Modern N-Body Integrators for Solar System Dynamics

Harvard Applied Math 225
Michael S. Emanuel

1 Introduction

1.1 Abstract

In this paper I explore the N-body problem and state of the art software to solve it numerically. The first section includes a review of the problem and its history and relevance. I explain why I chose to learn to use a modern library, `REBOUND`, rather than coding my own from scratch. I next explore the mathematical background underpinning the N-body problem, covering the Hamiltonian formalism of the gravitational problem and the idea of symplectic integrators.

Three integration schemes are then presented in detail, covering both the algorithms and mathematical theories: the leapfrog integrator, the Wisdom-Holman integrator, and an adaptive non-symplectic integrator in `REBOUND` called `IAS15`. The next section includes the results of simulating the solar system for 100 or 100,000 years using these three integrators. I show how to use `REBOUND`, visualize the results, and compare the performance of the three integrators.

In the final section I simulate the near earth approach of the asteroid Apophis, which is expected to come very close to our planet in April 2029. I matched predictions made NASA to within three significant figures and the time of nearest approach to a tolerance of one minute. I also simulated a range of possible trajectories based on the uncertainty in the current estimates of the orbital elements for Apophis. This calculation shows that there is no risk of Apophis striking the earth in the next 100 years, but that if the parameter uncertainty increased by a factor of 100, an earth strike would be a remote but real possibility, occurring once in a sample of 1000 trials.

1.2 Motivation - Orbital Dynamics and Asteroid Strikes

The calculation of planetary orbits is arguably the canonical problem in mathematical physics. Isaac Newton invented differential calculus, which he called the method of fluxions, while working on this problem, and used his theory of gravitation to solve it. While the Newtonian theory of gravitation is mature, efficiently computing the motion of a large system of particles under gravity (commonly called the N-body problem) is very much an open problem. Indeed, as we will see in this paper, significant practical advances have been made in the last five years.

Without exaggeration, we can say that the challenge of computing accurate orbits is important to the security of life on earth. Even elementary school children know that a large meteor strike is theorized to have led to the extinction of dinosaurs 66 million years ago. [1] As members of the Harvard community, we are privileged to be in close proximity to a global center of excellence in this field, the Minor Planet Center (MPC) hosted at the Harvard-Smithsonian Center for Astrophysics. Today NASA devotes significant resources to tracking potentially dangerous asteroids. The MPC is at the front lines of these efforts, compiling detections from around the world, and analyzing them to catalog newly discovered asteroids. The MPC transmits estimated orbits of newly discovered asteroids to NASA, which is then responsible for estimating the risk posed and formulating a response if appropriate. While this might sound like the plot of the 1998 science fiction movie Armageddon, it's all real.

[1] The Cretaceous-Paleogene extinction event was hypothesized by Walter and Luis Alverez to have been caused by a meteor strike in 1980. Their theory was given a substantial boost after the discovery of the Chicxulub crater in the 1990s and is now widely accepted.

1.3 N-Body Integrators - Classical and Modern

I am planning to write a Masters Thesis on the topic of using neural networks to approximately solve solar system orbits and map detections to orbits, so I have a practical interest in this problem. To train a neural network, I will need a large quantity of training data, which in this case will be synthetic data- the output of many simulator runs. Because the N-body problem is so important, a lot of the software that solves it is, to put it diplomatically, ancient. There are still computer codes floating around that trace a direct lineage to events depicted in the movie Hidden Figures, which described the space race between the U.S.A. and U.S.S.R. from the perspective of three African American women whose contributions at the time were not widely recognized. Also making a cameo appearance in this movie were IBM computers that filled up entire rooms and Fortran codes that ran on ... punch cards. Anecdotal evidence suggests that more than a few people with tangential interest in orbits (though not necessarily professional astronomers) are still using what can fairly be described as legacy codes.

My original plan for this project was to make a start on porting one of the most popular legacy Fortran integrators, called `MERCURY`, into modern C++. The idea was to use what I have learned in AM 225 to write a piece of software that could be useful to me and perhaps a wider audience. I was lucky enough to meet Matt Holman and his colleague Matt Payne while working on a project at the end of CS 209b. The two Matts were kind enough to tell me that computational astronomers have in fact made substantial efforts to modernize in recent years. They pointed me to an open source project called `REBOUND` that is available on GitHub. This project does everything I was hoping to do and much more, much better than I could have any hope to do myself in any realistic time budget.

I have a theory that learning can often be separated into "direct learning" and "meta-learning." To take an example from music, a piano student can learn to play smoothly with a *legato* technique. A harder skill to master by far is learning how to practice effectively– how to listen carefully to what you are playing and make a focused effort to improve it. I know a number of professional musicians, and most report that they only learned to practice effectively in their late teens or early 20s, typically after 15 years of serious music study. In the domain of computer programming to solve applied math problems, we have learned many powerful skills in AM 225. But there is also the meta-learning problem: when do you write a computer program yourself, and when do you use code written by someone else? I call this question the "roll your own" problem and have a standing joke that any time you ask it, the answer is probably to use a library. [2]

After spending several days reviewing the documentation, example problems, and source code of `REBOUND` I determined that my efforts would be far better spent learning how to use this library to write a program to achieve my goals, rather than attempting to recreate a small fragment of its capabilities from scratch. Accordingly, I revised the goals of this project. In this project I will explain the mathematical theory behind three types of integrators that have been used to solve the N-body problem numerically. I will then use the `REBOUND` library to compare the performance of these integrators on simulating our solar system. Finally I will simulate the near earth approach of the asteroid Apophis.

[2] At the risk of sounding like a dinosaur, I will point out here that one lesson from this experience is that there is no substitute to communicating with domain experts. After extensive online research both on my own and with Professor Rycroft, I had found numerous integrators including `MERCURY`, but it was only after speaking to the two Matts from the CFA that they told me about `REBOUND`.

2 Mathematical Background

2.1 Hamiltonian Formalism of the Gravitational Problem

Newton's second law, familiar to high school physics students around the world, states that $\mathbf{F} = m\mathbf{a}$ where $\mathbf{F}$ is the sum of all forces acting on a body, m is its mass, and a is its acceleration. In standard SI units used in modern physics, m is a scalar measured in kilograms (kg), $\mathbf{a}$ is a 3-vector in meters/second^2 and $\mathbf{F}$ is a 3-vector in Newtons or kg $\cdot$ meter/second^2. [3] Newton's law of universal gravitation states that all objects attract each other with a force that is proportional to the product of their masses over the square of the distance between them:

$$F = G\frac{m_1 m_2}{r^2}$$

Here m_1 and m_2 are the masses of two objects, again in kg. r is the distance between them, the standard Euclidean 2-norm in 3 dimensional space, measured in meters. F is the magnitude of the force in Newtons, a scalar quantity. $\mathbf{F}$ is a 3 dimensional vector pointing between the two masses; gravity is an attractive force that pulls each object toward the other. The universal gravitational constant G is an important physical constant. In SI units the gravitational constant is $G = 6.67508 \times 10^{-11}\,\mathrm{m}^3 \cdot \mathrm{kg}^{-1} \cdot \mathrm{s}^{-}2$. These two equations together completely define the dynamics of the N-body problem.

Classical mechanics can also be formulated in Lagrangian and Hamiltonian formalisms. While these formalisms may at first appear to be a tautological change of variables when limited to Cartesian coordinates, they provide immediate benefits in other coordinate frames, and have proven to be remarkably fruitful. Their main strength lies is in encouraging different ways of thinking about problems. As anyone who has studied the Schrodinger equation in an undergraduate physics course can attest, some of these insights have extended to statistical physics and quantum mechanics in ways that would have been far less intuitive starting from Newton's second law. [4]

A Hamiltonian system can be defined in the abstract by a real-valued Hamiltonian function $\mathcal{H} = \mathcal{H}(q, p, t)$ with time evolution governed by:

$$\frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} \quad , \quad \frac{d\mathbf{q}}{dt} = +\frac{\partial \mathcal{H}}{\partial \mathbf{p}}$$

This is still an abstract mathematical formalism that is not limited to classical mechanics. Systems of this kind are called Hamiltonian Systems and have a number of special properties. If the Hamiltonian function $\mathcal{H}$ is time invariant, i.e. if it does not depend directly on t, then $\mathcal{H}$ is a conserved quantity as the system evolves. The system has a symplectic geometry and the volume of a region in phase space $d\mathbf{q} \wedge d\mathbf{p}$ is preserved as the system evolves in time.

