2}{|\mathbf{r}_{\odot sat}|^2}. \tag{63}$$

$\mathbf{r}_{\odot sat}$ should be in km. A factor ℓ can be applied to represent the fraction of the Sun's face that is visible. When km and km/s are the units used for position and velocity, respectively, the acceleration due to SRP is

$$\ddot{r}_{SRP} = \left[c_R\, \ell \frac{A_\odot}{m} (149,597,870\,\text{km})^2\, p_{SR,AU} \frac{1\,\text{km}}{1000\,\text{m}} \right] \frac{\mathbf{r}_{\odot sat}}{|\mathbf{r}_{\odot sat}|^3}. \tag{64}$$

$\ell = 1$ when there is a clear line of sight vector between the spacecraft and the Sun, and $\ell = 0$ when the Sun's light is blocked by a body. When determining the line of sight, it was assumed that the center planet (`extras.center`) is the only body that would eclipse the Sun. The angle between the center of the planet and the center of the Sun is called ς and is found using the dot product.

$$\varsigma = \cos^{-1}\left(\frac{\mathbf{r}_{\odot sat} \cdot \mathbf{r}_{\leftmoon\ sat}}{|r_{\odot sat}||r_{\leftmoon\ sat}|}\right) \tag{65}$$

$\mathbf{r}_{\odot sat}$ is the vector from the center of the Sun to the spacecraft and $\mathbf{r}_{\leftmoon\ sat}$ is from the central body to the spacecraft. $|r_{\odot sat}|$ and $|r_{\leftmoon\ sat}|$ are the magnitudes of those vectors. The angle between the center and limb for the two bodies are found using these equations, assuming that the bodies are spherical.

$$\varsigma_{\odot} = \sin^{-1}\left(\frac{R_{\odot}}{|r_{\odot sat}|}\right) \tag{66}$$

$$\varsigma_{\leftmoon} = \sin^{-1}\left(\frac{R_{\leftmoon}}{|r_{\leftmoon\ sat}|}\right) \tag{67}$$

$R_{\odot}$ and $R_{\leftmoon}$ are the radii of the primary bodies. Table 6 shows the values TurboProp uses for the planet radii. Note that `extras.radius` does not override these values for eclipse computations. Most radius values were taken from Appendix D of Vallado and McClain (2001), but the radii for Mercury, Venus, and Mars were taken from the DE405 ephemeris. "Earth+Moon" represents the Earth/Moon barycenter instead of an eclipsing body, so it is given a radius of zero.

Table 6: PLANETARY RADII FOR ECLIPSE COMPUTATIONS.

Planet	radius (km)
Mercury	2439.76
Venus	6052.3
Earth+Moon	0
Mars	3397.515
Jupiter	71,492.0
Saturn	60,268.0
Uranus	25,559.0
Neptune	24,764.0
Pluto	1151.0
Sun	696,000.0
Earth	6378.1363
Moon	1738.0

Units are km^3/s^2

The relations below were used in determining ℓ.

$$\ell = \begin{cases} 1, & \varsigma \geq \varsigma_{\leftmoon} + \varsigma_{\odot} \\ \dfrac{\varsigma - \varsigma_{\leftmoon}}{2\varsigma_{\odot}} + \frac{1}{2}, & \varsigma_{\leftmoon} - \varsigma_{\odot} \leq \varsigma < \varsigma_{\leftmoon} + \varsigma_{\odot} \\ 0, & \varsigma < \varsigma_{\leftmoon} - \varsigma_{\odot} \end{cases} \tag{68}$$

This is based on the assumption that $\varsigma_{\leftmoon} \gg \varsigma_{\odot}$. If that is false, it only negatively affects the computation of ℓ when the Sun is partially eclipsed.

5.1.8 Atmospheric Drag, "_drag"

The acceleration due to drag for the `TwoBody`-type derivative functions is computed using the equations

$$\begin{aligned} \ddot{x}_{drag} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,x}|V_a| \\ \ddot{y}_{drag} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,y}|V_a| \\ \ddot{z}_{drag} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,z}|V_a|, \end{aligned} \tag{69}$$

where C_D is the coefficient of drag, which is part of the state vector. $\frac{A}{m}$ is the area-to-mass ratio of the spacecraft, specified by `extras.A_m`. $V_{a,x}$, $V_{a,y}$, and $V_{a,z}$ are the velocity components of the spacecraft with respect to the atmosphere (assumed to rotate with the planet). $|V_a|$ is the magnitude of that vector.

The atmospheric density, ρ, at a certain radius from the center of the body, r, is computed using the parameters in `extras.atmos`.

$$\rho(r) = \rho_0 e^{\left(\frac{r_0 - r}{H}\right)} \tag{70}$$

See Section 4.15.12 for more information.

In `TwoBody_drag`-type derivative functions, The atmospheric-relative velocity vector is computed from the inertial velocities ($\dot{x}$, $\dot{y}$, and $\dot{z}$) and positions (x and y) using the equations

$$\begin{aligned} V_{a,x} &= \dot{x} + \dot{\theta}y \\ V_{a,y} &= \dot{y} - \dot{\theta}x \\ V_{a,z} &= \dot{z}. \end{aligned} \tag{71}$$

$\dot{\theta}$ is the rotation rate of the central body and is contained in `extras.EOP` (see Section 4.15.11).

The acceleration due to drag for the `DE`-type derivative functions is computed using the equations

$$\begin{aligned} \ddot{x}_{drag} &= -500C_D\frac{A}{m}\rho V_{a,x}|V_a| \\ \ddot{y}_{drag} &= -500C_D\frac{A}{m}\rho V_{a,y}|V_a| \\ \ddot{z}_{drag} &= -500C_D\frac{A}{m}\rho V_{a,z}|V_a|. \end{aligned} \tag{72}$$

The 500 coefficient is the conversion factor 1000 divided by two. The conversion factor is needed because the units for $\frac{A}{m}$ are m^2/kg, the units for ρ are kg/m^3, and the units for V_a are km/s.

In `DE_drag`-type derivative functions, The atmospheric-relative velocity vector is computed using the equation (Vallado and McClain (2001) p. 523)

$$\mathbf{V}_a = \dot{\mathbf{r}} - \omega \times \mathbf{r}, \tag{73}$$

where $\dot{\mathbf{r}}$ is the inertial velocity vector, $\mathbf{r}$ is the inertial position vector, and ω is the Earth's rotation vector.

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \omega_\oplus \begin{bmatrix} \gamma(3,1) \\ \gamma(3,2) \\ \gamma(3,3) \end{bmatrix} \tag{74}$$

$\omega_\oplus$ is the rotation rate of the Earth, and it is multiplied by the third row of the γ matrix used to convert a position from inertial to fixed (see Section 5.1.6).

Applying the cross product gives

$$\begin{aligned} V_{a,x} &= \dot{x} - \omega_2 z + \omega_3 y \\ V_{a,y} &= \dot{y} - \omega_3 x + \omega_1 z \\ V_{a,z} &= \dot{z} - \omega_1 y + \omega_2 x. \end{aligned} \tag{75}$$

For `DE_drag`-type functions, $\omega_\oplus$ is computed automatically, and is not supplied by the user.

5.1.9 State Transition Matrix, "_stm"

In `stm`-type derivative functions, the state vector is augmented with a state transition matrix (STM). When the state vector includes a state transition matrix (STM), it should be added on to the end of the state vector in column order. As an example assignment in MATLAB with a six-element state and a STM, you could type

```
y0 = [x y z dx dy dz]';                    (this assigns the state)
y0(6+1:6+36) = reshape(eye(6),36,1);       (this adds on the STM)
```

State deviations at time t may be mapped back to the initial epoch, t_k, with the state transition matrix Φ.

$$\mathbf{x}(t) = \Phi(t, t_k)\mathbf{x}_k \tag{76}$$

A sample state transition matrix for two-dimensions of position and velocity would be

$$\Phi(t, t_k) = \begin{bmatrix} \frac{\partial x}{\partial x_k} & \frac{\partial x}{\partial y_k} & \frac{\partial x}{\partial \dot{x}_k} & \frac{\partial x}{\partial \dot{y}_k} \\ \frac{\partial y}{\partial x_k} & \frac{\partial y}{\partial y_k} & \frac{\partial y}{\partial \dot{x}_k} & \frac{\partial y}{\partial \dot{y}_k} \\ \frac{\partial \dot{x}}{\partial x_k} & \frac{\partial \dot{x}}{\partial y_k} & \frac{\partial \dot{x}}{\partial \dot{x}_k} & \frac{\partial \dot{x}}{\partial \dot{y}_k} \\ \frac{\partial \dot{y}}{\partial x_k} & \frac{\partial \dot{y}}{\partial y_k} & \frac{\partial \dot{y}}{\partial \dot{x}_k} & \frac{\partial \dot{y}}{\partial \dot{y}_k} \end{bmatrix}. \tag{77}$$

The state transition matrix is integrated along with the state using the following relation:

$$\dot{\Phi}(t,t_k) = A(t)\Phi(t,t_k) \qquad \text{where} \qquad A(t) = \frac{\partial \mathbf{F}(\mathbf{X},t)}{\partial \mathbf{X}(t)}, \qquad \text{and is of the form:} \tag{78}$$

$$A = \begin{bmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial y} & \frac{\partial \dot{x}}{\partial z} & \cdots & \frac{\partial \dot{x}}{\partial \dot{z}} \\ \frac{\partial \dot{y}}{\partial x} & \frac{\partial \dot{y}}{\partial y} & \frac{\partial \dot{y}}{\partial z} & & \\ \frac{\partial \dot{z}}{\partial x} & \frac{\partial \dot{z}}{\partial y} & \frac{\partial \dot{z}}{\partial z} & & \vdots \\ \vdots & & & \ddots & \\ \frac{\partial \ddot{z}}{\partial x} & & \cdots & & \frac{\partial \ddot{z}}{\partial \dot{z}} \end{bmatrix}. \tag{79}$$

Although it isn't shown, the parameters C_D and c_R would also be included in the state transition matrix if they were in the state vector.

The following sections describe how these partials are computed.

Central Body STM To include the two-body acceleration in the variational equations used to create matrix A, the partials of the two-body portion of $\ddot{\mathbf{r}}$ with respect to $\mathbf{r}$ are computed in this manner:

$$\begin{array}{llllll} \frac{\partial \ddot{x}_{2body}}{\partial x} = & 3\frac{\mu x^2}{r^5} - \frac{\mu}{r^3} & \frac{\partial \ddot{x}_{2body}}{\partial y} = & 3\frac{\mu xy}{r^5} & \frac{\partial \ddot{x}_{2body}}{\partial z} = & 3\frac{\mu xz}{r^5} \\ \frac{\partial \ddot{y}_{2body}}{\partial x} = & 3\frac{\mu xy}{r^5} & \frac{\partial \ddot{y}_{2body}}{\partial y} = & 3\frac{\mu y^2}{r^5} - \frac{\mu}{r^3} & \frac{\partial \ddot{y}_{2body}}{\partial z} = & 3\frac{\mu yz}{r^5} \\ \frac{\partial \ddot{z}_{2body}}{\partial x} = & 3\frac{\mu xz}{r^5} & \frac{\partial \ddot{z}_{2body}}{\partial y} = & 3\frac{\mu yz}{r^5} & \frac{\partial \ddot{z}_{2body}}{\partial z} = & 3\frac{\mu z^2}{r^5} - \frac{\mu}{r^3} \end{array}, \tag{80}$$

where x, y, and z are the inertial position vector from the center of mass of the body to the spacecraft and r is the magnitude of that vector. μ is the gravitational parameter of the central body. $\ddot{x}$, $\ddot{y}$, and $\ddot{z}$ are the accelerations in the x, y, and z-directions, respectively.

