General Use Binary Asteroid Simulator: User Guide

Alex Davis

1 Introduction

The goal of the general use binary asteroid simulator is to provide a simple to use, but fast, tool for both observers and mission designers to predict binary asteroid system behaviors. This software tool accomplishes this by implementing the Hou 2016 realization of the full two-body problem (F2BP) [1]. The F2BP models binary asteroid systems as two arbitrary mass distributions whose mass elements interact gravitationally and result in both gravity forces and torques. To account for these mass distributions and model the mutual gravity of the F2BP, the inertia integrals of each body are computed up to a user defined expansion order. This approach provides a recursive expression of the mutual gravity potential and represents a significant decrease in the computational burden of the F2BP when compared to other methods of representing the mutual potential.

From a software perspective, the tool has been designed and implemented with modularity as a focal point in order to enable a wide set of uses and allow for easy integration into larger tool sets. For this reason the tool was developed as a Python shell script which writes commands to a C++ executable and then post-processes the results. All of the dynamics and integration takes place within the executable such that it can be operated independently from the shell script. This also allows the shell script to be easily modified or replaced to better fit the user's needs. In the standard architecture all interactions with the tool are through the config file and a single command line call is used to perform a simulation.

2 Software Architecture

The architecture of the tool is set up such that a single config file dictates each run of the software. The config file is read in by a shell script responsible for pre and post-processing the inputs and outputs of the executable script. A pre-processed text file is passed from the shell script to the executable script, containing shape file names, integration parameters and initial conditions. The executable script in turn reads this input file, generates the inertia integrals, and begins numerically integrating the system based on user specifications. During the numerical integration, the integrated time steps are written out to two binary files containing the integrated time and state throughout the simulation. The shell script then reads in the binary files and post-processes them based on user defined settings. The post processed output is saved into five separate files: an energy and angular momentum file, an energy and angular momentum fractional conservation file, a Lagrangian states file, a Hamiltonian states file and the "FHamiltonian" states file, which is formatted such that the config file can be set to read in a set of states as initial conditions. Each of the state files also contains the time and potential energy at each step. Fig. 1 below illustrates the flow of this architecture.

2.1 Directory Description

Of key importance to this architecture is the structure of the directory. The software is set up to be run within a separate directory for each run, copied from a "locked" software folder. The generic structure for a directory is summarized in the list below. All files should be at the same directory level unless otherwise specified.

- **hou_cpp_final:** C++ executable
- **hou_cpp.cpp:** C++ executable source
- **Fahnestock files:** see corresponding entry in the File Formats section
- **benchmark_funcs.py:** Python script containing function to read Fahnestock input files

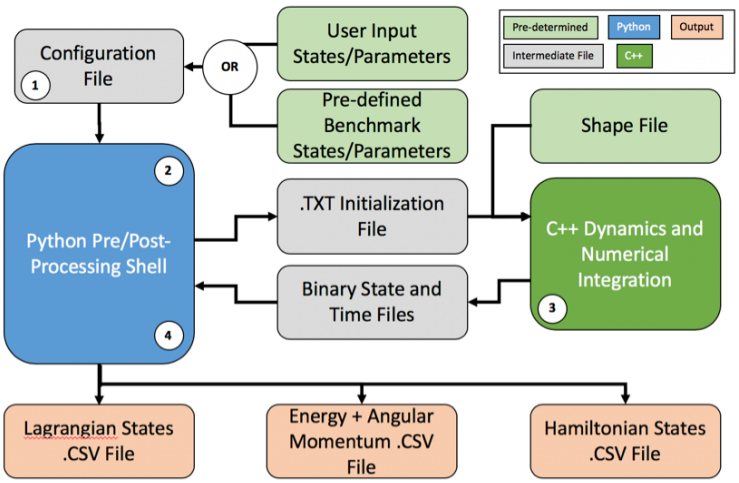

Figure 1: Flow and structure of general use binary asteroid simulator software architecture.

- **coefficient_funcs.py:** Python script containing functions to compute recursive binomial expansion coefficients for the mutual gravity potential
- **hou_config_funcs.py:** Python script containging function to read config file
- **hou_shell_cfg.py:** Python shell script
- **inertia_functions_met.py:** Python script containing functions relating to inertia integral and moments computations
- **potenital+derivs_funcs.py:** Python script containing mutual gravity potential functions
- **write_icfile.py:** Python script containing function to write C++ executable input file
- **hou_config.cfg:** Python config file
- **ic_input.txt:** see corresponding entry in the File Formats section
- **Inertia files:** see corresponding entry in the File Formats sections
- **Post-Processed Outputs:** see corresponding entry in the File Formats sections
- **Shape Model Files:** see corresponding entry in the File Formats sections
- **output_t directory:** contains time output binary, see corresponding entry in the File Formats sections
- **output_x directory:** contains state output binary, see corresponding entry in the File Formats sections

2.2 Config File (hou_config.cfg)

The config file is designed to be the only point of interaction for the user with the software tool. As such it has several sections designed for the various steps in the simulation process. While detailed documentation exists within the config file itself, its various parameters and use cases are detailed here. Before going further it should also be noted that all units in the config file are MKS units, except for density which is in the more common g/cm^3.

