ISTS-2017-d-115／ISSFD-2017-115

Open Source Implementation of A Fast and Precise N-body Low-Thrust Propagator

By Weichen XIAO,[1] Chit hong YAM,[2] and Wen han CHIU[1]

[1] *Department of Physics, The Hong Kong University of Science and Technology, Hong Kong*
[2] *ispace inc., Tokyo, Japan*

(Received April 24th, 2017)

Predicting the trajectory of a spacecraft accurately is essential to the design and operation of all space missions. We develop an open source numerical propagator in hope that it can become the standard benchmark software of high precision propagator. We demonstrate that this program has the merit of both high precision and high speed, and is highly customizable as well. We are also modifying it to support non-inertial reference frames for more convenient input and output by doing Chebyshev interpolations to the non-inertial central body.

Key Words: Numerical propagator, Low thrust, High speed, High precision

1. Introduction

Predicting the trajectory of a spacecraft accurately is essential to the design and operation of all space missions, particularly for missions that involves multiple bodies and precise control. However to this date, there is no high precision propagator available to the research and industrial community as a benchmark. In this paper we introduce a numerical propagator which aims to fill in this gap.

We demonstrate with a simple low-thrust trajectory that the precision of our program is very high, so that it is capable of practical use in mission design. Compared to another MATLAB version of numerical propagator, it also has the advantage of higher speed as it is written in C++, which is a compiled language. In addition, the use of the NAIF SPICE toolkit together with the boost-odeint library makes our program highly-customizable. We are also modifying the program to handle non-inertial frames by doing Chebyshev interpolation of the non-inertial central body.

We are willing to make the program free and open source to share it among the community of astrodynamics academia and industry. We hope that it will become a standard benchmark software of high precision propagator in the future.

2. Mechanism of the Propagator

2.1. The mathematical problem

The purpose of this program is to numerically calculate the trajectory of a spacecraft in a n-body gravitational field, taking into account the low thrust of the spacecraft and other force exerted on it, solar radiation pressure for example. To achieve this we have to solve the ordinary differential equations

$$x_i''(t) = \sum_j \frac{GM_j x_i(t)}{|\vec{x(t)} - \vec{r_j(t)}|^{\frac{3}{2}}} + \frac{T_i}{m(t)} + a_i \qquad (1)$$

$$m'(t) = -\frac{|\vec{T}|}{u_{eq}} \qquad (2)$$

in which $\vec{x}(t)$ is the position of the spacecraft at time t in terms of kilometres and $x_i(t)$ are the spatial components of it. GM_j and $\vec{r_j(t)}$ are the gravitational parameters and the positions(in km) of celestial bodies at time t. The celestial bodies to be considered should be listed in a file as an input to the program. T_i is the spatial components of the thrust of the spacecraft, $m(t)$ is the mass of it and u_{eq} is the equivalent engine exhaust velocity. a_i represents the acceleration caused by other factors, such as the solar radiation pressure on the space craft.

2.2. Ephemerides

To find the gravitational parameters and positions of celestial bodies, we use the CSPICE toolkit developed by NAIF, NASA. One should follow the instructions on the website of the toolkit to install it, and declare it when compiling the program.

The ID of the celestial bodies to be considered are read from a text file, and the bodvrd_c command of CSPICE reads the GM values from the ephemerides given the ID. Similarly, given the time and the ID, the positions are also found using the spkezr_c command during integration.

It is up to the user to choose the ephemerides to be used with the furnsh_c command. The benchmarking in the following sections will be based on de423.bsp, de430.bsp, naif0010.tls.pc, pck00010.tpc and de-403-masses.tpc.

2.3. Finding numerical solutions to the ODE

The program solves the equations of motion of the spacecraft using the boost-odeint library. To make use of this feature one should install the boost library first. As the user inputs the initial state, the thrust and the flight time in a text file, the program solves the equations numerically with the integrate_adaptive function of the odeint library. In the following sections the integration method is chosen to be the 5th order Runge-Kutta Cash-Karp method. The step size, precision requirements(absolute and relative tolerance) and integration method are again customizable.