In the case of Hamiltonian mechanics, the Hamiltonian function $\mathcal{H} = T + V$ is the total energy in the system. T is the kinetic energy and V is the potential energy. The position variables $\mathbf{q}$ can be the positions of the particles in space, and their conjugate momenta $\mathbf{p}$ are the regular momentum variables $\mathbf{p} = m\mathbf{v}$. The conserved quantity $\mathcal{H}$ is equivalent to conservation of energy. It is important to note that the $\mathbf{q}$ and $\mathbf{p}$ variables do not have to be Cartesian spatial variables. Any set of spatial variables $\mathbf{q}$ that uniquely parameterizes the location of each object can be used. The momentum $\mathbf{p}$ then generalizes to what is called the "conjugate momentum", which is the partial derivative of kinetic energy T with respect to $\dot{q}$, i.e. $p_i = \partial T / \partial \dot{q}_i$. The classical example of a non-Cartesian coordinate in a Hamiltonian is an angle θ with conjugate momentum L, the angular momentum.

We can quickly see that this version is equivalent to the Newtonian formulation. Going back to high school physics, the gradient of the potential energy with respect to spatial variables x, y and z is the force acting on an object from the influence of the field with that potential. Therefore $\partial V / \partial \mathbf{q} = -\mathbf{F}$. The potential energy of a conservative field depends only on position, not velocity, so $\partial V / \partial \mathbf{p} = 0$. Adding these, we find

[3] See e.g. Newton's Laws of Motion

[4] I certainly found it beneficial to concurrently take Math 117 (classical mechanics) when I took Physics 143a (quantum mechanics) with Professor Michael Tinkham when dinosaurs still roamed in Cambrdige in the fall of 1996.

$-\frac{\partial \mathcal{H}}{\partial \mathbf{q}} = \mathbf{F}$.
On the left side of the first Hamilton equation,

$$\frac{d\mathbf{p}}{dt} = \frac{d}{dt}(m\mathbf{v}) = m\frac{d\mathbf{v}}{dt} = m\mathbf{a}$$

The first Hamilton equation is thus equivalent to $\mathbf{F} = m\mathbf{a}$.
The kinetic energy T of a moving object is

$$T = \frac{1}{2}mv^2 = \frac{1}{2m}(m\mathbf{v}) \cdot (m\mathbf{v}) = \frac{1}{2}\mathbf{p} \cdot \mathbf{p}$$

Taking the gradient of this expression with respect to the momentum $\mathbf{p}$ we can see that $\partial T/\partial \mathbf{p} = (1/m)\mathbf{p} = \mathbf{v}$. The second Hamilton equation is just saying that the change in coordinates $\dot{q}$ is equal to the velocity vector $\mathbf{v}$, which is true by definition. [5]

We are now ready to put the pieces together and state the Hamiltonian formulation of the N-body problem. We have already seen that the kinetic energy of each moving particle is $p_i^2/2m_i$. The gravity force takes the form $F = k/r^2$ where $k = Gm_1m_2$. Integrating $\int_{-\infty}^{r} t^{-2}dt = -r^{-1}$, we can see that the gravitational potential energy between a pair of objects is given by $V(r) = -Gm_1m_2/r$ where r is the distance between them. The Hamiltonian for the N-body system is therefore

$$\mathcal{H} = \sum_{i=0}^{N-1} \frac{p_i^2}{2m_i} \quad - G \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} \frac{m_i m_j}{r_{ij}}$$

Here $\mathbf{p_i}$ is the momentum of particle i, a 3 vector; and p_i^2 is its magnitude, a scalar. r_{ij} is similarly the scalar distance between particles i and j with positions customarily denoted r_i and r_j respectively. I follow the convention of Rein et. al. here and number the N bodies from 0 to $N-1$.

2.2 Symplectic Integrators

A symplectic integrator is a numerical integration scheme for solving Hamiltonian systems with a particular desirable property. As we saw above, the true Hamiltonian flow conserves volume in phase space. In the language of differential forms, it preserves the symplectic two-form $d\mathbf{p} \wedge d\mathbf{q}$. A symplectic integrator is one which also preserves this two-form. A corollary of this property is that symplectic integrators also preserve as an invariant a conserved quantity that is a slightly perturbed version of the true Hamiltonian $\mathcal{H}$. This in turn implies that symplectic integrators have small and bounded errors for the true Hamiltonian, even when integrated over long periods of time. In other words, symplectic integrators come very close to conserving energy, and in particular don't have a bias in the energy due to the scheme. This is a valuable property, because when it fails, the effects can render an integrator useless.

A classic example of the failure of a non-symplectic integrator is a naive attempt to solve the (stable and well conditioned!) two body problem using forward Euler. No matter how small the time step dt is chosen to be, this method will fail spectacularly. It has a slight bias and consistently increases the energy in the system with every step. It also increases volume in phase space. All the planets will fly away from the sun, and the people in the simulation will get very cold and die. That's sad.

[5] Here we use the notation $\dot{q}$ to denote the derivative of q with respect to time. Interesting historical note - the dot notation for time derivatives is due to Newton in the Method of Fluxions, while the familiar dq/dt notation is due to Leibniz. Perhaps for these historical reasons, mechanics is one of the few areas in which the dot notation is still used widely.

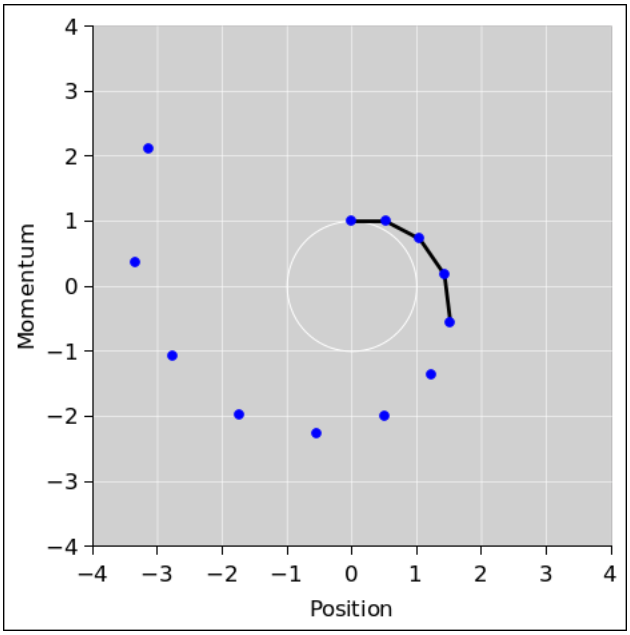


Figure 1: A non-symplectic first order Euler step scheme for the two body problem. The energy drifts higher and the planet is spuriously ejected from the system. [6]

A backwards Euler scheme has an analogous problem but in the opposite direction. Volume is contracted in phase space, and the system loses energy with every time step, regardless of how small the time step is set. In this simulation, all the planets will crash into the sun, and the people will burn up and die. That is also sad.

Symplectic integrators are a popular choice because it is often possible to modify a simple non-symplectic scheme to make it symplectic. To continue with the first order Euler method example, a small change to the naive explicit Euler method leads to the semi-implicit Euler method. This is no harder to program and will have comparable runtime, but behaves much better. A common approach to devising symplectic integration schemes is to split the Hamiltonian. Any Hamiltonian which is *separable* in the sense $\mathcal{H}(p,q) = T(p) + V(q)$ is amenable to this treatment. This special case covers a broad range of important problems, including particles in conservative fields and the N-body problem in particular.