Gravity Parameter: The gravity parameter section is a simple, but very important section and is used to define the universal gravity constant. Changes in this value can have substantial effects on the simulated dynamics of a system.

Initial Conditions: In the initial conditions section the user has the ability to define the initial conditions of the system or set the config file to read in data from a separate input file.

- **Fahnestock Input File Flag:** A value of 0 for this flag tells the software to read in the initial states, gravity parameter and densities entered in the config file. If the parameter is instead set to 1, then the values entered in the config file will be ignored and read from the Fahnestock formatted file "initstate_standard_MKS_units" and "systemdata_standard_MKS_units', detailed in the File Formats section. The "FHamiltonian" state output file is formatted to have the same state representations as "initstate_standard_MKS_units" for ease of repeating simulations.

- **Frame Definitions:** The user has no control over the reference frames used, but will need to understand their definitions in order to input the initial conditions. Three frames are used for the analysis, the N, A, and B frames. The N frame is defined as the inertial frame, because this model does not account for Solar gravity or other non-gravitational effects, the inertial frame can be selected arbitrarily. The A frame is defined as the primary body's principal axis, body fixed frame. The B frame is defined as the secondary body's principal axis, body fixed frame. It is assumed that in both body fixed frames that the x-axis aligns with the semi-major axis, the y-axis aligns with the semi-intermediate axis, and the z-axis aligns with the semi-minor axis. It is thus expected that the shape model files reflect this assumptions.

- **Relative Position:** The relative position is defined in the A frame as the distance between the two centers of mass, with each X, Y, and Z entry corresponding to the vector element along that axis.

- **Relative Velocity:** The relative velocity is defined as the velocity of the secondary's center of mass with respect to the primary's center of mass. Each X, Y, and Z entry corresponds to the vector element along that axis.

- **Angular Velocity:** The angular velocities are each defined in the body fixed frame of either the primary or secondary with units of rad/s. Each X, Y, and Z entry corresponds to the vector element along that axis.

- **Rotations Matrices:** There are two rotation matrices, the B into A matrix and and A into N matrix. Each can be defined in two ways by toggling the corresponding "Euler Flag" to 0 or 1. When set to 0 the rotation matrix will be defined using the 9 matrix elements specified as (row, column). If the flag is set to 1 then the Euler angle 1-2-3 set from A to B or N to A defines the rotation matrix. The rotation matrices are specified such that a column vector, defined in A, left-multiplied by the A into N matrix will results in the same column vector expressed in the N frame (similarly a matrix in B would be mapped into A by the B into A matrix).

Integration Settings: In the integration settings section the user is able to control the integration time, precision, and method.

- **Start and Final Time:** These inputs define the time at which the initial conditions occur, as the start time, and the end time for the integration, as the final time. Both should be expressed in seconds.

- **Integrator Flag:** This flag allows the user to select between the three implemented numerical integration tools. The implemented tools are the standard fixed-step RK 4, the fixed-step Lie-Group Variational Integrator (LGVI), and the standard adaptive RK 7(8) Dormand-Prince. With the RK 4 and RK 7(8) methods implemented are fairly standard and documented throughout the literature, while the LGVI is an integrator specific to the F2BP. The LGVI is developed to be a symplectic

integrator, such that it forces the solution to remain on the same energy and angular momentum trajectory manifolds, but also constrains the rotational solutions to lie on the Lie Group, which ensures the rotations are conserved along SO(3) [2]. To select the RK 4, set the flag to 1, for the LGVI the flag is set to 2, and for the RK 7(8) the flag is set to 3. It should be noted that the RK 7(8) does not have an interpolator implemented so the resulting trajectory will not be evenly spaced in time.

- **Fixed Time Step:** When the RK 4 or LGVI are selected this input is the fixed time step for these integrators.
- **Absolute Tolerance:** When the RK 7(8) is selected this input defines the absolute tolerance for the adaptive step to solve for.

Output Settings: The output settings section allows the user to control the post-processing of the simulated dynamics as well as naming of output files.

- **Fixed Output Frequency:** This input allows the user to select the time steps used in the post-processing of the simulation. Selecting a value of 0 indicates that all integrated time steps should be post-processed. This setting can be very computationally expensive for long simulation runs due to the number of time steps. This is also the only option for the RK 7(8) integrator because the time steps are not evenly spaced; if the RK 7(8) is selected the post-processing will use every time step regardless of what is entered in this input. If the entry is set to any value greater than 0 then the post-processing will step through the integrated times by the entered value. Thus any value entered, which is greater than 0, must be a multiple of the fixed integration step. If the entry is set to -1 then a input file of specified times is used by the post-processing loop to seek out the specified times and only post process for these time steps. The specified times must also be multiples of the fixed integration step and the file name must be provided in the Specified Time List File Name entry.
- **Specified Time List File Name:** This input allows the user to specify the input time list if the Fixed Output Frequency is set to -1. The input file should be a one line CSV listing the times of interest. See the File Formats section for more detail on the structure of this file.
- **Case Name:** This input should be a string which will be appended to the end of the output file names. See the File Format section of the user guide for further details on the output file names and formats.