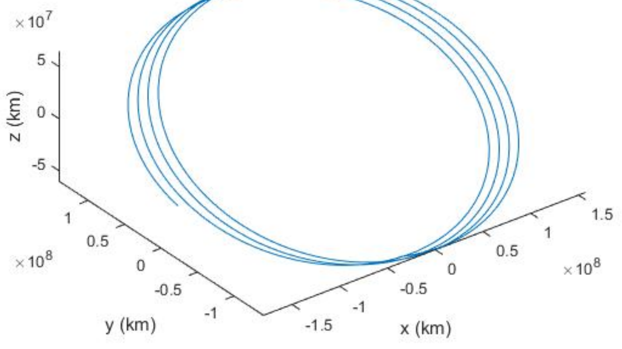

Fig. 1. The test orbit with duration of 1440 days

Table 1. Difference between results of the C++ propagator and the MATLAB propagator

Duration(days)	$\lvert\vec{r}_{difference}\rvert$ (km)	$\lvert\vec{v}_{difference}\rvert$ (km/s)
90	1.45065e-06	6.51478e-13
180	1.34675e-05	3.19821e-12
360	1.74936e-05	1.83252e-12
720	1.67313e-05	1.07238e-12
1440	2.49389e-04	4.2622e-11

3. Benchmarking

3.1. Precision

Our propagator has the advantage of high precision. To demonstrate this, a simple low-thrust trajectory in the J2000 frame was used as a test case. Figure 1 shows the trajectory with duration of 1440 days. Starting from a point about 100000km from the Earth, the spacecraft has a constant low thrust pointing towards the +z direction, eventually end up in an orbit of radius at the same magnitude as the Earth's. The gravity of the Sun, all the planets as well as Pluto and the Moon was taken into account. The orbit was integrated for durations of 90, 180, 360, 720 and 1440 days, with absolute and relative tolerance set to be 10^{-14}. A same test was also done with a MATLAB propagator for comparison.

The results of the two programs are compared in Table 1.

Another method to check the precision of the propagators is to integrate an orbit forward in time first, and use the final state as the initial state for a backward integration. The results of the backward integration are to be compared with the original initial state given.

This test is performed for both of the propagators, as shown in Table 2.

The two tests above indicates that our C++ propagator agrees with the existing MATLAB version. Moreover, its precision is very high, such that the error is less than 1 meter for an integration of 8 years in duration(4 years forward, 4 years backward).

3.2. Speed

Another advantage of this propagator is its high speed. The program is written in C++ which is a compiled language that runs faster than interpreted languages. For the test case mentioned in the previous section, the runtime of the C++ propagator and the MATLAB propagator was also recorded in Table 3.

Table 2. Comparison between the original initia state and the results of the backward integration

Duration(days)	$\lvert\vec{r}_{difference}\rvert$ (km)	$\lvert\vec{v}_{difference}\rvert$ (km/s)
90(C++)	3.29119e-06	1.00799e-12
90(MATLAB)	7.40899e-06	2.90201e-13
180(C++)	1.31978e-05	3.12783e-12
180(MATLAB)	1.46025e-05	1.98345e-12
360(C++)	8.00936e-05	1.56245e-11
360(MATLAB)	6.5907e-05	1.32222e-11
720(C++)	0.000290803	5.72099e-11
720(MATLAB)	0.000113308	2.07391e-11
1440(C++)	0.000983651	1.92833e-10
1440(MATLAB)	4.11109e-06	2.5371e-12

Table 3. Runtime comparion of the low-thrust trajectory test trajectory

Duration(days)	C++ Runtime(sec)	MATLAB Runtime(sec)
90	0.095	0.614
180	0.155	0.755
360	0.283	0.941
720	0.537	1.346
1440	1.061	2.044

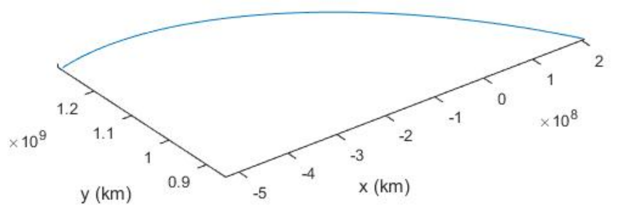

Fig. 2. A section of the Jupiter-Saturn transfer orbit with lower curvature

Table 3 indicates that the C++ propagator is 2-6 times faster than the MATLAB version. In addition, its speed advantage becomes more apparent when it comes to shorter durations.

Moreover, the C++ propagator takes even less time to run for trajectories with less curvature, since less iterations will be needed to numerically solve the ODE given the tolerance. Figure 2 shows a section of transfer orbit from Jupiter to Saturn which was used for demonstration. The time it takes to integrate this trajectory for both of the programs is recorded in Table 4 with respect to duration.

For the same duration, it takes both of the programs less time to integrate this trajectory than the previous one. Again the C++ propagator has a larger speed advantage for shorter durations. Hence, our program may be best suited for search algorithms and optimizers that requires repeated integration of trajectories in small time intervals.