Third Body STM To include the third-body acceleration in the variational equations used to create matrix A, the partials of the third-body portion of $\ddot{\mathbf{r}}$ with respect to $\mathbf{r}$ are computed in this manner:

$$\begin{array}{llllll} \frac{\partial \ddot{x}_{3body}}{\partial x_3} = & 3\frac{\mu_3 x_3^2}{r_3^5} - \frac{\mu_3}{r_3^3} & \frac{\partial \ddot{x}_{3body}}{\partial y_3} = & 3\frac{\mu_3 x_3 y_3}{r_3^5} & \frac{\partial \ddot{x}_{3body}}{\partial z_3} = & 3\frac{\mu_3 x_3 z_3}{r_3^5} \\ \frac{\partial \ddot{y}_{3body}}{\partial x_3} = & 3\frac{\mu_3 x_3 y_3}{r_3^5} & \frac{\partial \ddot{y}_{3body}}{\partial y_3} = & 3\frac{\mu_3 y_3^2}{r_3^5} - \frac{\mu_3}{r_3^3} & \frac{\partial \ddot{y}_{3body}}{\partial z_3} = & 3\frac{\mu_3 y_3 z_3}{r_3^5} \\ \frac{\partial \ddot{z}_{3body}}{\partial x_3} = & 3\frac{\mu_3 x_3 z_3}{r_3^5} & \frac{\partial \ddot{z}_{3body}}{\partial y_3} = & 3\frac{\mu_3 y_3 z_3}{r_3^5} & \frac{\partial \ddot{z}_{3body}}{\partial z_3} = & 3\frac{\mu_3 z_3^2}{r_3^5} - \frac{\mu_3}{r_3^3} \end{array}. \tag{81}$$

x_3, y_3, and z_3 are the inertial position vector from the center of mass of the third-body to the spacecraft and r_3 is the magnitude of that vector. μ_3 is the gravitational parameter of the third-body. These computations are repeated for each third-body included in the equations of motion. Notice that this is the same as for the central body.

CRTBP STM For the Circular-restricted Three-body Problem, the partials used to create matrix A are

$$
\begin{aligned}
\frac{\partial \ddot{x}_{CRTBP}}{\partial x} &= 1 + \frac{3\mu(\mu-1+x)^2}{r_2^5} - r_2^3 + \frac{3(1-\mu)(\mu+x)^2}{r_1^5} + \frac{\mu-1}{r_1^3} \\
\frac{\partial \ddot{x}_{CRTBP}}{\partial y} &= 3y\left(\frac{\mu(\mu-1+x)}{r_2^5} + \frac{(1-\mu)(\mu+x)}{r_1^5}\right) \\
\frac{\partial \ddot{x}_{CRTBP}}{\partial z} &= 3z\left(\frac{\mu(\mu-1+x)}{r_2^5} + \frac{(1-\mu)(\mu+x)}{r_1^5}\right) \\
\frac{\partial \ddot{x}_{CRTBP}}{\partial \dot{y}} &= 2 \\
\frac{\partial \ddot{y}_{CRTBP}}{\partial x} &= \frac{\partial \ddot{x}_{CRTBP}}{\partial y} \\
\frac{\partial \ddot{y}_{CRTBP}}{\partial y} &= 1 + \frac{3\mu y^2}{r_2^5} - r_2^3 + \frac{3(1-\mu)y^2}{r_1^5} + \frac{\mu-1}{r_1^3} \\
\frac{\partial \ddot{y}_{CRTBP}}{\partial z} &= 3yz\left(\frac{\mu}{r_2^5} + \frac{1-\mu}{r_1^5}\right) \\
\frac{\partial \ddot{y}_{CRTBP}}{\partial \dot{x}} &= -2 \\
\frac{\partial \ddot{z}_{CRTBP}}{\partial x} &= \frac{\partial \ddot{x}_{CRTBP}}{\partial z} \\
\frac{\partial \ddot{z}_{CRTBP}}{\partial y} &= \frac{\partial \ddot{y}_{CRTBP}}{\partial z} \\
\frac{\partial \ddot{z}_{CRTBP}}{\partial z} &= \frac{3\mu z^2}{r_2^5} - r_2^3 + \frac{3(1-\mu)z^2}{r_1^5} + \frac{\mu-1}{r_1^3}.
\end{aligned}
\tag{82}
$$

See Section 5.1.2 for details on the notation.

For the Circular-restricted Three-body Problem centered at the secondary, with the $\tilde{x}$ axis pointed toward the primary, the partials used to create matrix A are

$$
\begin{aligned}
\frac{\partial \ddot{\tilde{x}}_{CRTBP}}{\partial \tilde{x}} &= 1 - \frac{1-\mu}{r_1^3} + 3\left(\frac{(1-\mu)(\tilde{x}-1)^2}{r_1^5} + \frac{\mu\tilde{x}^2}{r_2^5}\right) - \frac{\mu}{r_2^3} \\
\frac{\partial \ddot{\tilde{x}}_{CRTBP}}{\partial \tilde{y}} &= 3\tilde{y}\left(\frac{(1-\mu)(\tilde{x}-1)}{r_1^5} + \frac{\mu\tilde{x}}{r_2^5}\right) \\
\frac{\partial \ddot{\tilde{x}}_{CRTBP}}{\partial \tilde{z}} &= 3\tilde{z}\left(\frac{(1-\mu)(\tilde{x}-1)}{r_1^5} + \frac{\mu\tilde{x}}{r_2^5}\right) \\
\frac{\partial \ddot{\tilde{x}}_{CRTBP}}{\partial \dot{y}} &= 2 \\
\frac{\partial \ddot{\tilde{y}}_{CRTBP}}{\partial \tilde{x}} &= \frac{\partial \ddot{\tilde{x}}_{CRTBP}}{\partial \tilde{y}} \\
\frac{\partial \ddot{\tilde{y}}_{CRTBP}}{\partial \tilde{y}} &= 1 - \frac{1-\mu}{r_1^3} + 3\tilde{y}^2\left(\frac{(1-\mu)}{r_1^5} + \frac{\mu}{r_2^5}\right) - \frac{\mu}{r_2^3} \\
\frac{\partial \ddot{\tilde{y}}_{CRTBP}}{\partial \tilde{z}} &= 3\tilde{y}\tilde{z}\left(\frac{(1-\mu)}{r_1^5} + \frac{\mu\tilde{x}}{r_2^5}\right) \\
\frac{\partial \ddot{\tilde{y}}_{CRTBP}}{\partial \dot{x}} &= -2 \\
\frac{\partial \ddot{\tilde{z}}_{CRTBP}}{\partial \tilde{x}} &= \frac{\partial \ddot{\tilde{x}}_{CRTBP}}{\partial \tilde{z}} \\
\frac{\partial \ddot{\tilde{z}}_{CRTBP}}{\partial \tilde{y}} &= \frac{\partial \ddot{\tilde{y}}_{CRTBP}}{\partial \tilde{z}} \\
\frac{\partial \ddot{\tilde{z}}_{CRTBP}}{\partial \tilde{z}} &= -\frac{1-\mu}{r_1^3} + 3\tilde{z}^2\left(\frac{(1-\mu)}{r_1^5} + \frac{\mu}{r_2^5}\right) - \frac{\mu}{r_2^3}.
\end{aligned}
\tag{83}
$$

See Section 5.1.2 for details on the notation.

Gravity Field STM To include the gravity field in the variational equations used to create the matrix A, the partial of the gravity field portion of $\ddot{\mathbf{r}}$ with respect to $\mathbf{r}$ is computed. These variational equations were derived in Gottlieb (1993), and are provided here as a reference. The resulting matrix is

$$
\frac{\partial \ddot{\mathbf{r}}}{\partial \mathbf{r}} = \frac{\mu}{r^3}\left\{ \begin{bmatrix} \hat{\mathbf{r}} & \hat{k} \end{bmatrix} \begin{bmatrix} F & G \\ G & M \end{bmatrix} \begin{bmatrix} \hat{\mathbf{r}}^T \\ \hat{k}^T \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{r}} & \mathbf{d} \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{r}}^T \\ \mathbf{d}^T \end{bmatrix} + \begin{bmatrix} N-\Lambda & -\Omega & Q \\ -\Omega & -(N+\Lambda) & R \\ Q & R & -\Lambda \end{bmatrix} \right\}
\tag{84}
$$

where

$$\begin{aligned}
\epsilon &= \frac{z}{r} && (85)\\
\hat{k} &= \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T && (86)\\
\mathbf{d} &= \epsilon\begin{bmatrix}Q & R & 0\end{bmatrix}^T + \begin{bmatrix}S & T & 0\end{bmatrix}^T && (87)\\
\Lambda &= (\epsilon a_3 + a_4) && (88)\\
F &= L + \epsilon(M\epsilon + 2(P + a_3)) + \Lambda && (89)\\
G &= -(M\epsilon + P + a_3). && (90)
\end{aligned}$$

Additionally,

$$\begin{aligned}
L &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=0}^{n}(n+m+1)(n+m+2)\bar{A}_{n,m}(\bar{C}_{n,m}\bar{E}_m + \bar{S}_{n,m}\bar{F}_m) && (91)\\
M &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=0}^{n}\bar{A}_{n,m+2}\Psi_{n,m}(\bar{C}_{n,m}\bar{E}_m + \bar{S}_{n,m}\bar{F}_m) && (92)\\
N &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=2}^{n}m(m-1)\bar{A}_{n,m}(\bar{C}_{n,m}\bar{E}_{m-2} + \bar{S}_{n,m}\bar{F}_{m-2}) && (93)\\
\Omega &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=2}^{n}m(m-1)\bar{A}_{n,m}(\bar{C}_{n,m}\bar{F}_{m-2} - \bar{S}_{n,m}\bar{E}_{m-2}) && (94)\\
P &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=0}^{n}(n+m+1)\bar{A}_{n,m+1}\Gamma_{n,m}(\bar{C}_{n,m}\bar{E}_m + \bar{S}_{n,m}\bar{F}_m) && (95)\\
Q &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=1}^{n}m\bar{A}_{n,m+1}\Gamma_{n,m}(\bar{C}_{n,m}\bar{E}_{m-1} + \bar{S}_{n,m}\bar{F}_{m-1}) && (96)\\
R &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=1}^{n}m\bar{A}_{n,m+1}\Gamma_{n,m}(\bar{S}_{n,m}\bar{E}_{m-1} - \bar{C}_{n,m}\bar{F}_{m-1}) && (97)\\
S &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=1}^{n}(n+m+1)m\bar{A}_{n,m}(\bar{C}_{n,m}\bar{E}_{m-1} + \bar{S}_{n,m}\bar{F}_{m-1}) && (98)\\
T &= \sum_{n=2}^{\infty}\left(\frac{R}{r}\right)^n\sum_{m=1}^{n}(n+m+1)m\bar{A}_{n,m}(\bar{S}_{n,m}\bar{E}_{m-1} - \bar{C}_{n,m}\bar{F}_{m-1}). && (99)
\end{aligned}$$