Because of their attractive properties, some practitioners appear to have adopted the view that if it isn't symplectic, it isn't good. I believe that this is the reason that the ideas behind the IAS15 integrator (presented below) were not tried until so recently. Before we see the last and most complicated integrator, though, we will warm up with a very simple but surprisingly effective integrator: leapfrog.

2.3 The Leapfrog Integrator

The primary appeal of the leapfrog integrator is its simple intuition and ease of coding. For some problems, it can be a reasonable choice; in particular it is the preferred method in Hamiltonian Monte Carlo simulations. But I would not recommending doing any serious work with it for celestial mechanics. I present it here as a simple baseline that is used in comparison with the two more sophisticated integrators presented below.

The leapfrog integration algorithm is very simple. Suppose that the acceleration $\mathbf{a}$ is given as a function of only the positions $\mathbf{x}$, an assumption which holds for the N-body problem. Write $\mathbf{x}_i$ and $\mathbf{v}_i$ for the position and velocity at time step i. The most common statement of the leapfrog algorithm updates velocity and

[6] Chart due to www.av8n.com

position with different indices separated by one half:

$$\begin{aligned}\mathbf{a}_i &= \mathbf{F}(\mathbf{x}_i)\\ \mathbf{v}_{i+1/2} &= \mathbf{v}_{i-1/2} + \mathbf{a}_i \Delta T\\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{v}_{i+1/2}\Delta T\end{aligned}$$

The intuition behind these updates is clear. We approximate the acceleration acting on the time interval $[i-1/2, i+1/2]$ with the snapshot $\mathbf{a}_i$ at the midpoint. We similarly approximate the motion over time interval $[i, i+1]$ with the velocity snapshot $\mathbf{v}_{i+1/2}$ taken at the midpoint. Although it might look like a first order algorithm, because of the symmetric steps it is in fact second order. It is also a symplectic integrator.

I have to admit that even though this algorithm isn't nearly as effective as the state of the art integrators presented below, I still like it a lot. It's so intuitive that I was able to program it without a single mistake on the very first try. I used it to simulate the motion of the planets with a hand written Python code for a project in CS 207.

2.4 The Wisdom-Holman Symplectic Integrator

2.4.1 The Wisdom-Holman 1991 Paper

The Wisdom-Holman symplectic integrator is an elegant and sophisticated integrator. It was first described in a seminal paper by Wisdom and Holman in 1991, and the ideas behind it were widely adopted. [7] The 1991 paper is still commonly cited in the modern literature on N-body integrators. I have to admit that after reading it twice, I found it to be difficult to understand in parts. Holman and Wisdom called their approach the mapping method, and based it on the decomposition of the Hamiltonian into four parts

$$\mathcal{H} = \mathcal{H}_{\rm Kepler} + \mathcal{H}_{\rm Orbital} + \mathcal{H}_{\rm Resonant} + \mathcal{H}_{\rm Secular}$$

This approach is specifically tailored to systems with a single dominant central mass. Of course the solar system satisfies this property; the eight planets comprise approximately 0.0014 solar masses, of which Jupiter is approximately 0.0010. [8] The terms in $\mathcal{H}_{\rm Kepler}$ represent the interaction of each body with a large central mass. The terms in $\mathcal{H}_{\rm Orbital}$ represent "rapidly oscillating terms which depend on the mean longitudes of the bodies but are not resonant in the region of interest". The terms in $\mathcal{H}_{\rm Resonant}$ are analogous to $\mathcal{H}_{\rm Orbital}$, but are resonant. Here is an example of resonant terms. There is a distance in the asteroid belt at which the orbital period is close to 2/5 of Jupiter's 12 year period. [9] This region is clear of asteroids because any objects that used to be in orbits there were ejected due to repeated resonant interactions with Jupiter. Holman and Wisdom argued that orbital terms had high frequencies that tended to cancel out and could thus be safely ignored. They also argued that high frequency terms of similar magnitude could be strategically *added* to the Hamiltonian.

After the discussion of resonances, Holman and Wisdom pivoted to presenting a separation of the Hamiltonian that I found to be far more intuitive and comprehensible:

$$\mathcal{H} = \mathcal{H}_{\rm Kepler} + \mathcal{H}_{\rm Interaction}$$

The interaction term $\mathcal{H}_{\rm Interaction}$ has the potential energy of gravitational attraction between the various planets. A further complication arises because either of the two obvious coordinate choices (center of mass or heliocentric) introduces problems for the analysis. The goals in choosing a coordinate frame are for the

[7] The Holman who coauthored this paper is the same Matt Holman who is the big cheese at the Minor Planet Center. Except he had hair back in 1991.

[8] See Solar System and Jupiter for masses of the sun : solar system and Jupiter, respectively.

[9] See Orbital Resonance and the discussion of the Kirkood gaps.

kinetic energy term to have a diagonal sum of squares form without cross terms and to allow updates to be done one at a time.

2.4.2 Jacobi Coordinates in Celestial Mechanics

The classical solution to this dilemma is to use Jacobi coordinates to describe the positions of the planets in the system. In the Jacobi coordinate system, the particles are sorted in ascending order of their distance from the central mass. Each particle is measured from an origin that is the center of mass of the particles with lower index. This presentation now follows the Rein and Tamayo paper on their implemenation of the WHFast integrator.

Let the N particles be indexed from 0 to $N-1$ with the Sun at index 0. Let m_i be the mass of planet i and $\mathbf{r}_i$ be its position. Then the Jacobi coordinates $\mathbf{r}'_i$ are given by:

$$\mathbf{r}'_i = \mathbf{r}_i - \mathbf{R}_{i-1} \quad \text{for } i = 1, \dots N-1$$
$$\mathbf{R_i} = \frac{1}{M_i} \sum_{j=0}^{i} m_j \mathbf{r}_j$$
$$M_i = \sum_{j=0}^{i} m_i$$

The velocity and acceleration transform identically because Jacobi coordinates are a linear function of the Cartesian coordinates. The momenta conjugate to the Jacobi coordinates $\mathbf{r}'_i$ and their associated Jacobi masses are given by:

$$\mathbf{p}'_i = m'_i \dot{\mathbf{r}}'_i = m_i \mathbf{v}'_i$$
$$m'_i = m_i \frac{M_{i-1}}{M_i} = \frac{m_i M_{i-1}}{m_i + M_{i-1}}$$

That is, the Jacobi mass m'_i is the reduced mass of $_i$ and M_{i-1}. Veterans of Harvard Physics 16 or a similar course will recognize that the reduced mass μ in the two body problem is the special case for $N = 2$ that generalizes up to Jacobi coordinates.

The above formulas start at $i = 1$. For the 0-th coordinate, a special convention is used:

$$\mathbf{r}'_0 = \mathbf{R}_{N-1}, \quad m'_0 = M_{N-1}, \quad \mathbf{p}'_0 = \sum_{j=0}^{N-1} p)j$$

The 0-th position thus corresponds to the whole system: it is located at the center of mass, has mass equal to the full system, and momentum equal to that of the whole system

The virtue of this coordinate system is that the Hamiltonian in the Jacobi frame now takes a form which is particularly convenient to solve:

$$\mathcal{H} = \sum_{i=0}^{N-1} \frac{\mathbf{p_i}'^2}{2m'_i} - \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} \frac{G m_1 m_2}{|\mathbf{r}_i - \mathbf{r}_j|}$$

The Hamiltonian still has a kinetic term that is a sum of squares with no cross terms. [10]

[10] Please note that the absence of primes on the $\mathbf{r}_i$ in the denominator of the potential term is *not* a typo. The kinetic term is re-written in the Jacobi coordinate system, but the pairwise interactions in the potential term remain in the original coordinates.