Body Model Definitions: The body model definitions section allows the user to define the shape and density of each body in the binary system.

- **Inertia Integral Generation Flag:** This flag allows the user to tell the software whether or not to compute the inertia integrals during the run. It is strongly suggested that this flag is always set to 1 such that the inertia integrals are computed during the run. This is because the inertia integrals take fractions of a second to compute and avoids any problems with the software using the incorrect inertia integrals based on an old file. If the flag is instead set to 0 it will read in "TDP_[N].mat" and "TDS_[N].mat', where N is the gravity expansion truncation order. More information on these files can be found in the File Formats section of the user guide, but it is important to note that there is not currently a way to read in specific files or generate custom names for the files. This was not deemed a valuable capability because of the speed of inertia integral computation.
- **Shape Flag:** This flag allows the user to select between a spherical, ellipsoidal, or polyhedral shape model for either the primary or secondary body. The spherical and ellipsoidal body inertia integrals are computed via closed form inertia integral equations for ellipsoids, while the polyhedral shape model uses the standard vertex and tetrahedral file format, see the File Format section of the user guide for more information [3]. To select a spherical shape set the flag to 0, for an ellipsoid set the flag to 1, for a polyhedral shape set the flag to 2. If the sphere is selected its radius will be taken as the primary

or secondary "Semi-Major Axis" input. If the ellipsoid is selected each of the semi-axis inputs for the specified body will be used. If the polyhedral shape is selected all of the semi-axis inputs will be ignored and the primary or secondary vertex and tetrahedron files will be used to define the shape. For all three options the body is assumed to be constant density.

- **Semi-Axes:** Each of the semi-axis entries allows the user to define the shape of an ellipsoidal body selection, while the semi-major axis entries allow the user to define the radius of a spherical shape selection. As stated in the frame definitions the major axis is aligned with the body fixed x-axis, the intermediate axis is aligned with the body fixed y-axis and the minor axis is aligned with the body fixed z-axis.
- **Polyhedral Shape Files:** The shape models for polyhedral bodies are defined with the standard vertex and tetrahedron file format, see the File Formats section of the user guide for more details. These inputs allow the user to define the file names for the vertex and tetrahedron shape models for either the primary or secondary body.
- **Densities:** This entry allows the user to select the constant density value for each body. The entry should be provided in g/cm^3 units.

Mutual Gravity Expansion Parameters: This section defines the expansion order to be used for the overall mutual gravity potential expansion as well as the inertia integral generation. The parameters here define an expansion truncation order from 0 to N, where the orders are analogous to those of spherical harmonics. That is to say that order 0 defines the mass, order 2 defines obliquity effects similar to J_2 and C_{22}. For a detailed comparison and conversion between spherical harmonics and the inertia integrals, see the appendices of Tricarico 2008 [4].

- **Gravity Expansion Truncation Order:** This entry allows the user to input the mutual potential calculation truncation order. This truncation order defines the order to which the mutual gravity potential will be computed and will access inertia integral values up to this order. It is of note that if one of the body's inertia integral truncation orders is selected higher than the gravity expansion order, the gravity expansion order will be increased to match the largest inertia integral truncation order.
- **Inertia Integral Truncation Order:** This entry allows the user to define what order the inertia integrals for each body will be computed for. If the body is a sphere or ellipsoid it will not be computed beyond order 4 as there are not closed form solutions derived beyond this order and to increase the order beyond this value would assume more knowledge than is likely available. It is also of note that if one of the bodies inertia integral truncation orders is below the gravity expansion truncation order then all intermediate orders for for that body will be set to 0; this is based on the assumption that the lower inertia integral truncation order is selected due to a lack of knowledge beyond the truncation order

2.3 Python Shell

The Python shell, "hou_shell_cfg.py', pre-processes the config file and passes the simulation parameters to the C++ executable, then reads in the integrated states and times and post-processes them into the output files. The first pre-processing step in the shell is to check the user inputs from the config file and ensure they are correctly entered. During this process the user is notified of the parameters the software has read in and set, in addition to any error notifications based on the user inputs. The shell script first resets the gravity expansion truncation order based on the largest inertia integral truncation order; the user is notified of the selected value by a string output to the terminal. The fixed output frequency is then checked to ensure that it is selected as either a multiple of the integrator step size or one of its other specified options. If this is not the case, the software will give the following error: "Selection of output frequency compared to integration step size is bad'. The integrator selection is next checked to ensure a valid selection has been

made and if a valid selection is made, the integration parameters will be printed to the terminal. If an invalid selection is made the software will give the following error: "Invalid Integrator Selection'. The shape selection and parameters for each body are checked next. If either body is selected as a polyhedron, but the inertia integral truncation order for the body is set below the gravity potential truncation order, then the software will increase the inertia integral truncation order for the body to the gravity potential truncation order and notify the user via a message printed to the terminal. Following this the software will print the shape selection and parameters for each body to the terminal. If one of the shape selections is erroneous the software will give the following error: "Bad Shape Selection for [Primary/Secondary]'.