Note the equation for S corrects a mistake in Gottlieb (1993). Finally,

$$\Psi_{n,m} = \Pi_{n,m+2}/\Pi_{n,m} = \sqrt{\frac{(n+m+1)(n+m+2)(n-m)(n-m-1)(2-\delta_{m,0})}{2}}. \quad (100)$$

Once the matrix $\dfrac{\partial\ddot{\mathbf{r}}}{\partial\mathbf{r}}$ is computed, it must be rotated from the body-fixed frame to the inertial frame before being added to the rest of the A matrix. This is done with the rotation γ described earlier as

shown in Eqn. 4-54 in Long et al. (1989). The inertial version of $\frac{\partial \ddot{\mathbf{r}}}{\partial \mathbf{r}}$ is called $\frac{\partial \ddot{\mathbf{r}}_{GCRF}}{\partial \mathbf{r}_{GCRF}}$ in the example below:

$$\left[\frac{\partial \ddot{\mathbf{r}}_{GCRF}}{\partial \mathbf{r}_{GCRF}}\right]_{3\times 3} = \gamma^T \left[\frac{\partial \ddot{\mathbf{r}}}{\partial \mathbf{r}}\right]_{3\times 3} \gamma. \tag{101}$$

$\frac{\partial \ddot{\mathbf{r}}_{GCRF}}{\partial \mathbf{r}_{GCRF}}$ can then be added to the proper portion of the A matrix.

Drag STM For the `TwoBody_drag`-type derivative functions. The variational equations used to create matrix A have contributions due to drag as shown below:

$$\begin{aligned}
\frac{\partial \ddot{x}_{drag}}{\partial x} &= \frac{C_D}{2}\frac{A}{m}\rho V_{a,x}\left[\frac{|V_a|x}{rH} + \dot{\theta}\frac{V_{a,y}}{|V_a|}\right] \\
\frac{\partial \ddot{x}_{drag}}{\partial y} &= \frac{C_D}{2}\frac{A}{m}\rho\left[V_{a,x}\frac{|V_a|y}{rH} - \dot{\theta}\left(\frac{V_{a,x}^2}{|V_a|} + |V_a|\right)\right] \\
\frac{\partial \ddot{x}_{drag}}{\partial z} &= \frac{C_D}{2}\frac{A}{m}\rho V_{a,x}\frac{|V_a|z}{rH} \\
\frac{\partial \ddot{x}_{drag}}{\partial \dot{x}} &= -\frac{C_D}{2}\frac{A}{m}\rho\left[\frac{V_{a,x}^2}{|V_a|} + |V_a|\right] \\
\frac{\partial \ddot{x}_{drag}}{\partial \dot{y}} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,x}\frac{V_{a,y}}{|V_a|} \\
\frac{\partial \ddot{x}_{drag}}{\partial \dot{z}} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,x}\frac{V_{a,z}}{|V_a|} \\
\frac{\partial \ddot{x}_{drag}}{\partial C_D} &= -\frac{1}{2}\frac{A}{m}\rho V_{a,x}|V_a|
\end{aligned} \tag{102}$$

$$\begin{aligned}
\frac{\partial \ddot{y}_{drag}}{\partial x} &= \frac{C_D}{2}\frac{A}{m}\rho\left[V_{a,y}\frac{|V_a|x}{rH} + \dot{\theta}\left(\frac{V_{a,y}^2}{|V_a|} + |V_a|\right)\right] \\
\frac{\partial \ddot{y}_{drag}}{\partial y} &= \frac{C_D}{2}\frac{A}{m}\rho V_{a,y}\left[\frac{|V_a|y}{rH} - \dot{\theta}\frac{V_{a,x}}{|V_a|}\right] \\
\frac{\partial \ddot{y}_{drag}}{\partial z} &= \frac{C_D}{2}\frac{A}{m}\rho V_{a,y}\frac{|V_a|z}{rH} \\
\frac{\partial \ddot{y}_{drag}}{\partial \dot{x}} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,y}\frac{V_{a,x}}{|V_a|} \\
\frac{\partial \ddot{y}_{drag}}{\partial \dot{y}} &= -\frac{C_D}{2}\frac{A}{m}\rho\left[\frac{V_{a,y}^2}{|V_a|} + |V_a|\right] \\
\frac{\partial \ddot{y}_{drag}}{\partial \dot{z}} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,y}\frac{V_{a,z}}{|V_a|} \\
\frac{\partial \ddot{y}_{drag}}{\partial C_D} &= -\frac{1}{2}\frac{A}{m}\rho V_{a,y}|V_a|
\end{aligned} \tag{103}$$

$$\begin{aligned}
\frac{\partial \ddot{z}_{drag}}{\partial x} &= \frac{C_D}{2}\frac{A}{m}\rho V_{a,z}\left[\frac{|V_a|x}{rH} + \dot{\theta}\frac{V_{a,y}}{|V_a|}\right] \\
\frac{\partial \ddot{z}_{drag}}{\partial y} &= \frac{C_D}{2}\frac{A}{m}\rho V_{a,z}\left[\frac{|V_a|y}{rH} - \dot{\theta}\frac{V_{a,x}}{|V_a|}\right] \\
\frac{\partial \ddot{z}_{drag}}{\partial z} &= \frac{C_D}{2}\frac{A}{m}\rho V_{a,z}\frac{|V_a|z}{rH} \\
\frac{\partial \ddot{z}_{drag}}{\partial \dot{x}} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,z}\frac{V_{a,x}}{|V_a|} \\
\frac{\partial \ddot{z}_{drag}}{\partial \dot{y}} &= -\frac{C_D}{2}\frac{A}{m}\rho V_{a,z}\frac{V_{a,y}}{|V_a|} \\
\frac{\partial \ddot{z}_{drag}}{\partial \dot{z}} &= -\frac{C_D}{2}\frac{A}{m}\left[\frac{V_{a,z}^2}{|V_a|} + |V_a|\right] \\
\frac{\partial \ddot{z}_{drag}}{\partial C_D} &= -\frac{1}{2}\frac{A}{m}\rho V_{a,z}|V_a|.
\end{aligned} \tag{104}$$

For the `DE_drag`-type derivative functions. The variational equations used to create matrix A have contributions due to drag as shown below:

$$\begin{aligned}
\frac{\partial \ddot{x}_{drag}}{\partial x} &= -500C_D\frac{A}{m}\rho V_{a,x}\left[-\frac{|V_a|x}{rH} + \frac{\omega_2 V_{a,z} - \omega_3 V_{a,y}}{|V_a|}\right] \\
\frac{\partial \ddot{x}_{drag}}{\partial y} &= -500C_D\frac{A}{m}\rho\left[-V_{a,x}\frac{|V_a|y}{rH} + \omega_3|V_a| + V_{a,x}\frac{\omega_3 V_{a,x} - \omega_1 V_{a,z}}{|V_a|}\right] \\
\frac{\partial \ddot{x}_{drag}}{\partial z} &= -500C_D\frac{A}{m}\rho\left[-V_{a,x}\frac{|V_a|z}{rH} - \omega_2|V_a| + V_{a,x}\frac{\omega_1 V_{a,y} - \omega_2 V_{a,x}}{|V_a|}\right] \\
\frac{\partial \ddot{x}_{drag}}{\partial \dot{x}} &= -500C_D\frac{A}{m}\rho\left[\frac{V_{a,x}^2}{|V_a|} + |V_a|\right] \\
\frac{\partial \ddot{x}_{drag}}{\partial \dot{y}} &= -500C_D\frac{A}{m}\rho V_{a,x}\frac{V_{a,y}}{|V_a|} \\
\frac{\partial \ddot{x}_{drag}}{\partial \dot{z}} &= -500C_D\frac{A}{m}\rho V_{a,x}\frac{V_{a,z}}{|V_a|} \\
\frac{\partial \ddot{x}_{drag}}{\partial C_D} &= -500C_D\frac{A}{m}\rho V_{a,x}|V_a|
\end{aligned} \tag{105}$$

$$
\begin{aligned}
\frac{\partial \ddot{y}_{drag}}{\partial x} &= -500 C_D \frac{A}{m} \rho \left[-V_{a,y} \frac{|V_a| x}{rH} - \omega_3 |V_a| + V_{a,y} \frac{\omega_2 V_{a,z} - \omega_3 V_{a,y}}{|V_a|} \right] \\
\frac{\partial \ddot{y}_{drag}}{\partial y} &= -500 C_D \frac{A}{m} \rho V_{a,y} \left[-\frac{|V_a| y}{rH} + \frac{\omega_3 V_{a,x} - \omega_1 V_{a,z}}{|V_a|} \right] \\
\frac{\partial \ddot{y}_{drag}}{\partial z} &= -500 C_D \frac{A}{m} \rho \left[-V_{a,y} \frac{|V_a| z}{rH} + \omega_1 |V_a| + V_{a,y} \frac{\omega_1 V_{a,y} - \omega_2 V_{a,x}}{|V_a|} \right] \\
\frac{\partial \ddot{y}_{drag}}{\partial \dot{x}} &= -500 C_D \frac{A}{m} \rho V_{a,y} \frac{V_{a,x}}{|V_a|} \\
\frac{\partial \ddot{y}_{drag}}{\partial \dot{y}} &= -500 C_D \frac{A}{m} \rho \left[\frac{V_{a,y}^2}{|V_a|} + |V_a| \right] \\
\frac{\partial \ddot{y}_{drag}}{\partial \dot{z}} &= -500 C_D \frac{A}{m} \rho V_{a,y} \frac{V_{a,z}}{|V_a|} \\
\frac{\partial \ddot{y}_{drag}}{\partial C_D} &= -500 C_D \frac{A}{m} \rho V_{a,y} |V_a|
\end{aligned}
\tag{106}
$$

$$
\begin{aligned}
\frac{\partial \ddot{z}_{drag}}{\partial x} &= -500 C_D \frac{A}{m} \rho \left[-V_{a,z} \frac{|V_a| x}{rH} + \omega_2 |V_a| + V_{a,z} \frac{\omega_2 V_{a,z} - \omega_3 V_{a,y}}{|V_a|} \right] \\
\frac{\partial \ddot{z}_{drag}}{\partial y} &= -500 C_D \frac{A}{m} \rho \left[-V_{a,z} \frac{|V_a| y}{rH} - \omega_1 |V_a| + V_{a,z} \frac{\omega_3 V_{a,x} - \omega_1 V_{a,z}}{|V_a|} \right] \\
\frac{\partial \ddot{z}_{drag}}{\partial z} &= -500 C_D \frac{A}{m} \rho V_{a,z} \left[-\frac{|V_a| z}{rH} + \frac{\omega_1 V_{a,y} - \omega_2 V_{a,x}}{|V_a|} \right] \\
\frac{\partial \ddot{z}_{drag}}{\partial \dot{x}} &= -500 C_D \frac{A}{m} \rho V_{a,z} \frac{V_{a,x}}{|V_a|} \\
\frac{\partial \ddot{z}_{drag}}{\partial \dot{y}} &= -500 C_D \frac{A}{m} \rho V_{a,z} \frac{V_{a,y}}{|V_a|} \\
\frac{\partial \ddot{z}_{drag}}{\partial \dot{z}} &= -500 C_D \frac{A}{m} \rho \left[\frac{V_{a,z}^2}{|V_a|} + |V_a| \right] \\
\frac{\partial \ddot{z}_{drag}}{\partial C_D} &= -500 \frac{A}{m} \rho V_{a,z} |V_a|.
\end{aligned}
\tag{107}
$$

See Section 5.1.8 for a description of the notation.