2.4.3 Hamiltonian Splitting for the Wisdom-Holman Integrator

The goal now is to express the Hamiltonian as a sum of parts. We want the parts to be separable (commute with each other) as much as possible. We also want one term to dominate the others. At this point, Hanno and Rein add and subtract a term representing the gravitational potential of each reduced mass m_i' with the cumulative mass M_i

$$\mathcal{H}_{\pm} = \sum_{i=1}^{N-1} \frac{G m_i' M_i}{|\mathbf{r}_i'|}$$

The Hamiltonian is then grouped into three terms, $\mathcal{H}_0$, the intertial drift of the center of mass in a straight line; $\mathcal{H}_{\text{Kepler}}$, the interaction of each planet with the center of mass indexed lower; and $\mathcal{H}_{\text{Interaction}}$, the gravitational attraction between planets.

$$\mathcal{H} = \underbrace{\frac{\mathbf{p}_0'^2}{2m_0'}}_{\mathcal{H}_0} + \underbrace{\sum_{i=0}^{N-1} \frac{\mathbf{p_i}'^2}{2m_i'} - \sum_{i=1}^{N-1} \frac{G m_i' M_i}{|\mathbf{r}_i'|}}_{\mathcal{H}_{\text{Kepler}}} + \underbrace{\sum_{i=1}^{N-1} \frac{G m_i' M_i}{|\mathbf{r}_i'|} - \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} \frac{G m_1 m_2}{|\mathbf{r}_i - \mathbf{r}_j|}}_{\mathcal{H}_{\text{Interaction}}}$$

Now more good things happen if you want to solve this numerically. The terms in $\mathcal{H}_{\text{Kepler}}$ split up into separable parts

$$\left(\mathcal{H}_{\text{Kepler}}\right)_i = \frac{\mathbf{p}_i'^2}{2m_i'} - \frac{G m_i' M_i}{|\mathbf{r_i}'|}$$

This is just the motion of planet i about the center of mass of the lower indexed bodies. This is the physical intuition explaining why these Hamiltonians are separable; advancing the body 1 on its elliptical orbit doesn't change the center of mass seen by planets 2 through $N-1$. The flavor is similar to solving a linear algebra problem with an LU decomposition. The interaction term is further simplified to

$$\mathcal{H}_{\text{Interaction}} = \sum_{i=2}^{N-1} \frac{G m_i' M_i}{|\mathbf{r}_i'|} - \sum_{i=0}^{N-1} \sum_{\substack{j=i+1 \\ j \neq 1}}^{N-1} \frac{G m_1 m_2}{|\mathbf{r}_i - \mathbf{r}_j|}$$

The first term can be evaluated quickly in Jacobi coordinates, the second term quickly in Cartesian coordinates.

John Chambers gives a clear and intuitive explanation of the principles of Hamiltonian splitting in his paper describing the `MERCURY` integrator. Express the rate of change of any quantity of interest ϕ as

$$\begin{aligned} \frac{d\phi}{dt} &= \sum_{i=1}^{N} \left(\frac{\partial \phi}{\partial q_i} \frac{dq_i}{dt} + \frac{\partial \phi}{\partial p_i} \frac{dp_i}{dt} \right) \\ &= \sum_{i=1}^{N} \left(\frac{\partial \phi}{\partial q_i} \frac{\partial \mathcal{H}}{\partial p_i} - \frac{\partial \phi}{\partial p_i} \frac{\partial \mathcal{H}}{\partial q_i} \right) \end{aligned}$$

If we think of the Hamiltonian flow as an operator $F\phi$, the solution to the Hamiltonian system is

$$\phi(t) = e^{\tau F} \phi(t - \tau) = \left(1 + \tau F + \frac{\tau^2 F^2}{2} + \cdots \right)$$

where $\phi(t-\tau)$ is the value of ϕ at an earlier time. Now suppose $\mathcal{H} = \mathcal{H}_A + \mathcal{H}_B$ and define operators A and B for the Hamiltonian flows induced by $\mathcal{H}_A$ and $\mathcal{H}_B$. The operators A and B are linear and associative, but in general they do not commute, i.e. $AB \neq BA$. The exponential for the split Hamiltonian now takes the following form:

$$e^{\tau(A+B)} = 1 + \tau(A+B) + \frac{\tau^2}{2}\left(A^2 + AB + BA + B^2\right) + \cdots$$

Suppose instead we apply the operators A and B *sequentially* rather simultaneously. We will make a small error if A and B don't commute:

$$e^{\tau A}e^{\tau B} = \left(1 + \tau A + \frac{\tau^2 A^2}{2} + \cdots\right)\left(1 + \tau B + \frac{\tau^2 B^2}{2} + \cdots\right) = 1 + \tau(A+B) + \frac{\tau^2}{2}\left(A^2 + 2AB + B^2\right) + \cdots$$

Comparing these two expressions, we can see that they match to first order in τ, with an error that is second order in τ and proportional to the commutator bracket $[A, B] = AB - BA$. A second order scheme with step size τ is thus described in operator notation by

$$\phi(t) = e^{\tau B/2}e^{\tau A}e^{\tau B/2}\phi(t-\tau)$$

Chambers also shows with a simple argument that as long as $\mathcal{H}_A \gg \mathcal{H}_B$, the integrator will have an error that scales as $\epsilon\tau^3$ where ϵ is the ratio of the mass of the planets over the mass of the sum, which is 0.014 for the solar system.

This is why the Wisdom-Holman integrator works so well-except when there are close encounters. When an object such as an asteroid approaches very close to a planet, the assumption that the Keplerian Hamiltonian dominates the interaction Hamiltonian breaks down. The moon for instance is dominated by the earth's gravity, not the sun's. Chambers created a popular hybrid integrator named `MERCURY` with a simple idea. It integrates time steps using WH, but checks for close encounters. When a close encounter is detected, the bodies undergoing the encounter are integrated numerically for the next time step using a Bulirsch-Stoer scheme.

Here is the scheme used for the `WHFast` numerical integrator based on the Hamiltonian split between the Kepler, Interaction, and center of mass terms:

- Drift - Evolve the system under $\mathcal{H}_{\text{Kepler}}(dt/2) \circ \mathcal{H}_0(dt/2)$
- Kick - Evolve the system under $\mathcal{H}_{\text{Interaction}}(dt)$
- Drift - Evolve the system under $\mathcal{H}_{\text{Kepler}}(dt/2) \circ \mathcal{H}_0(dt/2)$

To put this in words, in the first step each planet evolves according to the classical Kepler 2-body problem about the center of mass of the bodies closer to the center than itself. It also follows an inertial drift. In the second step, the planets gently tug on each other. Only their velocities are affected. The third step is the same as the first.

An important optimization is possible. The first and third step are the same, so the third step of one epoch can be consolidated with the first step of the following epoch, running a single Kepler step of length dt. Only the first and last stage before an output is generated require the half step. Because the Kepler step is the most expensive, this can speed up a code by nearly a factor of 2. The orbital problem has now been reduced to efficiently taking time steps with the simpler Kepler and Interaction Hamiltonians. The interaction Hamiltonian is very straightforward. The next section describes how to compute an arbitrary Kepler step.

2.4.4 Numerical Solution of the Kepler 2-Body Problem

The two body Kepler problem was known to produce elliptical orbits even before the invention of calculus. Indeed it was Kepler who studied this problem and published his famous three laws of planetary motion, the first of which states that the planets move in elliptical orbits with the sun at one focus. Assuming elliptical orbits however is not the best approach for numerical work, and can break down if bodies are ejected from the system and switch to hyperbolic orbits.