After the config file inputs have been checked, the shell script writes in the input file for the C++ executable, "ic_inputs.txt', and calls the executable as "hou_cpp_final" from the local directory. For more details on the format of "ic_inputs.txt" read the File Formats section of the user guide. Once the executable finishes running, the time is printed to the terminal and post-processing begins.

The first step in the shell scripts post-processing is to read in the inertia integral tensors from the "TDP_[N].csv" and "TDS_[N].csv" files, where N is the gravity expansion truncation order. The files are written out by the executable after the inertia integrals are computed and represent the primary's inertia integral tensor, P, and the secondary's inertia integral tensor, S. Because the indexing for 3D arrays in NumPy and Armadillo are ordered differently, the CSV data must be rewrapped such that the Armadillo 3rd index is the NumPy first index. Following this step the moments of inertia written by the executable, "IDP.csv" and "IDS.csv', are read in and the various mass parameters are computed from the inertia integrals and moments. With the mass parameters set, the shell script reads in binary integration files of the time and states from the executable and computes the various energy, angular momentum, and state values provided in the output files based on the fixed output frequency entry in the config file. The specific states and parameters computed and reported in each file are described in the File Formats section of the user guide.

2.4 C++ Executable

The C++ executable, compiled from "hou_cpp.cpp', contains all of the software tools for generating the inertia integrals, modeling the F2BP dynamics and integrating the simulation. The executable is designed to be automated based on the inputs provided in "ic_inputs.txt'. As described in the Python shell section this file is generated from the shell script, however given its format described in the File Formats section the "ic_inputs.txt" file could be generated by other means. As it stands in the current architecture the executable is called by the shell script and begins by reading the "ic_inputs.txt" file. The flags, file names, and initial conditions contained in this file are used by the executable to decide the integration path. If the user has requested that the inertia integrals are generated (highly suggested) then the software will first generate the inertia integrals and save the associated inertia integrals and moments into CSV and Armadillo MAT files; the software will concurrently print a message to the terminal that the inertia integrals are being generated. From this point the executable will read in the saved MAT files containing the inertia integrals and moments, in case the user selected to not regenerate inertia integrals, and prepare the integration. Based on the integrator flag selected in the config file the software will then print the name of the integrator being used and begin the integration process. In each of the three integrators the integrated times and states are saved at evenly spaced increments throughout the integration and the user is notified of each save by the string "Saving" and the number of saves being printed to the terminal. For the RK 4 and LGVI each save occurs after a day of integrated time. Because the RK 7(8) is adaptive, its saves occur every 1000 integration steps. In all three cases the states are saved to the binary "outputs_x/x_out.bin" and the times are saved to the binary "outputs_t/t_out.bin'. The goal of this regular saving is two-fold: first it prevents long integration runs from storing large sets of data in temporary memory and secondly it creates a regular storage of the integration in case the process is interrupted. The format of these binary files is specified in the File Formats section. Once the executable completes the integration, its processes end and the two binary data sets are read in by the shell script for post-processing.

3 General Use Instructions

3.1 Installation and Environment Setup

Before the user is able to run the software the Armadillo C++ library must be installed. This tool was developed with Armadillo 8.3, however versions created after 8.3 should not be significantly different enough in their functionality to be problematic. To install Armadillo follow the instructions provided on the sourceforge: **http://arma.sourceforge.net/**. Armadillo is the only non-standard library used by the C++ executable and after it is installed the user should be able to compile the C++ executable. It is suggested that C++11 is specified for compilation as different environments may have compilation problems if it is not specified. The compilation command suggested, and specified at the top of "hou_cpp.cpp', is **g++ -std=c++11 hou_cpp.cpp -o hou_cpp_final -larmadillo**. Here -std=c++11 specifies to the compiler to use C++11, -o hou_cpp_final specifies the name of the executable file which will be called by the Python shell, and -larmadillo tells the compiler to use Armadillo during compilation. Because the shell script expects hou_cpp_final to be the executable file name, be sure that this is the name of the executable compiled.

Once the executable has been compiled the user must set up their Python environment to run the shell script. All of the Python scripts for the software tool were written using version 2.7 so the user must set up a Python environment for that version; Python 3 will not be able to run the shell script without significant rewriting. In addition to ensuring that the Python environment is set up correctly, the user must ensure that the various Python libraries used by the shell script are installed. The necessary libraries are: **scipy, matplotlib, numpy, random, subprocess, os, datetime, configparser, and struct**. These libraries are all commonly used tools and should be available through packages such as conda, pip, or homebrew.

3.2 Preparing a Simulation

Setting up and running a simulation is designed to be a very simple two-step process. The first step is to edit the config file, "hou_config.cfg" to reflect the system and initial conditions of interest. Once the config file is modified the user simply calls the shell script, "hou_shell_cfg.py', from the command line with either python or ipython; the command is thus **python hou_shell_cfg.py** or **ipython hou_shell_cfg.py**. The difference between these two is that ipython will give more detailed and easily read error reports. After the shell script is called it will print a number of messages to the screen updating the user on its progress. These messages are detailed above in the Python Shell and C++ Executable subsections of the Software Architecture section.