SRP STM When Solar Radiation Pressure is modeled, TurboProp treats it as simply an adjustment to the Sun's gravitational parameter, so the partials of the acceleration due to SRP with respect to the position of the spacecraft are lumped into the partials due to the Sun's third-body gravity. The partials with respect to the reflectance, c_R, are explicity computed.

$$
\frac{\partial \ddot{\mathbf{r}}_{SRP}}{\partial c_R} = p_{SR} \frac{A}{m} \frac{\mathbf{r}_{\odot sat}}{|\mathbf{r}_{\odot sat}|} \tag{108}
$$

See Section 5.1.7 for a description of the notation.

5.1.10 Two-dimensional states "_2D"

The model has only two dimensions of position and velocity.

5.1.11 Unscented Kalman Filter Integrators "_ukf"

In the Unscented Kalman Filter (Julier and Uhlmann (1997)), the unscented transformation generates $2L$ additional sigma points that must be propagated forward. L is the number of filter estimated states. To facilitate this process, a single derivative function can be used to propagate all of the points in a single call to the integrator, as opposed to $2L$+1 calls. These derivative functions are the same as their counterparts without the `_ukf` tag, but they loop over multiple states in the `y0` array and compute derivatives for each. Thus, one call to the integration function will integrate all $2L$+1 states.

5.2 ODE Function Descriptions

5.2.1 TwoBody

This ODE function is a model of the two-body problem, where a spacecraft orbits a single, massive central body. The state, `y0`, is a 6×1 vector:

`y0(1)` x position
`y0(2)` y position
`y0(3)` z position
`y0(4)` $\dot{x}$ velocity
`y0(5)` $\dot{y}$ velocity
`y0(6)` $\dot{z}$ velocity

The origin of the coordinate frame is at the center of mass of the primary body. `extras.mu` must be included and should contain μ, the gravitational parameter of the primary body. It is up to the user to make sure the units of `y0`, `tspan`, and the gravitational parameter are consistent.

5.2.2 TwoBody_stm

This ODE function is a model of the two-body problem, where a spacecraft orbits a single, massive central body. The state, `y0`, is a 42×1 vector. The first 6 values are:

`y0(1)` x position
`y0(2)` y position
`y0(3)` z position
`y0(4)` $\dot{x}$ velocity
`y0(5)` $\dot{y}$ velocity
`y0(6)` $\dot{z}$ velocity

`y0(7:42)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the coordinate frame is at the center of mass of the primary body. `extras.mu` must be included and should contain μ, the gravitational parameter of the primary body. It is up to the user to make sure the units of `y0`, `tspan`, and the gravitational parameter are consistent.

5.2.3 TwoBody_grav

`TwoBody_grav` is a model of the two-body problem, where a spacecraft orbits a single, massive central body. The central body also has a spherical harmonic gravity field. The state, `y0`, is a 6×1 vector:

`y0(1)`	x position
`y0(2)`	y position
`y0(3)`	z position
`y0(4)`	$\dot{x}$ velocity
`y0(5)`	$\dot{y}$ velocity
`y0(6)`	$\dot{z}$ velocity

The origin of the coordinate frame is at the center of mass of the primary body. The `extras` structure should have all of the following fields:

`extras.mu`	(Section 4.15.1)
`extras.radius`	(Section 4.15.3)
`extras.degord`	(Section 4.15.6)
`extras.gravfile`	(Section 4.15.8)
`extras.EOP` (2-vector)	(Section 4.15.11)

`extras.mu` should contain μ, the gravitational parameter of the central body (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the central body (see Section 4.15.3). All units of length and time must be consistent.

5.2.4 TwoBody_grav_stm

`TwoBody_grav_stm` is a model of the two-body problem, where a spacecraft orbits a single, massive central body. The central body also has a spherical harmonic gravity field. The state, `y0`, includes the state transition matrix and is a 42×1 vector. The first 6 values are:

`y0(1)`	x position
`y0(2)`	y position
`y0(3)`	z position
`y0(4)`	$\dot{x}$ velocity
`y0(5)`	$\dot{y}$ velocity
`y0(6)`	$\dot{z}$ velocity

`y0(7:42)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the coordinate frame is at the center of mass of the primary body. The `extras` structure should have all of the following fields:

`extras.mu`	(Section 4.15.1)
`extras.radius`	(Section 4.15.3)
`extras.degord`	(Section 4.15.6)
`extras.degordstm`	(Section 4.15.7)
`extras.gravfile`	(Section 4.15.8)
`extras.EOP` (2-vector)	(Section 4.15.11)

`extras.mu` should contain μ, the gravitational parameter of the central body (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the central body (see Section 4.15.3). All units of length and time must be consistent.

5.2.5 `TwoBody_drag_grav`

`TwoBody_drag_grav` is a model of the two-body problem, where a spacecraft orbits a single, massive central body. The central body also has a spherical harmonic gravity field and an exponential atmosphere model for drag. The state, `y0`, is a 7×1 vector:

`y0(1)` x position
`y0(2)` y position
`y0(3)` z position
`y0(4)` $\dot{x}$ velocity
`y0(5)` $\dot{y}$ velocity
`y0(6)` $\dot{z}$ velocity
`y0(7)` C_D coefficient of drag

The origin of the coordinate frame is at the center of mass of the primary body. The `extras` structure should have all of the following fields:

`extras.mu`	(Section 4.15.1)
`extras.radius`	(Section 4.15.3)
`extras.degord`	(Section 4.15.6)
`extras.gravfile`	(Section 4.15.8)
`extras.A_m`	(Section 4.15.9)
`extras.EOP` (2-vector)	(Section 4.15.11)
`extras.atmos`	(Section 4.15.12)

`extras.mu` should contain μ, the gravitational parameter of the central body (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the central body (see Section 4.15.3). All units of length and time must be consistent.

5.2.6 `TwoBody_drag_grav_stm`

`TwoBody_drag_grav_stm` is a model of the two-body problem, where a spacecraft orbits a single, massive central body. The central body also has a spherical harmonic gravity field and an exponential atmosphere model for drag. The state, `y0`, includes a state transition matrix and is a 56×1 vector. The first 7 values are:

`y0(1)` x position
`y0(2)` y position
`y0(3)` z position
`y0(4)` $\dot{x}$ velocity
`y0(5)` $\dot{y}$ velocity
`y0(6)` $\dot{z}$ velocity
`y0(7)` C_D coefficient of drag

`y0(8:56)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the coordinate frame is at the center of mass of the primary body. The `extras` structure should have all of the following fields:

`extras.mu`	(Section 4.15.1)
`extras.radius`	(Section 4.15.3)
`extras.degord`	(Section 4.15.6)
`extras.degordstm`	(Section 4.15.7)
`extras.gravfile`	(Section 4.15.8)
`extras.A_m`	(Section 4.15.9)
`extras.EOP` (2-vector)	(Section 4.15.11)
`extras.atmos`	(Section 4.15.12)

`extras.mu` should contain μ, the gravitational parameter of the central body (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the central body (see Section 4.15.3). All units of length and time must be consistent.

5.2.7 TwoBody_grav_ukf

`TwoBody_grav_ukf` is a model of the two-body problem, where a spacecraft orbits a single, massive central body. However, it is designed for use in an Unscented Kalman Filter (UKF) (Julier and Uhlmann (1997)). In the UKF, the unscented transformation generates $2L$ additional sigma points that must be propagated forward. L is the number of filter estimated states. To facilitate this process, a single derivative function is used to propagate all of the points in a single call to the integrator, as opposed to $2L + 1$ calls. The central body also has a spherical harmonic gravity field. The state, `y0`, is a 78×1 vector:

`y0(1+(i-1)*6)`	x position of the i-th state
`y0(2+(i-1)*6)`	y position of the i-th state
`y0(3+(i-1)*6)`	z position of the i-th state
`y0(4+(i-1)*6)`	$\dot{x}$ velocity of the i-th state
`y0(5+(i-1)*6)`	$\dot{y}$ velocity of the i-th state
`y0(6+(i-1)*6)`	$\dot{z}$ velocity of the i-th state

In this case, $i = 1, \ldots, 13$.

The origin of the coordinate frame is at the center of mass of the primary body. The `extras` structure should have all of the following fields:

`extras.mu`	(Section 4.15.1)
`extras.radius`	(Section 4.15.3)
`extras.degord`	(Section 4.15.6)
`extras.gravfile`	(Section 4.15.8)
`extras.EOP` (2-vector)	(Section 4.15.11)

`extras.mu` should contain μ, the gravitational parameter of the central body (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the central body (see Section 4.15.3). All units of length and time must be consistent.

5.2.8 TwoBody_drag_grav_ukf

`TwoBody_drag_grav_ukf` is a model of the two-body problem, where a spacecraft orbits a single, massive central body. The central body also has a spherical harmonic gravity field and an exponential

atmosphere model for drag. This function is intended for use in a UKF. The state, `y0`, is a 105×1 vector:

`y0(1+(i-1)*7))`	x position of the i-th state
`y0(2+(i-1)*7))`	y position of the i-th state
`y0(3+(i-1)*7))`	z position of the i-th state
`y0(4+(i-1)*7))`	$\dot{x}$ velocity of the i-th state
`y0(5+(i-1)*7))`	$\dot{y}$ velocity of the i-th state
`y0(6+(i-1)*7))`	$\dot{z}$ velocity of the i-th state
`y0(7+(i-1)*7))`	C_D coefficient of drag of the i-th state

In this case, $i = 1, \ldots, 15$

The origin of the coordinate frame is at the center of mass of the primary body. The `extras` structure should have all of the following fields:

`extras.mu`	(Section 4.15.1)
`extras.radius`	(Section 4.15.3)
`extras.degord`	(Section 4.15.6)
`extras.gravfile`	(Section 4.15.8)
`extras.A_m`	(Section 4.15.9)
`extras.EOP` (2-vector)	(Section 4.15.11)
`extras.atmos`	(Section 4.15.12)

`extras.mu` should contain μ, the gravitational parameter of the central body (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the central body (see Section 4.15.3). All units of length and time must be consistent.