Table 4. Runtime comparion of the transfer orbit test trajectory

Duration(days)	C++ Runtime(sec)	MATLAB Runtime(sec)
90	0.080	0.501
180	0.092	0.541
360	0.112	0.648
720	0.196	0.850
1440	0.244	1.089

Table 5. Comparison of stepper types

Stepper Type	Position error (km)	Runtime (sec)
Cash-Karp 54	0.000983651	1.061
Dormand-Prince 5	0.000296106	1.128
Fehlberg 78	2.7163e-05	0.232

3.3. Integration Methods

The use of odeint in this propagator makes it available for the user to customize the integration method. This is done by defining the stepper type at the beginning of the program. In the previous sections we have been using the Cash-Karp method 54. Here however, we would like to compare three different steppers: the Cash-Karp 54, the Dormand-Prince 5, and the Fehlberg 78. The test case used here is again the 1440 days forward and backward integration in Section 3.1.

Table 5 shows the difference between the integration methods in terms of error and runtime. In this particular test case, the difference in error is too small to be significant, but the Fehlberg 78 method does have a huge advantage in efficiency. The user should choose the integration method according to practical situations, and a few test runs are advised to determine the most suited integration method.

The users are also free to turn on the theoretical error estimation function of odeint. One should refer to the tutorial of odeint for more details.

4. Non-inertial Frame and Chebyshev Interpolation

Currently our program accepts input states with respect to the inertial frame(SSB) only. However it is sometimes more convenient to use a non-inertial frame instead, for example using the Earth frame to study the trajectory of a spacecraft close to the Earth.

The most straightforward way to modify the propagator to be able to deal with non-inertial frame is simply doing the coordinate transformation. We modify the program to transform coordinates in the non-inertial frame into those with respect to SSB, do the integration, and then transform the results back to the non-inertial frame. However the coordinates in SSB may be of several orders larger than those in the non-inertial frame(consider a satellite in the low Earth orbit for example), which is a potential source of error when the integration is carried out.

The other way is to do the integration directly in the non-inertial frame. In this case we have to know the acceleration of the non-inertial frame relative to the SSB. Unfortunately the ephemerides available provides only the position and velocity, but not the acceleration of the planets. To compute the acceleration we plan to do a Chebyshev interpolation of the orbit of the planet.

The Chebyshev polynomials can be expressed as

$$T_0(x) = 1 \tag{3}$$

$$T_1(x) = x \tag{4}$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{5}$$

and their derivative

$$T'_0(x) = 0 \tag{6}$$

$$T'_1(x) = 1 \tag{7}$$

$$T'_{n+1}(x) = 2T_N(X) + 2xT'_n(x) - T'_{n-1}(x) \tag{8}$$

While the Chebyshev coefficients and the interpolation function being

$$c_j = \frac{2}{N}\sum_{k=1}^{N} f(cos(\frac{\pi(k-\frac{1}{2})}{N}))cos(\frac{\pi j(k-\frac{1}{2})}{N}) \tag{9}$$

and

$$f'(x) \approx \sum_{k=0}^{N-1} c_k T'_k(x) \tag{10}$$

in which N is the number of Chebyshev nodes, and x runs from -1 to 1.

Therefore by rescaling the time from -1 to 1 and finding out $\vec{v}(t)$ at each Chebyshev node, we can interpolate the velocity of the non-inertial central body, hence finding out its acceleration at each time.

The code of Chebyshev interpolation has already been finished and the benchmarking is ongoing.

5. Getting the Program

The propagator is going to be made free and open source. Its source code is to be posted on GitHub as a public project soon.

6. Conclusion

In this paper we introduced a numerical propagator written in C++ with the help of the boost-odeint library and the NAIF SPICE toolkit. We demonstrated that the propagator has the advantage of precision and speed, and being highly customizable in terms of ephemerides and integration methods. We are also modifying the program to deal with non-inertial frames by doing the Chebyshev interpolation of the central body. We would like to make the propagator free and open source, hoping that it can fill the gap and become the standard program among the astrodynamics community.

References

1) Yam, C. H. and Kawakatsu, Y.: *GALLOP: A Low-Thrust Trajectory Optimization Tool for Preliminary and HighFidelity Mission Design*, pp.1–5.
2) Acton, C., Bachman, N., Liukis, M., Semenov, B. and Wright, E.: Dynamical Systems, The SPICE Toolkit, The NASA Planetary Science Division's Ancillary Information System, https://naif.jpl.nasa.gov/naif/toolkit.html (accessed April 24, 2017).
3) Ahnert, K. and Mulansky, M.: ”odeint tutorial,”, http://headmyshoulder.github.io/odeint-v2/doc/index.html, (accessed April 24, 2017).
4) ”HORIZONS Web-Interface”, https://ssd.jpl.nasa.gov/horizons.cgi, (accessed April 24, 2017).
5) ”WebGeocalc”, The NASA Planetary Science Division's Ancillary Information System, https://naif.jpl.nasa.gov/naif/webgeocalc.html., (accessed April 24, 2017).
6) Weisstein, E. W.: *Chebyshev Approximation Formula*, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/ChebyshevApproximationFormula.html (accessed April 24, 2017).