It is suggested that for each run of the simulation the directory containing the scripts is copied and run from the copied directory. The goals of this is to avoid the original scripts being edited and to ensure that no data is lost between simulations runs. For a more detailed explanation of why this approach is suggested, read the Python Shell and C++ Executable subsections of the Software Architecture section.

4 Mutual Gravity Potential

In order to model the dynamics of binary asteroids we use F2BP dynamical model. The F2BP describes the mutual gravitational interactions between two arbitrary mass distributions, Fig. 1. To model the mutual gravity of the mass distributions the influence of each infinitesimal mass element of both bodies on all other mass elements in the opposing body must be accounted for. The self gravitation potential is ignored in this implementation of the F2BP as both bodies are assumed to be rigid for the time scales of interest and thus the self potential is constant. The generic form of the mutual gravity potential is a double volume integral over both mass distributions which must be evaluated for a relative separation and orientation of the bodies, Eq. 1.

$$U = -G \int_A \int_B \frac{1}{d} dm_A dm_B \tag{1}$$

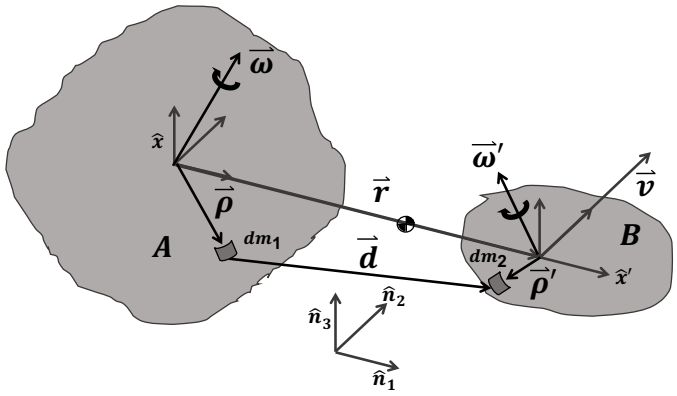

Figure 2: Diagram of initial system geometry. $\hat{x}$ represents the principal frame of the primary, A, $\hat{x}'$ is the principal frame of the secondary, B, and $\hat{n}$ is the inertial frame. The finite mass elements dm_1 and dm_2 must be integrated over the full body using their locations relative to the body centers of mass, $\vec{\rho}$ and $\vec{\rho'}$, to compute the mass parameters and mutual gravity potential.

In the Hou et al. reformulation of the mutual gravity potential, used here, a recursive binomial expansion to order N is applied to Eq. 1 such that it takes for the form of Eq. 2 and 3. Here the recursive coefficient is represented by t_k^n and the binomial expansion coefficients are $a^k_{(i_1,i_2,i_3)(i_4,i_5,i_6)}$ and $b^{n-k}_{(j_1,j_2,j_3)(j_4,j_5,j_6)}$. For these coefficients the superscripts and subscripts serve as indices as descibed in the coefficient definitions, Eq. 4-6. The i and j expansion indices are constrained by Eq. 7 and 8, and summation over these indices sums over the possible combinations of these indices based on the values of k and n and the constraint equations. The variables R and e_x, e_y, and e_z represent the magnitude and unit direction of the relative separation. The variables T_A and T'_B are the inertia integral sets for the primary and secondary bodies, where the prime denotes that the inertia integrals of the secondary are rotated into the frame of the primary[1].

$$U = -G\sum_{n=0}^{N}\frac{1}{R^{n+1}}\tilde{U}_n \tag{2}$$

$$\tilde{U}_n = \sum_{k(2)=n}^{0} t_k^n \sum_{(i_1,i_2,i_3)(i_4 1,i_5,i_6)(j_1,j_2,j_3)(j_4,j_5,j_6)} a^k_{(i_1,i_2,i_3)(i_4,i_5,i_6)} b^{n-k}_{(j_1,j_2,j_3)(j_4,j_5,j_6)} \tag{3}$$
$$e_x^{i_1+i_4} e_y^{i_2+i_5} e_z^{i_3+i_6} T_A^{(i_1+j_1),(i_2+j_2),(i_3+j_3)} T_B'^{(i_4+j_4),(i_5+j_5),(i_6+j_6)}$$

Where k(2) implies stepping by 2 as opposed to 1

$$t^n_{k+2} = -\frac{(n-k)(n+k+1)}{(k+2)(k+1)} \tag{4}$$

$$a^k_{(i_1,i_2,i_3)(i_4,i_5,i_6)} = a^{k-1}_{(i_1-1,i_2,i_3)(i_4,i_5,i_6)} + a^{k-1}_{(i_1,i_2-1,i_3)(i_4,i_5,i_6)} + a^{k-1}_{(i_1,i_2,i_3-1)(i_4,i_5,i_6)} \tag{5}$$
$$-a^{k-1}_{(i_1,i_2,i_3)(i_4-1,i_5,i_6)} - a^{k-1}_{(i_1,i_2,i_3)(i_4,i_5-1,i_6)} - a^{k-1}_{(i_1,i_2,i_3)(i_4,i_5,i_6-1)}$$