5.2.9 CRTBP

`CRTBP` is a model of the circular-restricted three-body problem in a rotating frame described in Section 5.1.2. The state, `y0`, is a 6×1 vector:

`y0(1)`	x position (LU)
`y0(2)`	y position (LU)
`y0(3)`	z position (LU)
`y0(4)`	$\dot{x}$ velocity (LU/TU)
`y0(5)`	$\dot{y}$ velocity (LU/TU)
`y0(6)`	$\dot{z}$ velocity (LU/TU)

The origin of the coordinate frame is at the three-body barycenter. `extras.mu` must be included and should contain μ, the three-body system gravitational parameter. The units of length, time, and mass must be the nondimensional units described in Section 5.1.2.

5.2.10 CRTBP_stm

`CRTBP_stm` is a model of the circular-restricted three-body problem in a rotating frame described in Section 5.1.2. The state, `y0`, is a 42×1 vector. The first 6 values are:

`y0(1)` x position (LU)
`y0(2)` y position (LU)
`y0(3)` z position (LU)
`y0(4)` $\dot{x}$ velocity (LU/TU)
`y0(5)` $\dot{y}$ velocity (LU/TU)
`y0(6)` $\dot{z}$ velocity (LU/TU)

`y0(7:42)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the coordinate frame is at the three-body barycenter.

`extras.mu` must be included and should contain μ, the three-body system gravitational parameter. The units of length, time, and mass must be the nondimensional units described in Section 5.1.2.

5.2.11 `CRTBP_2D`

`CRTBP_2D` is a two-dimensional model of the circular-restricted three-body problem (or the planar CRTBP) in a rotating frame described in Section 5.1.2. The state, `y0`, is a 4×1 vector:

`y0(1)` x position (LU)
`y0(2)` y position (LU)
`y0(3)` $\dot{x}$ velocity (LU/TU)
`y0(4)` $\dot{y}$ velocity (LU/TU)

The origin of the coordinate frame is at the three-body barycenter.

`extras.mu` must be included and should contain μ, the three-body system gravitational parameter. The units of length, time, and mass must be the nondimensional units described in Section 5.1.2.

5.2.12 `CRTBP_stm_2D`

`CRTBP_stm_2D` is a two-dimensional model of the circular-restricted three-body problem (or the planar CRTBP) in a rotating frame described in Section 5.1.2. The state, `y0`, is a 20×1 vector. The first 4 values are:

`y0(1)` x position (LU)
`y0(2)` y position (LU)
`y0(3)` $\dot{x}$ velocity (LU/TU)
`y0(4)` $\dot{y}$ velocity (LU/TU)

`y0(5:20)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the coordinate frame is at the three-body barycenter.

`extras.mu` must be included and should contain μ, the three-body system gravitational parameter. The units of length, time, and mass must be the nondimensional units described in Section 5.1.2.

5.2.13 `CRTBP_grav`

`CRTBP_grav` is a model of the circular-restricted three-body problem in a rotating frame described in Section 5.1.2. However, `CRTBP_grav` also includes a perturbation to the three-body problem due to a spherical harmonic gravity model for the smaller primary, or the secondary body. Currently, the software includes the assumption that the secondary body is in phase lock with the primary body, i.e. the rotation period is the same as the orbit period. Thus, this function is only applicable to the Earth-Moon system where the lunar gravity field is modeled. The state, `y0`, is a 6×1 vector:

`y0(1)` $\tilde{x}$ position (LU)
`y0(2)` $\tilde{y}$ position (LU)
`y0(3)` $\tilde{z}$ position (LU)
`y0(4)` $\dot{\tilde{x}}$ velocity (LU/TU)
`y0(5)` $\dot{\tilde{y}}$ velocity (LU/TU)
`y0(6)` $\dot{\tilde{z}}$ velocity (LU/TU)

The origin of the coordinate frame is not at the three-body barycenter like the other `CRTBP` models, but the origin is at the center of the secondary, with the x-axis pointed toward the primary and the z-axis perpendicular to the orbit plane of the primary and secondary. Note that this reference frame is rotated 180° about the z-axis from the other `CRTBP` models. To convert from the standard `CRTBP` coordinates to the `CRTBP_grav` coordinates, use code like this:

```
y0(1) = y0(1) - (1-mu);
y0(1:2) = -y0(1:2);
y0(4:5) = -y0(4:5);
```

This code converts a state vector `y0` from the normal `CRTBP` barycentric coordinate system to a `CRTBP_grav` coordinate system centered on the secondary with the x-axis positive in the direction of the primary body.

The `extras` structure should have all of the following fields:

`extras.mu` (Section 4.15.1)
`extras.radius` (LU, Section 4.15.3)
`extras.degord` (Section 4.15.6)
`extras.gravfile` (Section 4.15.8)

`extras.mu` should contain μ, the three-body system gravitational parameter (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the secondary body in length units (LU) (see Section 4.15.3).

All units of length, time, and mass must be the nondimensional units described in Section 5.1.2.

5.2.14 `CRTBP_grav_stm`

`CRTBP_grav_stm` is a model of the circular-restricted three-body problem in a rotating frame described in Section 5.1.2. However, `CRTBP_grav_stm` also includes a perturbation to the three-body problem due to a spherical harmonic gravity model for the smaller primary, or the secondary body. Currently, the software includes the assumption that the secondary body is in phase lock with the primary body, i.e. the rotation period is the same as the orbit period. Thus, this function is only applicable to the Earth-Moon system where the lunar gravity field is modeled. The state, `y0`, is a 42×1 vector. The first 6 values are:

`y0(1)` $\tilde{x}$ position (LU)
`y0(2)` $\tilde{y}$ position (LU)
`y0(3)` $\tilde{z}$ position (LU)
`y0(4)` $\dot{\tilde{x}}$ velocity (LU/TU)
`y0(5)` $\dot{\tilde{y}}$ velocity (LU/TU)
`y0(6)` $\dot{\tilde{z}}$ velocity (LU/TU)

`y0(7:42)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the coordinate frame is not at the three-body barycenter like the other `CRTBP` models, but

the origin is at the center of the secondary, with the x-axis pointed toward the primary and the z-axis perpendicular to the orbit plane of the primary and secondary. Note that this reference frame is rotated 180° about the z-axis from the other `CRTBP` models. To convert from the standard `CRTBP` coordinates to the `CRTBP_grav_stm` coordinates, use code like this:

```
y0(1) = y0(1) - (1-mu);
y0(1:2) = -y0(1:2);
y0(4:5) = -y0(4:5);
```

This code converts a state vector `y0` from the normal `CRTBP` barycentric coordinate system to a `CRTBP_grav_stm` coordinate system centered on the secondary with the x-axis positive in the direction of the primary body.

The `extras` structure should have all of the following fields:

`extras.mu`	(Section 4.15.1)
`extras.radius`	(LU, Section 4.15.3)
`extras.degord`	(Section 4.15.6)
`extras.degordstm`	(Section 4.15.7)
`extras.gravfile`	(Section 4.15.8)

`extras.mu` should contain μ, the three-body system gravitational parameter (see Section 4.15.1). `extras.radius` should be included also and should contain the radius of the secondary body in length units (LU) (see Section 4.15.3).

All units of length, time, and mass must be the nondimensional units described in Section 5.1.2.

5.2.15 DE

`DE` is a model of the solar system using the JPL DE ephemerides. The state, `y0`, is a 6×1 vector:

`y0(1)`	x position (km)
`y0(2)`	y position (km)
`y0(3)`	z position (km)
`y0(4)`	$\dot{x}$ velocity (km/s)
`y0(5)`	$\dot{y}$ velocity (km/s)
`y0(6)`	$\dot{z}$ velocity (km/s)

The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). `tspan` must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.reftime`	(JED, Section 4.15.10)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)

5.2.16 DE_stm

`DE_stm` is a model of the solar system using the JPL DE ephemerides with a state transition matrix that includes the contributions of the planetary point masses. The state, `y0`, is a 42×1 vector. The first

6 values are:

`y0(1)`	x position (km)
`y0(2)`	y position (km)
`y0(3)`	z position (km)
`y0(4)`	$\dot{x}$ velocity (km/s)
`y0(5)`	$\dot{y}$ velocity (km/s)
`y0(6)`	$\dot{z}$ velocity (km/s)

`y0(7:42)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). `tspan` must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.reftime`	(JED, Section 4.15.10)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)

5.2.17 DE_SRP

`DE_SRP` is a model of the solar system using the JPL DE ephemerides and solar radiation pressure. The state, `y0`, is a 7×1 vector:

`y0(1)`	x position (km)
`y0(2)`	y position (km)
`y0(3)`	z position (km)
`y0(4)`	$\dot{x}$ velocity (km/s)
`y0(5)`	$\dot{y}$ velocity (km/s)
`y0(6)`	$\dot{z}$ velocity (km/s)
`y0(7)`	c_R reflectance

The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). `tspan` must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.A_m`	(m^2/kg, Section 4.15.9)
`extras.reftime`	(JED, Section 4.15.10)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)

5.2.18 DE_SRP_stm

`DE_SRP_stm` is a model of the solar system using the JPL DE ephemerides and solar radiation pressure. The state transition matrix is integrated as well and includes contributions due to solar radiation pressure and the planetary point masses. The state, `y0`, is a 49×1 vector. The first 7 parameters are:

`y0(1)` x position (km)
`y0(2)` y position (km)
`y0(3)` z position (km)
`y0(4)` $\dot{x}$ velocity (km/s)
`y0(5)` $\dot{y}$ velocity (km/s)
`y0(6)` $\dot{z}$ velocity (km/s)
`y0(7)` c_R reflectance

`y0(8:56)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). `tspan` must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.A_m`	(m^2/kg, Section 4.15.9)
`extras.reftime`	(JED, Section 4.15.10)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)

5.2.19 DE_grav

`DE_grav` is a model of the solar system using the JPL DE ephemerides and a spherical harmonic gravity model for one of the planets. The state, `y0`, is a 6×1 vector:

`y0(1)` x position (km)
`y0(2)` y position (km)
`y0(3)` z position (km)
`y0(4)` $\dot{x}$ velocity (km/s)
`y0(5)` $\dot{y}$ velocity (km/s)
`y0(6)` $\dot{z}$ velocity (km/s)

The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). `tspan` must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.degord`	(Section 4.15.6)
`extras.gravfile`	(Section 4.15.8)
`extras.reftime`	(JED, Section 4.15.10)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)
`extras.radius`	(km, optional, Section 4.15.3)
`extras.EOP` (15-vector)	(Optional when `extras.center=11`, Section 4.15.11)

5.2.20 DE_grav_stm

DE_grav_stm is a model of the solar system using the JPL DE ephemerides and a spherical harmonic gravity model for one of the planets. The state transition matrix is also integrated, and the contributions of a gravity model are included in the variational equations along with the planetary point masses. The state, y0, is a 42×1 vector. The first six parameters are:

y0(1)	x position (km)
y0(2)	y position (km)
y0(3)	z position (km)
y0(4)	$\dot{x}$ velocity (km/s)
y0(5)	$\dot{y}$ velocity (km/s)
y0(6)	$\dot{z}$ velocity (km/s)

y0(7:42) contain the state transition matrix in column order, as described in Section 4.11. The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). tspan must be in seconds also.