A better approach is the so-called universal variable formulation dating to Gauss and his famous f and g functions. This has an "almost analytical" form, is robust to different orbital shapes, and can be computed efficiently. By almost analytical, I mean that there is one crucial step that must be performed numerically,

after which the position and velocity can be expressed using the f and g functions. Here is a brief explanation of this approach, following the two Wikipedia articles. [11]

The universal variable formulation starts from the differential equation of motion in an attractive potential

$$\frac{d^2\mathbf{r}}{dt^2} + \mu\frac{\mathbf{r}}{r^3} = 0$$

where $r = r(t)$ is the scalar distance to the center. An auxiliary variable s is introduced that satisfies the differential equation

$$\frac{ds}{dt} = \frac{1}{r}$$

We can see that s is an angle-like quantity, in the sense that each revolution increases it by the same amount. For a circular orbit, s will be a multiple of the standard angle θ. In this sense, s is a generalization of angles that extends to ellipses, parabolas and hyperbolas. Applying the changes of variables to s in place of t yields a new form of the differential equation

$$\frac{d^2\mathbf{r}}{ds^s} + \alpha\mathbf{r} = -\mathbf{P}$$

where $\alpha = \mu/a$ and $\mathbf{P}$ is a constant vector. The equation now takes the form of a harmonic oscillator, one of the most well studied problems in mathematical physics. Differentiating one more time, the constant $\mathbf{P}$ drops out again and we have

$$\frac{d^3\mathbf{r}}{ds^3} + \alpha\frac{d\mathbf{r}}{ds} = 0$$

While this might look scary, it turns out this equation has closed form solutions in terms of a family of functions $c_k(x)$ called Stumpff functions, which generalize sine and cosine. The Stumpff functions are defined by

$$c_k(x) = \frac{1}{k!} - \frac{x}{(k+2)!} + \frac{x^4}{k+4!} - \cdots = \sum_{i=0}^{\infty}\frac{(-1)^i x^i}{(k+2i)!}$$

These functions converge absolutely for real valued x. By comparing their expansions to those of $\sin(x)$ and $\cos(x)$, the first two Stumpff functions can be evaluated in closed form:

$$c_0(x) = \begin{cases} \cos(\sqrt{x}) & \text{for } x \geq 0 \\ \cosh(\sqrt{-x}) & \text{for } x < 0 \end{cases}$$

$$c_1(x) = \begin{cases} \frac{\sin(\sqrt{x})}{\sqrt{x}} & \text{for } x \geq 0 \\ \frac{\sinh(\sqrt{-x})}{\sqrt{-x}} & \text{for } x < 0 \end{cases}$$

$c_k(x)$ for higher k can be evaluated using the recurrence relation

$$xc_{k+2}(x) = \frac{1}{k!} - c_k(x)$$

The solution to the above differential equation is a linear combination of the three primitive solutions 1, $sc_1(\alpha s^2)$, and $c_2(\alpha s^2)$. Note the analogy to a standard harmonic oscillator, whose solutions are a linear combination of sine and cosine. The solution to this differential equation for one time step then takes the form

$$t - t_0 = r_0 sc_1(\alpha s^2) + r_0\frac{dr_0}{dt}s^2c_2(\alpha s^2) + \mu s^3 c_3(\alpha s^2)$$

This is called the universal variable formulation of Kepler's equations. Here t_0 and r_0 are the time and

[11] Apparently this is considered sufficiently basic that none of the five articles I read reviewed it. Because I did not see this exact derivation in one semester of college physics plus a one semester course specializing in advanced classical mechanics (!) I thought it worth presenting here.

distance to the center at the start of a time step. This equation can be efficiently solved numerically using Newton's method. We can now see the motivation behind the definition of Gauss's f and g functions:

$$f(s) = 1 - \left(\frac{\mu}{r_0}\right) s^2 c_2(\alpha s^2) \qquad \dot{f}(s) = -\left(\frac{\mu}{r r_0}\right) s c_1(\alpha s^2)$$
$$g(s) = t - t_0 - \mu s^3 c_3(\alpha s^2) \qquad \dot{g}(s) = 1 - \left(\frac{\mu}{r}\right) s^2 c_2(\alpha s^2)$$

With these definitions, the position $\mathbf{r}(t)$ and velocity $\mathbf{v}(t)$ are given by

$$\mathbf{r}(t) = \mathbf{r}_0 f(s) + \mathbf{v}_0 g(s)$$
$$\mathbf{v}(t) = \mathbf{r}_0 \dot{f}(s) + \mathbf{v}_0 \dot{g}(s)$$

There's a lot happening here, so it's worth taking a moment to recapitulate. We can transform the Kepler two body problem into universal coordinates. The solution in general is sum over Stumpff functions $c_k(x)$ that generalize trigonometric functions. This leads to an implicit equation for a time step from t_0 to t in terms of the "generalized angle" s. The equation can be solved for s at the end of the time step efficiently using Newton's method. Once s is known, we can evaluate the Gauss f and g functions and their derivatives. These allow us to compute the new position and velocity at the end of the time step.

The critical benefit of this procedure is that because the Stumpff functions are periodic and numerically well behaved, we can take much longer time steps with this method than would otherwise be possible. This allows us to integrate a system without being forced to take a tiny step size to accommodate the shortest orbital period. In the solar system for example, Mercury has a period of 88 days, forcing time steps ~ 4 days or less that are too short a long term integration. We are now in a position to use an implementation of the Wisdom-Holman integrator such as `WHFast` with a full understanding of all the key mathematical steps underpinning it.

2.5 The IAS15 Adaptive Integrator

The `IAS15` integrator, presented in a 2014 paper by Rein and Spiegel, is a an impressive achievement. It a fast, adaptive, 15th order integrator for the N-body problem that is (amazingly!) accurate to machine precision over a billion orbits. The explanation is remarkably simple in comparison to what this algorithm can do. Rein and Spiegel start by writing the equation of motion in the form

$$y'' = F[y', y, t]$$

Here y is the position of a particle; y' and y'' are its velocity and acceleration; and F is a function with the force acting on it over its mass. In the case of gravitational forces, the only dependence of F is on y; but one of the major advantages of this framework is its flexibility to support other forces, including non-conservative forces that may depend on velocity. Two practical examples are drag forces and radiation pressure.

This expression for y'' is expanded to 7th order in t,

$$y''[t] \approx y_0'' + a_0 t + a_1 t^2 + \cdots + a_6 t^7$$

They next change variables to dimensionless units $h = t/dt$ and coefficients $b_k = a_k dt^{k+1}$:

$$y''[t] \approx y_0'' + b_0 h + b_1 h^2 + \cdots + b_6 h^7$$

The coefficients h_i represent relative sample points in the interval $[0, 1]$ that subdivide a time step. Rein and Spiegel call them substeps. The formula is rearranged in terms of new coefficients g_k with the property that

g_k depends only on force evaluations at substeps h_i for $i \leq k$.

$$y''[t] \approx y_0'' + g_1 h + g_2 h(h-h_1) + g_3 h(h-h_1)(h-h_2) + \cdots + g_8 h(h-h_1)\cdots(h-h_7)$$

Taking the first two g_i as examples and using the notation $y_n'' = y''[h_n]$,

$$g_1 = \frac{y_1'' - y_0''}{h_1} \qquad g_2 = \frac{y_2'' - y_0'' - g_1 h_2}{h_2(h_2 - h_1)}$$

This idea has a similar feeling to the Jacobi coordinates: a change of coordinates with a dependency structure to allow sequential computations.

Using the b_k coefficients, it is possible to write polynomial expressions for $y'[h]$ and $y''[h]$:

$$y'[h] \approx y_0' + hdt\left(y_0'' + \frac{h}{2}\left(b_0 + \frac{2h}{3}\left(b_1 + \cdots\right)\right)\right)$$
$$y[h] \approx y_0 + y_0' hdt + \frac{h^2 dt^2}{2}\left(y_0'' + \frac{h}{3}\left(b_0 + \frac{h}{2}\left(b_1 + \cdots\right)\right)\right)$$

The next idea is to use Gauss-Radau quadrature to approximate this integral with extremely high precision. Gauss-Radau quadrature is similar to standard Gauss quadrature for evaluating numerical integrals, but the first sample point is at the start of the integration window at $h = 0$. This is a strategic choice here because we already know y' and y'' at $h = 0$ from the previous time step. This setup now reduces calculation of a time step to finding good estimate of the coefficients b_k. Computing the b_k requires the forces during the time step at the sample points h_n, which in turn provide estimates for the g_k, and then feed back to a new estimate of b_k.