$$b^k_{(j_1,j_2,j_3)(j_4,j_5,j_6)} = b^{k-2}_{(j_1-2,j_2,j_3)(j_4,j_5,j_6)} + b^{k-2}_{(j_1,j_2-2,j_3)(j_4,j_5,j_6)} + b^{k-2}_{(j_1,j_2,j_3-2)(j_4,j_5,j_6)} \tag{6}$$
$$+b^{k-2}_{(j_1,j_2,j_3)(j_4-2,j_5,j_6)} + b^{k-2}_{(j_1,j_2,j_3)(j_4,j_5-2,j_6)} + b^{k-2}_{(j_1,j_2,j_3)(j_4,j_5,j_6-2)} - 2b^{k-2}_{(j_1-1,j_2,j_3)(j_4-1,j_5,j_6)}$$
$$-2b^{k-2}_{(j_1,j_2-1,j_3)(j_4,j_5-1,j_6)} - 2b^{k-2}_{(j_1,j_2,j_3-1)(j_4,j_5,j_6-1)}$$

$$k = i_1 + i_2 + i_3 + i_4 + i_5 + i_6 \tag{7}$$
$$n - k = j_1 + j_2 + j_3 + j_4 + j_5 + j_6 \tag{8}$$

4.1 Inertia Integrals

Central to this reformulation of the mutual potential is the use of the inertia integrals to describe the mass distribution. This aspect of the reformulation is accomplished by the application of a Legendre polynomial expansion to describe the mass distributions, where the Legendre coefficients are referred to as inertia integrals. In this way the inertia integrals can be considered analogous in use to spherical harmonics[4]. The mathematical form of the inertia integrals is similar to that of the moments and products of inertia for a rigid body wherein each term represents the mass distribution about some axis, however the inertia integrals are expanded to order N whereas moments of inertia are linear combinations of second order inertia integrals. Here we provide the general form of an inertia integral along with the 0th and 2nd order inertia integrals in terms of the mass, moments and product of inertia, Eq. 9-16.

$$T^{l,m,n} = \frac{1}{M}\int_B x^l y^m z^n dm, \text{ where } l+m+n = N \tag{9}$$
$$M = T^{0,0,0} \tag{10}$$
$$I_{xx} = T^{0,2,0} + T^{0,0,2} \tag{11}$$
$$I_{yy} = T^{2,0,0} + T^{0,0,2} \tag{12}$$
$$I_{zz} = T^{2,0,0} + T^{0,2,0} \tag{13}$$
$$I_{xy} = -T^{1,1,0} \tag{14}$$
$$I_{xz} = -T^{1,0,1} \tag{15}$$
$$I_{yz} = -T^{0,1,1} \tag{16}$$

5 File Formats

In this section the various files used thoughout the simulation tool are discussed in detail. The files are presented broadly in sections ordered by their use during the run of a simulation.

5.1 Fahnestock Files

In the config file the user has the option to input the states and system parameters using the Fahnestock file formats instead of the user inputs. Enabling this input method signals the config file reader to read "initstate_standard_MKS" and "systemdata_standard_MKS" instead of the initial conditions, density and gravity parameter in the config file.

initstate_standard_MKS: This file contains the states in a Hamiltonian form. The file is a single line space delimited text document with its contents ordered as follows:

- **Elements 1-3:** Relative position of secondary with respect to primary, expressed in primary-body-fixed frame (m)
- **Elements 4-6:** Relative linear momentum, expressed in primary-body-fixed frame (kg*m/s)
- **Elements 7-9:** Angular momentum of primary, expressed in primary-body-fixed frame (kg*m^2/s)
- **Elements 10-12:** Angular momentum of secondary, expressed in primary-body-fixed frame (kg*m^2/s)
- **Elements 13-21:** Rotation matrix mapping a vector from secondary-body-fixed frame to primary-body-fixed frame (unitless) and unwrapped by **row** such that the first three elements are the first row of the matrix
- **Elements 22-30:** Rotation matrix mapping a vector from primary-body-fixed frame to inertial frame (unitless) and unwrapped by **column** such that the first three elements are the first column of the matrix

systemdata_standard_MKS This file contains the system parameters, a number of which are not used by this software. The file is a single line space delimited text document with its contents ordered as follows:

- **Element 1:** density of primary (kg/m^3)
- **Element 2:** density of secondary (kg/m^3)
- **Element 3:** volume of primary (m^3)
- **Element 4:** volume of secondary (m^3)
- **Elements 5-13:** standard inertia dyad of primary (kg*m^2)
- **Elements 14-22:** standard inertia dyad of secondary (kg*m^2)
- **Element 23:** mass of primary (kg)
- **Element 24:** mass of secondary (kg)
- **Element 25:** a system mass parameter (kg) = (mass of primary)*(mass of secondary)/((mass of primary) + (mass of secondary))
- **Elements 26-28:** angular velocity of primary expressed in its own body-fixed frame (rad/s)
- **Elements 29-31:** angular velocity of secondary expressed in its own body-fixed frame (rad/s)
- **Element 32:** gravitational constant (m^3/kg/s^2)
- **Element 33:** mass of the sun (kg)
- **Element 34:** a solar radiation pressure constant at 1 AU from the sun (N/m^2)
- **Element 35:** number of meters in 1 AU (m)
- **Element 36:** length normalization constant (m / DU)
- **Element 37:** mass normalization constant = total mass of Didymos system (kg / MU)
- **Element 38:** time normalization constant = DRMs mean motion rate of mutual orbit (rad/s = TU/s)

5.2 Specified Time List File

The specified time list file is only used if the config file Fixed Output Frequency entry is set to -1. This file should be formatted as a single line CSV whose entries are multiples of the fixed integration time step input by the user.