The extras structure should have all of the following fields:

extras.planets	(Section 4.15.2)
extras.center	(Section 4.15.4)
extras.ephemfile	(Section 4.15.5)
extras.degord	(Section 4.15.6)
extras.degordstm	(Section 4.15.7)
extras.gravfile	(Section 4.15.8)
extras.reftime	(JED, Section 4.15.10)
extras.mu	(km^3/s^2, optional, Section 4.15.1)
extras.radius	(km, optional, Section 4.15.3)
extras.EOP (15-vector)	(Optional when extras.center=11, Section 4.15.11)

5.2.21 DE_SRP_grav

DE_SRP_grav is a model of the solar system using the JPL DE ephemerides, solar radiation pressure, and a spherical harmonic gravity model for one of the planets. The state, y0, is a 7×1 vector:

y0(1)	x position (km)
y0(2)	y position (km)
y0(3)	z position (km)
y0(4)	$\dot{x}$ velocity (km/s)
y0(5)	$\dot{y}$ velocity (km/s)
y0(6)	$\dot{z}$ velocity (km/s)
y0(7)	c_R reflectance

The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). tspan must be in seconds also.

The extras structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.degord`	(Section 4.15.6)
`extras.gravfile`	(Section 4.15.8)
`extras.A_m`	(m^2/kg, Section 4.15.9)
`extras.reftime`	(JED, Section 4.15.10)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)
`extras.radius`	(km, optional, Section 4.15.3)
`extras.EOP` (15-vector)	(Optional when `extras.center=11`, Section 4.15.11)

5.2.22 DE_SRP_grav_stm

`DE_SRP_grav_stm` is a model of the solar system using the JPL DE ephemerides, solar radiation pressure, and a spherical harmonic gravity model for one of the planets. The state transition matrix is also integrated, and the contributions of solar radiation pressure and a gravity model are included in the variational equations along with the planetary point masses. The state, `y0`, is a 56×1 vector. The first seven parameters are:

`y0(1)`	x position (km)
`y0(2)`	y position (km)
`y0(3)`	z position (km)
`y0(4)`	$\dot{x}$ velocity (km/s)
`y0(5)`	$\dot{y}$ velocity (km/s)
`y0(6)`	$\dot{z}$ velocity (km/s)
`y0(7)`	c_R reflectance

`y0(8:56)` contain the state transition matrix in column order, as described in Section 4.11. The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). `tspan` must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.degord`	(Section 4.15.6)
`extras.degordstm`	(Section 4.15.7)
`extras.gravfile`	(Section 4.15.8)
`extras.A_m`	(m^2/kg, Section 4.15.9)
`extras.reftime`	(JED, Section 4.15.10)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)
`extras.radius`	(km, optional, Section 4.15.3)
`extras.EOP` (15-vector)	(Optional when `extras.center=11`, Section 4.15.11)

5.2.23 DE_drag_grav

DE_drag_grav is a model of the solar system using the JPL DE ephemerides, atmospheric drag, and a spherical harmonic gravity model for one of the planets. The state, y0, is a 7×1 vector:

- y0(1) x position (km)
- y0(2) y position (km)
- y0(3) z position (km)
- y0(4) $\dot{x}$ velocity (km/s)
- y0(5) $\dot{y}$ velocity (km/s)
- y0(6) $\dot{z}$ velocity (km/s)
- y0(7) C_D drag coefficient

The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). tspan must be in seconds also.

The extras structure should have all of the following fields:

extras.planets	(Section 4.15.2)
extras.center	(Section 4.15.4)
extras.ephemfile	(Section 4.15.5)
extras.degord	(Section 4.15.6)
extras.gravfile	(Section 4.15.8)
extras.A_m	(m^2/kg, Section 4.15.9)
extras.reftime	(JED, Section 4.15.10)
extras.atmos	([kg/m^3 km km], Section 4.15.12)
extras.mu	(km^3/s^2, optional, Section 4.15.1)
extras.radius	(km, optional, Section 4.15.3)
extras.EOP (15-vector)	(Optional when extras.center=11, Section 4.15.11)

5.2.24 DE_drag_grav_stm

DE_drag_grav_stm is a model of the solar system using the JPL DE ephemerides, atmospheric drag, and a spherical harmonic gravity model for one of the planets. The state transition matrix is also integrated, and the contributions of a gravity model are included in the variational equations along with the planetary point masses, and drag. The state, y0, is a 56×1 vector. The first seven parameters are:

- y0(1) x position (km)
- y0(2) y position (km)
- y0(3) z position (km)
- y0(4) $\dot{x}$ velocity (km/s)
- y0(5) $\dot{y}$ velocity (km/s)
- y0(6) $\dot{z}$ velocity (km/s)
- y0(7) C_D drag coefficient

y0(8:56) contain the state transition matrix in column order, as described in Section 4.11. The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). tspan must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.degord`	(Section 4.15.6)
`extras.degordstm`	(Section 4.15.7)
`extras.gravfile`	(Section 4.15.8)
`extras.A_m`	(m^2/kg, Section 4.15.9)
`extras.reftime`	(JED, Section 4.15.10)
`extras.atmos`	([kg/m^3 km km], Section 4.15.12)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)
`extras.radius`	(km, optional, Section 4.15.3)
`extras.EOP` (15-vector)	(Optional when `extras.center=11`, Section 4.15.11)

5.2.25 DE_drag_SRP_grav

`DE_drag_SRP_grav` is a model of the solar system using the JPL DE ephemerides, atmospheric drag, solar radiation pressure, and a spherical harmonic gravity model for one of the planets. The state, `y0`, is a 8×1 vector:

`y0(1)`	x position (km)
`y0(2)`	y position (km)
`y0(3)`	z position (km)
`y0(4)`	$\dot{x}$ velocity (km/s)
`y0(5)`	$\dot{y}$ velocity (km/s)
`y0(6)`	$\dot{z}$ velocity (km/s)
`y0(7)`	C_D drag coefficient
`y0(8)`	c_R reflectance

The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). `tspan` must be in seconds also.

The `extras` structure should have all of the following fields:

`extras.planets`	(Section 4.15.2)
`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.degord`	(Section 4.15.6)
`extras.gravfile`	(Section 4.15.8)
`extras.A_m`	(m^2/kg, Section 4.15.9)
`extras.reftime`	(JED, Section 4.15.10)
`extras.atmos`	([kg/m^3 km km], Section 4.15.12)
`extras.mu`	(km^3/s^2, optional, Section 4.15.1)
`extras.radius`	(km, optional, Section 4.15.3)
`extras.EOP` (15-vector)	(Optional when `extras.center=11`, Section 4.15.11)

5.2.26 DE_drag_SRP_grav_stm

DE_drag_SRP_grav_stm is a model of the solar system using the JPL DE ephemerides, atmospheric drag, solar radiation pressure, and a spherical harmonic gravity model for one of the planets. The state transition matrix is also integrated, and the contributions of a gravity model are included in the variational equations along with the planetary point masses, solar radiation pressure, and drag. The state, y0, is a 72×1 vector. The first eight parameters are:

y0(1)	x position (km)
y0(2)	y position (km)
y0(3)	z position (km)
y0(4)	$\dot{x}$ velocity (km/s)
y0(5)	$\dot{y}$ velocity (km/s)
y0(6)	$\dot{z}$ velocity (km/s)
y0(7)	C_D drag coefficient
y0(8)	c_R reflectance

y0(9:72) contain the state transition matrix in column order, as described in Section 4.11. The origin of the inertial coordinate frame is at the center of the specified celestial body, using the ICRF coordinate system. The unit of length must be kilometers and the unit of time is seconds, so velocities are expressed in kilometers per second (km/s). tspan must be in seconds also.

The extras structure should have all of the following fields:

extras.planets	(Section 4.15.2)
extras.center	(Section 4.15.4)
extras.ephemfile	(Section 4.15.5)
extras.degord	(Section 4.15.6)
extras.degordstm	(Section 4.15.7)
extras.gravfile	(Section 4.15.8)
extras.A_m	(m^2/kg, Section 4.15.9)
extras.reftime	(JED, Section 4.15.10)
extras.atmos	([kg/m^3 km km], Section 4.15.12)
extras.mu	(km^3/s^2, optional, Section 4.15.1)
extras.radius	(km, optional, Section 4.15.3)
extras.EOP (15-vector)	(Optional when extras.center=11, Section 4.15.11)

5.2.27 PointMasses

PointMasses is a model of the gravity field using the point mass model. The state, y0, is a 6×1 vector:

y0(1)	x position
y0(2)	y position
y0(3)	z position
y0(4)	$\dot{x}$ velocity
y0(5)	$\dot{y}$ velocity
y0(6)	$\dot{z}$ velocity

The origin of the coordinate frame is at the center of mass of the point masses. The extras structure should have all of the following fields:

`extras.gravfile` (Section 4.15.8)
`extras.EOP` (2-vector) (Section 4.15.11)
All units of length must be in km and time must be in seconds.

6 Auxiliary Functions

6.1 deriv_mex

`deriv_mex` is a function that can be used to obtain the output from a ODE function directly. The syntax of the function call is:

```
dy = deriv_mex(odefun,t,y0,extras)
```

The inputs `odefun`, `y0`, and `extras` are the same as would be used when calling one of the integrators. The difference is that only a single time is sent, `t`, for which the derivative of the state at `y0` is computed. `dy` is a column vector containing that derivative. In MATLAB, type `help deriv_mex` for information.

6.2 UTC2JED.m

The function `UTC2JED.m` can be used to convert from a UTC calendar date to a Julian Ephemeris Date (JED). JED is similar to Julian Date except it specifically is computed for JPL Coordinate Time, which is similar to Dynamic Barycentric Time (TDB). This function must be updated with the epochs of new leap seconds in Modified Julian Dates (UTC). In MATLAB, type `help UTC2JED` for information.