This is an implicit system that Rein and Spiegel solve efficiently using what they call a predictor-corrector scheme. At the cold start, they set all the $b_k = 0$, corresponding to constant acceleration over the time step. This leads to improved estimates of the forces at the substeps, and an improved estimates for the path on the step. This process is iterated until the positions and velocities have converged to machine precision. The first two time steps are solved from the cold start this way.

Afterwards, a much more efficient initial guess is made. They keep track of the change between the initial prediction of b_k and its value after convergence, calling this correction e_k. At each step, the initial guess is b_k at the last step plus e_k. An adaptive criterion is used to test whether the predictor-corrector loop has converged. The error is estimated as

$$\widetilde{\delta b_6} = \frac{\max_i |\delta b_{6,i}|}{\max_i |y_i''|}$$

The index i runs over all 3 components of each particle. The loop terminates when $\widetilde{\delta b_6} < \epsilon_{\delta b}$; they choose $\epsilon_{\delta b} = 10^{-16}$. It turns out that the b_k behave well enough for practical problems that this procedure will typically converge in just 2 iterations!

The stepsize is controlled adaptively with an analogous procedure. The tolerance is set with a dimensionless parameter ϵ_b, which they set to 10^{-9}. As long as the step size dt is "reasonable" in the sense that it can capture the physical phenomena in question, the error in y'' will be bounded by the last term evaluated at $h = 1$, i.e. the error will be bounded by b_6. The relative error in acceleration $\widetilde{b_6} = b_6/y''$ is estimated as

$$\widetilde{b_6} = \frac{\max_i |b_{6,i}|}{\max_i |y_i''|}$$

These are similar to the error bounds for convergence of the predictor-corrector loop, but involve the magnitude of b_6 rather than its change δb_6. An immediate corollary is that changing the time step by a factor f will change b_6 by a factor of f^7.

An integration step is computed with a trial step size dt_{trial}. At the end of the calculation, we compute

the error estimate $\widetilde{b_6}$. If it is below the error tolerance ϵ_b, the time step is accepted. Otherwise, it is rejected and a new attempt is made with a smaller time step. Once a time step is accepted, the next time step is tuned adaptively according to $dt_{\rm required} = dt_{\rm trial} \cdot \left(\epsilon_b/\widetilde{b_6}\right)^{1/7}$ Please note that while the relative error in y'' may be of order 7, the use of a 15th order integrator implies that shrinking the time steps by a factor α will improve the error by a factor of α^{16}.

Hein and Spiegel include an interesting discussion of the different sources of error in N-body integrators. They explore the familiar errors arising from the numerical scheme, but also explore both random and biased numerical errors. They give a complete error decomposition

$$E_{\rm tot} = E_{\rm floor} + E_{\rm rand} + E_{\rm bias} + E_{\rm scheme}$$

$E_{\rm floor}$ is the baseline numerical error that is unavoidable when we try to represent real numbers to limited machine precision. $E_{\rm rand}$ is the familiar errors due to accumulated numerical round-off, provided they are distributed randomly (i.e. unbiased). $E_{\rm bias}$ is an accumulated effect of numerical round-off that has a bias.

This is a subtle point best illustrated by an example. Suppose you have a floating point representation of a 2x2 rotation matrix. For a given angle ϕ, the floating point sum of $\cos^2\phi + \sin^2\phi$ is likely not to be 1.0 to full machine precision. If it is even a tiny bit less than 1, then repeated application of this rotation matrix over many time steps will gradually lead to a contraction. Rein and Spiegel devote substantial effort to minimizing rounding errors, particularly by using compensated sums. This simple idea adds only slightly to the run time but can lead to significant accuracy improvements, especially over long term simulation, e.g. on the order of one billion orbital periods.

Rein and Spiegel test the phase accuracy of `IAS15` by running integrating the outer solar system forward in time for 50 orbits with a fixed time step, then backwards for 50 orbits with the same time step. The known answer is that the phases should be the same, allowing for sharp accuracy measurements. This test shows that `IAS15` is more accurate than the comparisons including WH at preserving phases.

They also introduce a criterion of optimality and demonstrate that `IAS15` integrator satisfies it. A result called Brouwer's Law status that in the presence of round-off, a cumulative sum will have have an error that scales as $n^{1/2}$, i.e. it is a random walk of n steps. Angular type variables including the orbital phase grow as $n^{3/2}$. The authors tested `IAS15` and other integrators on a long term integration of the outer solar system (Jupiter, Saturn, Uranus, Neptune) for one billion Jupiter orbits, approximately 12 billion years. The energy errors satisfied Brouwer's law, and grew more slowly than those made by the other schemes. One surprising conclusions is that the non-symplectic `IAS15` integrator is "more symplectic" (in the sense of having smaller energy errors) than the symplectic integrators! This is a completely non-obvious results, and shows the importance of analyzing and understanding all the sources of error in a calculation.

3 Numerical Experiments on Three Integrators

3.1 Simulating the Solar System for 100,000 Years

I started out by simulating the solar system for 100,000 years using three integrators: leapfrog, WH, and IAS15. One of the real pleasures of using `REBOUND` is that it comes with a convenient Python interface to the core C library's functionality. In this regard it follows popular tools like `numpy` and `tensorflow` that combine the flexibility and ease of use of Python with the speed of C for the performance intensive parts of the code.

The Python program `sim_planets.py` performs these simulations. In the past when I've worked on this problem, one of the most painful parts has been getting accurate initial conditions for the configuration of the planets (positions and velocities in an astronomical reference frame). The Python API of `REBOUND` makes this a breeze by including an interface to the NASA Horizons service. For the uninitiated, this is an extremely convenient service offered free to the public that provides ephemerides (orbital elements) for just about everything interesting in the solar system. Cataloged objects include the sun, 8 planets, 190 planetary satellites (a.k.a. moons), 3573 comets and a whopping 795,071 asteroids (and counting!). The following code snippet sets up a simulation in `REBOUND` with all 8 planets as of January 1, 2000 12:00 (the epoch of the current J2000.0 celestial coordinate system) [12]

```
# Import the library; it can be installed with anaconda or pip
import rebound
# Create an empty simulation
sim = rebound.Simulation()
# Set units to years, astronomical units and solar masses
sim.units = ('yr', 'AU', 'Msun')
# List of object names for NASA Horizons: sun and 8 planets
object_names = ['Sun', 'Mercury', 'Venus', 'Earth', 'Mars',
                'Jupiter', 'Saturn', 'Uranus', 'Neptune']
# Set the date time with a string
horizon_date = '2000-01-01 12:00'
# Create the simulation with planetary data from Horizons
sim.add(object_names, date=horizon_date)
```

To simulate this system for 100,000 years using the default IAS15 integrator with its default tolerance $\epsilon_b = 10^{-9}$ is just one line of code: `sim.integrate(100000.0)`. Rebound makes it convenient to save simulation archives, which allow the simulation to be restarted from any saved checkpoint. I set the 100,000 year simulations to save checkpoints every 1 year. The size of an archive varies with the integrator; the more parameters it has, the bigger the file. For the archives I ran with 100,000 snapshots, the leapfrog integrator used about 123 MB; the WH integrator used 242 MB; and the IAS15 integrator used 1186 MB.

The snapshots in a simulation archive are not saved at exact multiples of the specified time, merely close to them. If you want to compare results from two simulations saved as archives, you **should not** compare them element by element, because they will have slightly different snap times. This was the source of a bug that took me a while to run down; the errors were coming in too high because I was comparing unsynchronized snap times. To get the system state at the exact specified time, you need to use the method
`sim = sa.getSimulation(t, mode='exact')`

The file consists mostly of functions to do bookkeeping of simulation archives and keeping track of how they were generated.
The function `make_sim_planets` creates a new simulation with the positions of the sun and planets with the syntax that queries Horizons.