5.3 ic_inputs.txt

The "ic_inputs.txt" file is the input file used by the C++ executable. The file is formatted as a text document wherein each entry is a separate line in the file. Its contents are formatted as follows:

- **Line 1:** gravitational constant (kmˆ3/kg/sˆ2)
- **Line 2:** gravity expansion truncation order
- **Line 3:** primary inertia integral truncation order
- **Line 4:** secondary inertia integral truncation order
- **Line 5:** primary semi-major axis (m)
- **Line 6:** primary semi-intermediate axis (m)
- **Line 7:** primary semi-minor axis (m)
- **Line 8:** secondary semi-major axis (m)
- **Line 9:** secondary semi-intermediate axis (m)
- **Line 10:** secondary semi-minor axis (m)
- **Line 11:** primary shape flag
- **Line 12:** secondary shape flag
- **Line 13:** primary density (kg/kmˆ3)
- **Line 14:** secondary density (kg/kmˆ3)
- **Line 15:** initial time of integration (s)
- **Line 16:** final time of integration (s)
- **Line 17:** primary inertia integral set file name (unused in this implementation)
- **Line 18:** secondary inertia integral set file name (unused in this implementation)
- **Line 19:** primary inertia moments set file name (unused in this implementation)
- **Line 20:** secondary inertia moments set file name (unused in this implementation)
- **Line 21:** primary shape tetrahedron file name
- **Line 22:** primary shape vertex file name
- **Line 23:** secondary shape tetrahedron file name
- **Line 24:** secondary shape vertex file name
- **Line 25-27:** relative position in primary frame (km)

- **Line 28-30:** relative velocity in primary frame (km/s)
- **Line 31-33:** primary angular velocity in primary frame (rad/s)
- **Line 34-36:** secondary angular velocity in secondary frame (rad/s)
- **Line 37-45:** Rotation matrix from primary frame to inertial frame unwrapped row-wise such that the first three entries are the first row of the matrix
- **Line 37-45:** Rotation matrix from secondary frame to primary frame unwrapped row-wise such that the first three entries are the first row of the matrix
- **Line 55:** inertia integral generation flag
- **Line 56:** integrator selection flag
- **Line 57:** integrator fixed time step (s)
- **Line 58:** integrator adaptive tolerance

5.4 Shape Model Files

To model the polyhedral shapes of each body this software uses the standard tetrahedron and vertex file formats to define the shapes. The details of these files for this code are provided herein.

Tetrahedron Files: The tetrahedron files should be a CSV where each row lists three vertices, assuming the origin is the fourth vertex. The ID used for the vertex is their line number indexed from 1 in the vertex file.

Vertex Files The vertex files should be a CSV where each row has four elements. The last three elements should be the X,Y,Z coordinates of the vertex in the corresponding body-fixed frame and expressed in units of meters. The first element of each row can be either a vertex ID or any other value.

5.5 Inertia Files

For any run of the software, there are a total of eight inertia files, half of which are CSV's which are written by the C++ executable to be read in by the Python shell and the other half of which are Armadillo MAT files containing the same information, but in an internal Armadillo binary format. The four files represented in both CSV and MAT format are "TDP_[N]', "TDS_[N]', "IDP', and "IDS'; here the P means the file corresponds to the primary and S means the file corresponds to the secondary. The T files store inertia integrals and the I files store moments of inertia. The following provides a more detailed description of each file's contents:

T or Inertia Integral Files: Because of their 3-axis X,Y,Z definition the inertia integrals are internally stored as 3D arrays whose first index is treated as the x-index, second index is treated as the y-index, and third index is treated as the z-index. A challenge for this software is that the structure of 3D arrays in Python and Armadillo (C++) are not the same based on index. For the MAT formatted files this does not affect their usage as they store the inertia integrals in the Armadillo format for use by Armadillo only. For the CSV files however the Armadillo 3D arrays are unwrapped to create an (N+1)^2 by N+1 2D array save into the CSV file, where N is the inertia integral truncation order. Movement along the columns of the CSV represents movement along the x-index, movement between row blocks of (N+1) represents movement along the z-index and movement between rows within each block of (N+1) rows represents movement along the y-index. The units of the inertia integrals in these files are expressed in km and kg.

I or Inertia Moments Files: The inertia files exist as CSV and Armadillo MAT files, however their format is much simpler because of their lower dimensionality. Both the CSV and MAT versions of the inertia files are a single row of the three moments of inertia for the corresponding body ordered as the major, intermediate, and minor axes of inertia in km, kg units.