6.3 JD2JED.m

The function `JD2JED.m` can be used to convert from a Julian Date computed from UTC to a Julian Ephemeris Date (JED). This function must be updated with the epochs of new leap seconds in Modified Julian Dates (UTC). In MATLAB, type `help JD2JED` for information.

6.4 JED2UTC.m

The function `JED2UTC.m` can be used to convert from a Julian Ephemeris Date (JED) to a UTC calendar date. This function must be updated with the epochs of new leap seconds in Modified Julian Dates (UTC). In MATLAB, type `help JED2UTC` for information.

6.5 JED2JD.m

The function `JED2JD.m` can be used to convert from a Julian Ephemeris Date (JED) to a Julian Date based on UTC. This function must be updated with the epochs of new leap seconds in Modified Julian Dates (UTC). In MATLAB, type `help JED2JD` for information.

6.6 UTC2TT.m

The function `UTC2TT.m` can be used to convert from a UTC calendar date to a Julian Date in Terrestrial Time (JD_{TT}). This function must be updated with the epochs of new leap seconds in Modified Julian Dates (UTC). In MATLAB, type `help UTC2TT` for information.

6.7 TT2JD.m

The function `TT2JD.m` can be used to convert from a Julian Date based on Terrestrial Time (JD_{TT}) to a Julian Date based on UTC. This function must be updated with the epochs of new leap seconds in Modified Julian Dates (UTC). In MATLAB, type `help TT2JD` for information.

6.8 JED2TT.m

The function `JED2TT.m` can be used to convert from a Julian Ephemeris Date (JED) to a Julian Date based on Terrestrial Time (JD_{TT}). This function must be updated with the epochs of new leap seconds in Modified Julian Dates (UTC). In MATLAB, type `help JED2TT` for information.

6.9 JPLDE

`JPLDE` is a function that can be used to query the JPL binary planetary ephemeris file for the positions and velocities of a planet. Remember to input JED, not the Julian Date based on UTC. In MATLAB, type `help JPLDE` for information.

6.10 CRTBP_DE.m

`CRTBP_DE.m` is used to convert an orbit generated in the CRTBP into the JPL ephemeris model centered on the Solar-System barycenter, or vice versa. `ThreeBodySystem.m` was changed to output `LU` and `TU`, which can then be input directly into `CRTBP_DE.m` and `rot_inert.m`. Now `ThreeBodySystem.m` also offers automatic unit conversions. In MATLAB, type `help CRTBP_DE` for information.

6.11 rot_inert.m

`rot_inert.m` is used to convert an orbit in the CRTBP into an inertial reference frame (non-rotating) for the three-body problem, or vice versa. The inputs to the function `rot_inert.m` were changed to be compatible with the new `ThreeBodySystem.m`. In MATLAB, type `help rot_inert` for information.

6.12 ThreeBodySystem.m

`ThreeBodySystem.m` is a function that can be used to obtain two- and three-body gravitational parameters, the nondimensional length and time units, and radii of the primaries for several three-body systems. `ThreeBodySystem.m` was changed to output `LU` and `TU`, which can then be input

directly into `CRTBP_DE.m` and `rot_inert.m`. In MATLAB, type `help ThreeBodySystem` for information.

6.13 EOPfind.m

The function `EOPfind.m` can be used to search an IERS data file to find the proper Earth Orientation Parameters to fill the `extras.EOP` vector. In MATLAB, type `help EOPfind` for information.

6.14 EarthFrame.m

The function `EarthFrame.m` can be used, with optional data from a `extras.EOP` vector, to convert an ephemeris between the coordinate systems GCRF, J2000, MOD, TOD, and ITRF. In MATLAB, type `help EarthFrame` for information or see Section 5.1.6.

6.15 MoonFrame.m

The function `MoonFrame.m` can be used to convert an ephemeris between the Moon-centered inertial (ICRF) and the Moon-centered, Moon-fixed coordinate systems. In MATLAB, type `help MoonFrame` for information or see Section 5.1.6.

6.16 SurferIC.m

The function `SurferIC.m` can be used to generate initial conditions for `Surfer.m` using inputs of latitude, longitude, height, heading, and speed. In MATLAB, type `help SurferIC` for information.

For the Earth, the Earth-fixed position, $\mathbf{r}_{ITRF}$, is computed from geodetic latitude, ϕ_{gd}, longitude, λ, and ellipsoidal height, h_{ellp}, using Eqn. 3-14 in Vallado and McClain (2001) (error in 2nd ed.).

$$\mathbf{r}_{ITRF} = \begin{bmatrix} (C_{\oplus} + h_{ellp}) \cos(\phi_{gd}) \cos(\lambda) \\ (C_{\oplus} + h_{ellp}) \cos(\phi_{gd}) \sin(\lambda) \\ (S_{\oplus} + h_{ellp}) \sin(\phi_{gd}) \end{bmatrix} \tag{109}$$

$C_{\oplus}$ and $S_{\oplus}$ are given from Eqn. 3-7 in Vallado and McClain (2001).

$$\begin{aligned} C_{\oplus} &= \frac{R_{\oplus}}{\sqrt{1 - e_{\oplus}^2 \sin^2 \phi_{gd}}} \\ S_{\oplus} &= \frac{R_{\oplus}(1 - e_{\oplus}^2)}{\sqrt{1 - e_{\oplus}^2 \sin^2 \phi_{gd}}} \end{aligned} \tag{110}$$

$R_{\oplus}$ is the equatorial radius of the Earth and is set to 6378.1363 km. $e_{\oplus}$ is the eccentricity of the Earth and Vallado and McClain (2001) gives it a value of 0.081819221456 on in section 3.2 of the 2nd ed. For the Moon, finding the Moon-fixed position is much easier, since $e_{\leftmoon}$ is zero and $C_{\leftmoon} = S_{\leftmoon} = R_{\leftmoon}$. $R_{\leftmoon}$ is set to 1737.4 km to be consistent with lunar topographical data, although this is different than the radius used in gravity field computations.

For vehicles, the heading and speed are converted to $\mathbf{v}_{SEZ}$, a velocity vector in SEZ coordinates (South-East-Zenith).

$$\mathbf{v}_{SEZ} = |v| \begin{bmatrix} -\cos(\alpha) \\ \sin(\alpha) \\ 0 \end{bmatrix} \tag{111}$$

$|v|$ is the speed of the vehicle (converted from km/hr to km/s). α is the heading of the vehicle converted from degrees to radians and is zero for due north.

$\mathbf{v}_{SEZ}$ is then converted to an planet-fixed velocity, $\mathbf{v}_{fixed}$ using Eqn. 3-26 in Vallado and McClain (2001).

$$\mathbf{v}_{fixed} = \begin{bmatrix} \sin(\phi_{gd})\cos(\lambda) & -\sin(\lambda) & \cos(\phi_{gd})\cos(\lambda) \\ \sin(\phi_{gd})\sin(\lambda) & \cos(\lambda) & \cos(\phi_{gd})\sin(\lambda) \\ -\cos(\phi_{gd}) & 0 & \sin(\phi_{gd}) \end{bmatrix} \mathbf{v}_{SEZ} \tag{112}$$

$\mathbf{r}_{fixed}$ and $\mathbf{v}_{fixed}$ are then converted to the inertial frame using `EarthFrame.m` or `MoonFrame.m`, as appropriate.

6.17 Surfer.m

`Surfer.m` is a MATLAB function set up just like the ODE solvers, except that it creates an ephemeris for a ground station or a ground vehicle instead of an orbiting satellite. The ephemeris is created in the inertial frame, just like satellite ephemerides, so the ground stations can be used in orbit determination software without any special acommodation. Here is its function call:

```
[T,Y] = Surfer(odefun,tspan,y0,options,extras)
```

The outputs `T` and `Y` are just like the outputs of the ODE solvers. `odefun` is a string with the following ephemeris types:

```
Earth  Earth_stm  Moon  Moon_stm
```

The terms `Earth` or `Moon` specify the planet upon which the station or vehicle is located. The term `_stm` specifies that a state transition matrix (STM) is desired in the output matrix `Y`.

`tspan` is a vector of times, in seconds, for which the inertial state of the surface object is desired. `y0` is the inertial state vector of the ground object at the time indicated in `tspan/86400 + extras.reftime`. The units must be km and km/s. The MATLAB function `SurferIC.m` can be used to convert latitude, longitude, height, heading and speed information into an inertial vector to input as the `y0` vector.

`options` is a scalar that is 0 when the surface object is a ground station and 1 when it is a vehicle.

`extras` is a structure used to pass extra parameters to the `Surfer` function. For `Surfer.m`, the following fields are required:

`extras.center`	(Section 4.15.4)
`extras.ephemfile`	(Section 4.15.5)
`extras.reftime`	(JED, Section 4.15.10)
`extras.EOP` (15-vector)	(Optional when `extras.center=11`, Section 4.15.11)

As an example on how to used `extras.center`, if a station were on the surface of the Moon, but it's inertial coordinates and STM were desired in an Earth-centered frame (GCRF), set `extras.center=11` and `odefun='Moon_stm'`.

Surfer.m works in the following way: For the first value in tspan, t_0, the Earth orientation or lunar orientation conversions $\gamma(t_0)$ and $\dot{\gamma}(t_0)$ are computed using the techniques shown in Section 5.1.6. The inertial state y0 will be called $\mathbf{x}_{inertial}$, and it is converted to a planet-fixed state, $\mathbf{x}_{fixed}$ for a moving ground vehicle like this:

$$\mathbf{x}_{fixed}(t_0) = \begin{bmatrix} \gamma(t_0) & [0]_{3\times3} \\ \dot{\gamma}(t_0) & \gamma(t_0) \end{bmatrix} \mathbf{x}_{inertial}(t_0). \tag{113}$$

$[0]_{3\times3}$ is a 3×3 null matrix. For each successive time, t, in tspan, the new fixed state is computed by assuming that the planet-fixed velocity remains constant.

$$\mathbf{x}_{fixed}(t) = \begin{bmatrix} [I]_{3\times3} & (t-t_0)[I]_{3\times3} \\ [0]_{3\times3} & [I]_{3\times3} \end{bmatrix} \mathbf{x}_{fixed}(t_0) \tag{114}$$

$[I]_{3\times3}$ is a 3×3 identity matrix. Then the inertial state at time t is

$$\mathbf{x}_{inertial}(t) = \begin{bmatrix} [\gamma(t)]^T & [0]_{3\times3} \\ [\dot{\gamma}(t)]^T & [\gamma(t)]^T \end{bmatrix} \mathbf{x}_{fixed}(t). \tag{115}$$

Putting it all together gives

$$\mathbf{x}_{inertial}(t) = \begin{bmatrix} [\gamma(t)]^T & [0]_{3\times3} \\ [\dot{\gamma}(t)]^T & [\gamma(t)]^T \end{bmatrix} \begin{bmatrix} [I]_{3\times3} & (t-t_0)[I]_{3\times3} \\ [0]_{3\times3} & [I]_{3\times3} \end{bmatrix} \begin{bmatrix} \gamma(t_0) & [0]_{3\times3} \\ \dot{\gamma}(t_0) & \gamma(t_0) \end{bmatrix} \mathbf{x}_{inertial}(t_0), \tag{116}$$

which can be abbreviated by combining the three matrices

$$\mathbf{x}_{inertial}(t) = \phi(t, t_0)\mathbf{x}_{inertial}(t_0). \tag{117}$$

$\phi(t, t_0)$ is the state transition matrix.

$$\phi(t, t_0) = \begin{bmatrix} [\gamma(t)]^T & [0]_{3\times3} \\ [\dot{\gamma}(t)]^T & [\gamma(t)]^T \end{bmatrix} \begin{bmatrix} [I]_{3\times3} & (t-t_0)[I]_{3\times3} \\ [0]_{3\times3} & [I]_{3\times3} \end{bmatrix} \begin{bmatrix} \gamma(t_0) & [0]_{3\times3} \\ \dot{\gamma}(t_0) & \gamma(t_0) \end{bmatrix} \tag{118}$$

For a motionless station, the equations are altered so that the planet-fixed velocity is always zero. The state transition matrix would be

$$\phi(t, t_0) = \begin{bmatrix} [\gamma(t)]^T & [0]_{3\times3} \\ [\dot{\gamma}(t)]^T & [0]_{3\times3} \end{bmatrix} \begin{bmatrix} \gamma(t_0) & [0]_{3\times3} \\ [0]_{3\times3} & [0]_{3\times3} \end{bmatrix}, \tag{119}$$

and is used to compute the inertial state vector as shown above.