[12] See celestial coordinates for a discussion of celestial coordinate frames.

This is only done once; afterwards, the file is loaded from disk with the function `load_or_make_start`.
The function `make_archive` generates an archive from a configured simulation. The functions `batch_dt` and `batch_tol` run a batch of simulations for integrators that specify either a time step (e.g. leapfrog) or a tolerance (e.g. IAS15).
The function `make_sa_tbl_dt` creates a Python dictionary keyed by (integrator_name, dt) where the time step dt is an integer number of days. The dictionary stores a simulation archive for this run.
`make_sa_tbl_tol` is the analogous function for adaptive simulation runs. This time the second part of the key is a field called `ptol`, which is defined by $\epsilon = 2^{-ptol}$.
The function `sa2xyz_all` extracts the Cartesian coordinates x, y and z from a simulation archive as numpy arrays at all the snapshots.
The function `sa2xyz_sched` is similar, but it outputs the coordinates at exact times on the specified schedule. Outputs of this function from different simulation archives can safely be compared to each other as long as the schedules are the same.
The function `make_xyz_tbl_sched` generate a Python dictionaries keyed by (integrator_name, key2) where key2 is either dt or ptol.
The function `sa2orbit_sched` is similar to `sa2xyz_sched`, except it outputs orbital elements.

The main routine performs the following simulations of the solar system for 100,000 years and saves the outputs to simulation archive files. For the two integrators with fixed time steps, `leapfrog` and `whfast`, the time steps are run with dt = 1, 2, 4, 8, 16, 32, 64, 128, and 256 days. The adaptive `ias15` integrator is run with tolerances of 2^{ptol} for `ptol` in 4, 8, 12, 16, 20, 24, 28, and 32. (The default setting of $\epsilon = 10^{-9}$ corresponds to a ptol of about 30).

The programs `plot_orbits.py` and `plot_errors.py` generate visualizations of orbits and error analysis for the different integrators considered. Because this is quite straightforward I will not comment further on it.

3.2 Visualization of the Solar System Simulation

There is a tremendous amount of information in a solar system simulation and the best way to convey it visually is not obvious to me. I will begin with a plot of all the points occupied by the four inner planets over the first 100 years of the simulation (2000-2100). I choose the inner planets because they make for a chart that isn't overwhelmed with empty space and because they are harder to simulate accurately. These are the results for the IAS15 integrator with the tightest tolerance of 2^{-32}, which I treat as the "gold standard" for error analysis. We can see that all four planets retain their elliptical orbits very precisely.

The next plot is a montage of 6 versions of the same thing, but for the leapfrog integrator with dt at 1, 2, 4, 8, 16, and 32 days. We can see some interesting behavior here. Even the smallest time step of 1 day is unable to accurately model the orbit of Mercury for 100 years. Mercury has an orbital period of approximately 88 days, so a time step of 1 day is approximately 0.011 of an orbital period. That's a physically sensible choice, but it's not sharp enough to maintain accuracy over 100 years. As the time step increases, the other three planets get increasingly thick lines, indicating departures from the original elliptical shape. Mars in particular gets quite spread out with a time step of 32 days. If you double the time step one more time to 64 days, the leapfrog integrator fails bigly: it ejects Mercury from the Solar System! It is also blurring the orbits of the other three planets so much that earth and Venus are drifting together, with Mars not too far behind. [13]

The comparison between the leapfrog and WH is stark. Even though WH is also a fairly cheap and quick integrator, it is vastly superior to the leapfrog. With the longest time step of 32 days, it is still very accurate, even for Mercury.

What about the next 100,000 years? According to `ias15`, Venus and Earth will barely budge, but

[13] Please see urban dictionary if you are unfamiliar with the word *bigly*.

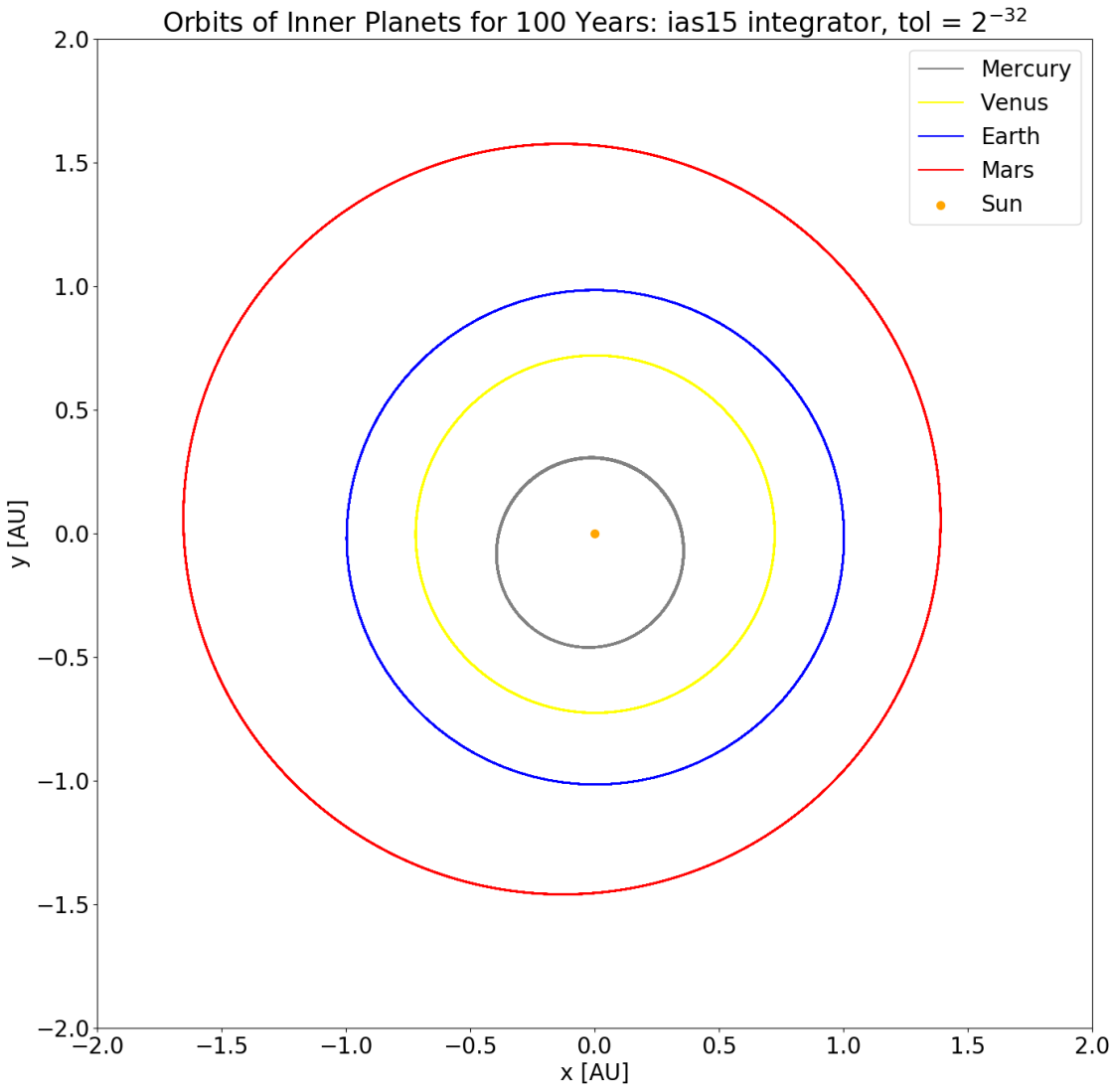


Figure 2: Orbits of the inner planets for 100 years with the IAS 15 integrator.
The elliptical orbits are maintained with perfect fidelity.