5.6 Output Binaries

The output binary files are saved by the executable throughout the integration process such that the data is not lost if the integration is interrupted. Two binary files are saved, t_out.bin and x_out.bin, which contain the times and states respectively at each step of the integration. Both files are standard non-delimited binary files containing double precision values in a single line. The files are saved into, and loaded from, the directories output_t and output_x respectively.

t_out.bin: Each value entered in the single line binary t_out.bin file is a double precision time step computed during the integration process ad expressed in seconds.

x_out.bin: The x_out.bin binary file contains a block of 30 double precision states for each time step during the integration. The contents of each block are as follows:

- **Entry 1-3:** relative position in primary frame (km)
- **Entry 4-6:** relative velocity in primary frame (km/s)
- **Entry 7-9:** primary angular velocity in primary frame (rad/s)
- **Entry 10-12:** secondary angular velocity in secondary frame (rad/s)
- **Entry 13-21:** Rotation matrix from primary frame to inertial frame unwrapped row-wise such that the first three entries are the first row of the matrix
- **Entry 22-30:** Rotation matrix from secondary frame to primary frame unwrapped row-wise such that the first three entries are the first row of the matrix

5.7 Post-Processed Outputs

Once the shell script post-processes the data based on user specification, it saves out the results into four separate CSV files. A standardized naming convention is applied to the file names such that some information about the run is maintained; this can easily be edited int he shell script to add or change the information included in the file name. In the current naming convention [tf] references the end time of the integration in seconds, [of] references the fixed output frequency entry, [h] references the integration fixed step size, [N] reference the gravity expansion order, [integ] references the integrator selection and [case] includes any user specified naming from the case name entry of the config file. Each row of the CSV files corresponds to a post-processing time step, with the columns being specific variables. The units are all provided in MKS

LagrangianStateOut_[tf]_[of]_[h]_[N]_[integ]_[case].csv: This file summarizes the states in a Lagrangian form. The column entries are as follows:

- **Column 1:** time
- **Column 2-4:** relative position in primary frame
- **Column 5-7:** relative velocity in primary frame
- **Column 8-10:** primary angular velocity in primary frame

- **Column 11-13:** secondary angular velocity in secondary frame
- **Column 14-22:** rotation matrix from secondary frame to primary frame unwrapped row-wise such that the first three entries are the first row of the matrix
- **Column 23-31:** rotation matrix from primary frame to inertial frame unwrapped row-wise such that the first three entries are the first row of the matrix
- **Column 32:** mutual potential energy

HamiltonianStateOut_[tf]_[of]_[h]_[N]_[integ]_[case].csv: This file summarizes the states in a Hamiltonian form. The column entries are as follows:

- **Column 1:** time
- **Column 2-4:** relative position in primary frame
- **Column 5-7:** relative linear momentum in primary frame
- **Column 8-10:** primary angular momentum in primary frame
- **Column 11-13:** secondary angular momentum in secondary frame
- **Column 14-22:** rotation matrix from secondary frame to primary frame unwrapped row-wise such that the first three entries are the first row of the matrix
- **Column 23-31:** rotation matrix from primary frame to inertial frame unwrapped row-wise such that the first three entries are the first row of the matrix
- **Column 32:** mutual potential energy

FHamiltonianStateOut_[tf]_[of]_[h]_[N]_[integ]_[case].csv: This file matches the data in "HamiltonianStateOut˙[tf]˙[of]˙[h]˙[N]˙[integ]+[case].csv', but transposes the rotation matrix from the primary frame to the inertial frame so as to match the Fahnestock file formats and be easily usable to initialize an integration by creating a Fahnestock formatted initial state file.

Energy_[tf]_[of]_[h]_[N]_[integ]_[case].csv: This file contains the energy and angular momentum for each post-processing time step as a means to check the integration's accuracy. The column entries are as follows:

- **Column 1:** total energy
- **Column 2-4:** total angular momentum vector

Conservation˙Energy_[tf]_[of]_[h]_[N]_[integ]_[case].csv: This file contains the fractional energy and angular momentum error relative to the initial values for each post-processing time step as a means to check the integration's accuracy. The column entries are as follows:

- **Column 1:** fractional energy error
- **Column 2:** fractional angular momentum magnitude error

Bibliography

[1] X. Hou, D. J. Scheeres, and X. Xin, "Mutual potential between two rigid bodies with arbitrary shapes and mass distributions," *Celestial Mechanics and Dynamical Astronomy*, 2016, pp. 1–27.

[2] T. Lee, M. Leok, and N. H. McClamroch, "Lie group variational integrators for the full body problem in orbital mechanics," *Celestial Mechanics and Dynamical Astronomy*, Vol. 98, No. 2, 2007, pp. 121–144.

[3] W. D. MacMillan, "The theory of the potential," 1958.

[4] P. Tricarico, "Figure–figure interaction between bodies having arbitrary shapes and mass distributions: a power series expansion approach," *Celestial Mechanics and Dynamical Astronomy*, Vol. 100, No. 4, 2008, pp. 319–330.