6.18 TimeFrame.m

The TimeFrame.m function is used to convert between different time scales, such as UTC to Julian Date. In MATLAB, type help TimeFrame for information.

6.19 randv.m

The MATLAB function `randv.m` will convert Keplerian Orbit Elements to Cartesian coordinates. For more information, type `help randv` in MATLAB.

6.20 elorb.m

The `elorb.m` function will convert Cartesian coordinates to Keplerian Orbit elements. For information, type `help elorb` in MATLAB.

7 MATLAB Graphical User Interface

In order to make learning TurboProp in MATLAB easier, a GUI has been developed to automatically generate code. This code is written to the MATLAB command window, and can be copied into a separate file for later execution.

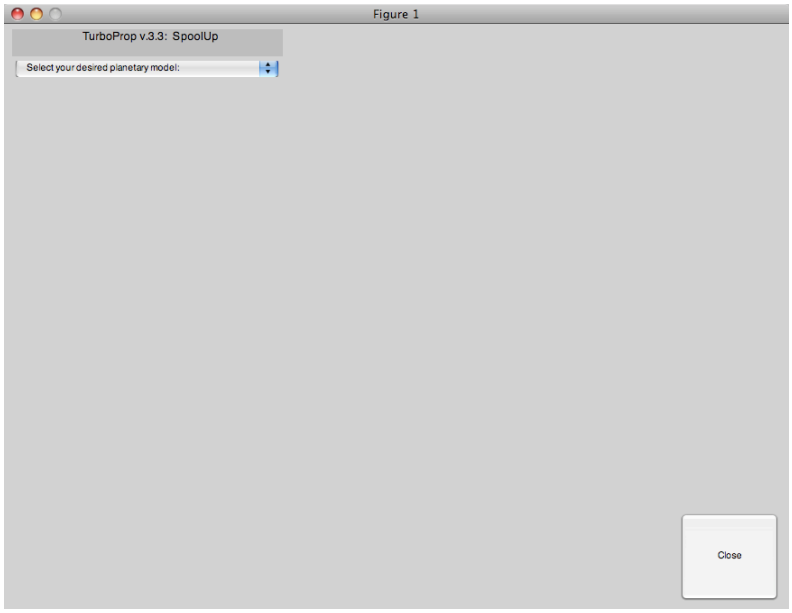

Figure 3: Initial MATLAB GUI

To execute the GUI, run the command `SpoolUp` from the MATLAB command line. The GUI should open, and is depicted in Figure 3. The GUI is comprised of several elements that only activated when other elements allow their usage. For example, the "Gravity Field" pane will only be enabled when the primary body is the Earth or the Moon (see Figure 4). To begin, select a desired planetary model from the drop menu in the top left corner. After the options have been selected for the propagated orbit, click the "Write Code" button. The generated code will be written to the MATLAB command window for future use.

The MATLAB GUI does not support all of the capabilities of TurboProp. For example, the `rk4sym()` integrator is not listed as an option. Code generated from `SpoolUp` can be adjusted accordingly to support all of the TurboProp features. Future versions will support these additional features.

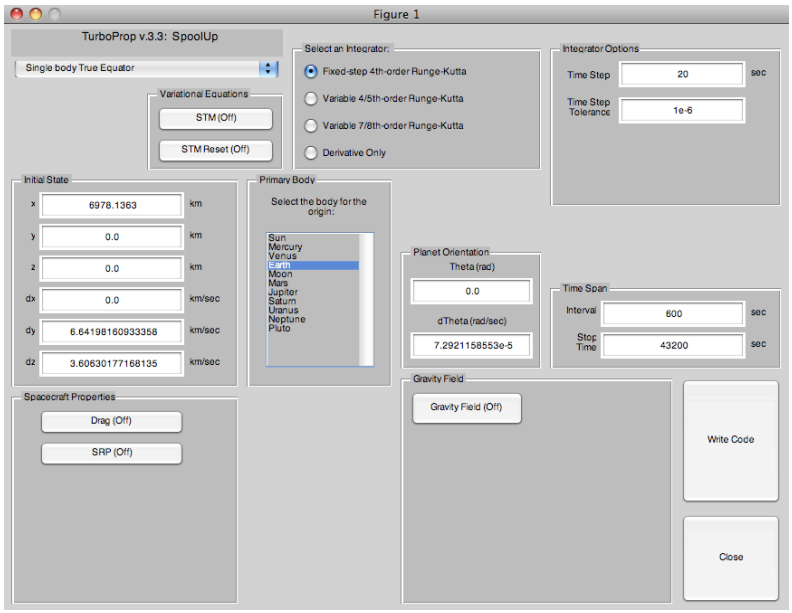

Figure 4: MATLAB GUI with the "Gravity Field" pane enabled. Note "Earth" is selected as the primary body.

References

Dormand, J. R. and R. J. Prince, "A Family of Embedded Runge-Kutta Formulae," *Journal of Computational and Applied Mathematics*, Volume 6, 1980, pp. 19–26.

Gottlieb, R. G., "Fast Gravity, Gravity Partials, Normalized Gravity, Gravity Gradient Torque and Magnetic Field: Derivation, Code and Data," Technical Report NASA Contractor Report 188243, NASA Lyndon B. Johnson Space Center, Houston, TX, February 1993.

Hairer, E., C. Lubich, and G. Wanner, *Geometric Numerical Integration : Structure-Preserving Algorithms for Ordinary Differential Equations*, Number 31 in Springer Series in Computational Mathematics, Springer, New York, 2002.

Hoffman, D. A., "A Set of C Utility Programs for Processing JPL Ephemeris Data," NASA Johnson Space Center, August 3, 1998.

Julier, S. J. and J. K. Uhlmann, "A New Extension of the Kalman Filter to Nonlinear Systems," *Proceedings of SPIE*, Volume 3068, 1997, pp. 182–193.

Konopliv, A., S. W. Asmar, E. Carranza, W. L. Sjogren, and D. Yuan, "Recent Gravity Models as a Result of the Lunar Prospector Mission," *Icarus*, Volume 150, 2001, pp. 1–18.

Leimkuhler, B. and S. Reich, *Simulating Hamiltonian Dynamics*, 1st Edition, Cambridge University Press, New York, 2004.

Lemoine, F. G. R., D. E. Smith, M. T. Zuber, G. A. Neumann, and D. D. Rowlands, "A 70th Degree Lunar Gravity Model (GLGM-2) from Clementine and other tracking data," *Journal of Geophysical Research*, Volume 102, 1997, pp. 16,339,359.

Long, A. C., J. O. Cappellari, C. E. Velez, and A. J. Fuchs, "Goddard Trajectory Determination System (GTDS) Mathematical Theory," Revision 1, July, 1989.

Long, A. C. and T. Lee, "GPS Enhanced Onboard Navigation System (GEONS) Mathematical Specifications," Version 2, Release 2.6, Honeywell Technology Solutions, Lanham, MD, February 23, 2006.

Lundberg, J. B. and B. E. Schutz, "Recursion Formulas of Legendre Functions for Use with Nonsingular Geopotential Models," *Journal of Guidance*, Volume 11(1), Jan.-Feb. 1988, pp. 31–38.

Mohr, P. J., B. N. Taylor, and D. B. Newell, "CODATA Recommended Values of the Fundamental Physical Constants: 2006," Technical report, National Institute of Standards and Technology, Gaithersburg, Maryland, 20899, USA, December 2007.

Murray, C. D. and S. F. Dermott, *Solar System Dynamics*, Cambridge University Press, Cambridge, UK, 1999.

Newhall, X. X. and J. G. Williams, "Estimation of the Lunar Physical Librations," *Celestial Mechanics and Dynamical Astronomy*, Volume 66, 1997, pp. 21–30.

NGA, "Department of Defense World Geodetic System 1984, Its Definition and Relationships With Local Geodetic Systems," Technical Report TR 8350.2, National Geospatial-Intelligence Agency, January 2000.

Standish, E. M., "JPL Planetary and Lunar Ephemerides, DE405/LE405," Jet Propulsion Laboratory Interoffice Memorandum IOM 312F-98-048, Aug. 26, 1998.

Standish, E. M., X. X. Newhall, J. G. Williams, and W. M. Folkner, "JPL Planetary and Lunar Ephemerides, DE403/LE403," Jet Propulsion Laboratory Interoffice Memorandum IOM 314.10-127, May 22, 1995.

Tapley, B., J. Ries, S. Bettadpur, D. Chambers, M. Cheng, F. Condi, B. Gunter, Z. Kang, P. Nagel, R. Pastor, T. Pekker, S. Poole, and F. Wang, "GGM02 - An Improved Earth Gravity Field Model from GRACE," *Journal of Geodesy*, Volume 79(8), 2005, pp. 467–478.

Tapley, B., M. Watkins, J. Ries, G. Davis, R. Eanes, S. Poole, H. Rim, B. Schutz, C. Shum, R. Nerem, F. Lerch, J. Marshall, S. Klosko, N. Pavlis, and R. Williamson, "The Joint Gravity Model 3," *Journal of Geophysical Research*, Volume 101(B12), 1996, pp. 28,029–28,050.

Tapley, B. D., B. E. Schutz, and G. H. Born, *Statistical Orbit Determination*, Elsevier Academic Press, Burlington, MA, 2004.

Vallado, D. and W. McClain, *Fundamentals of Astrodynamics and Applications*, 2nd Edition, Microcosm Press, El Segundo, CA, 2001.