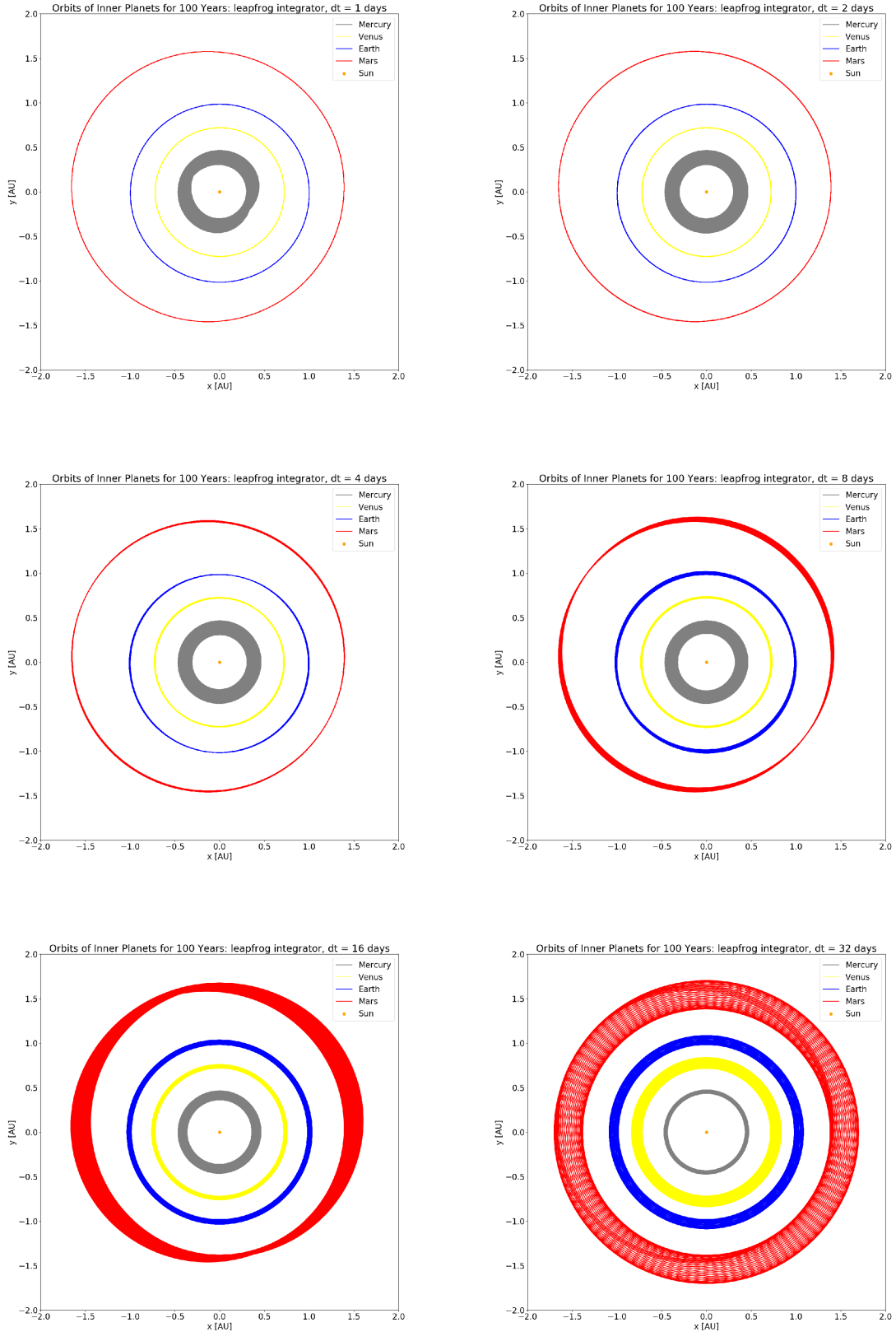


Figure 3: Orbits of the inner planets for 100 years with the leapfrog integrator. The time step ranges over 1, 2, 4, 8, 16, 32 days. Progressively larger numerical errors cause the reported orbits to spread out.

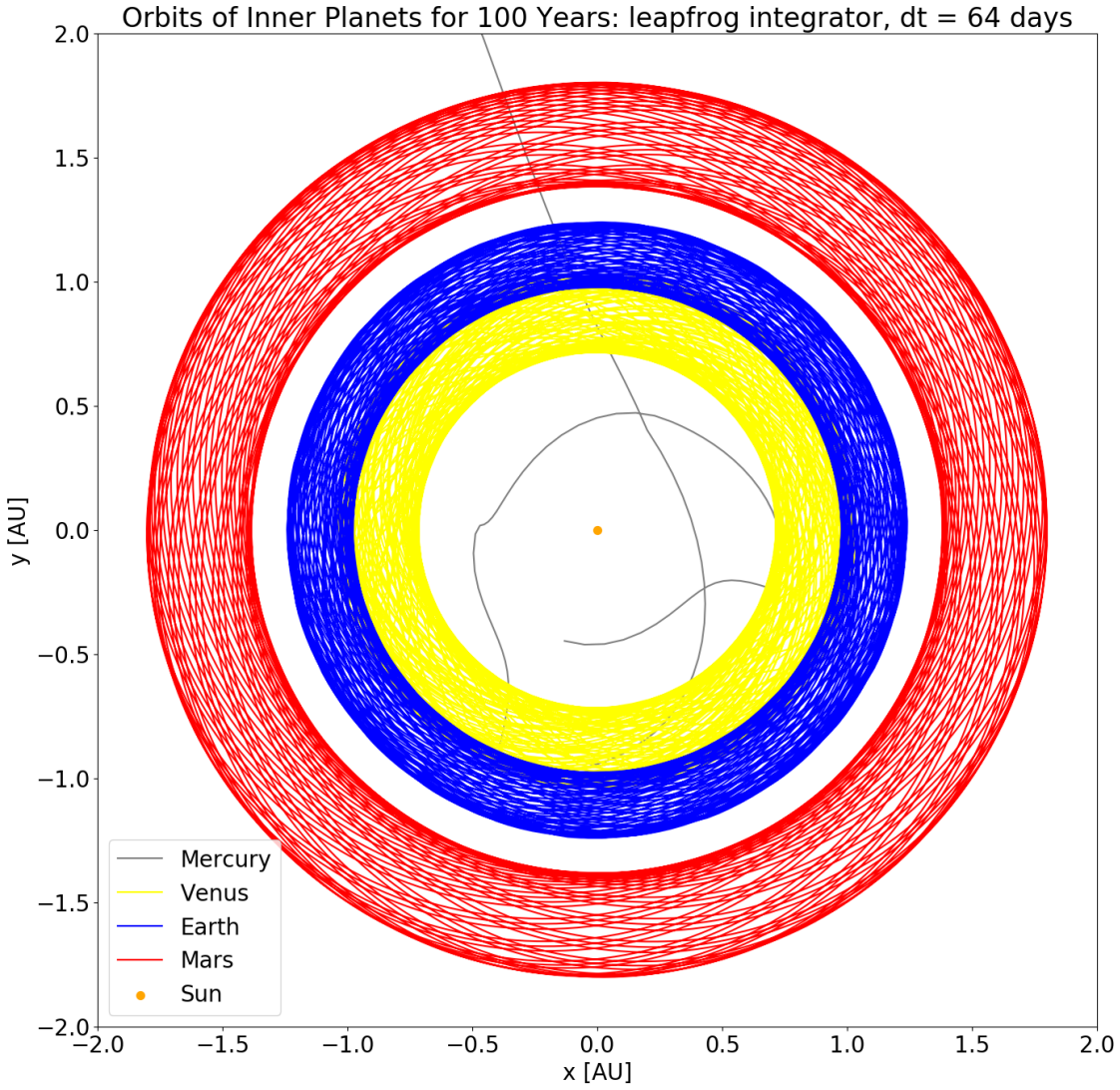


Figure 4: The leapfrog integrator with a time step of 64 days fails bigly, spuriously reporting the ejection of Mercury from the solar system.

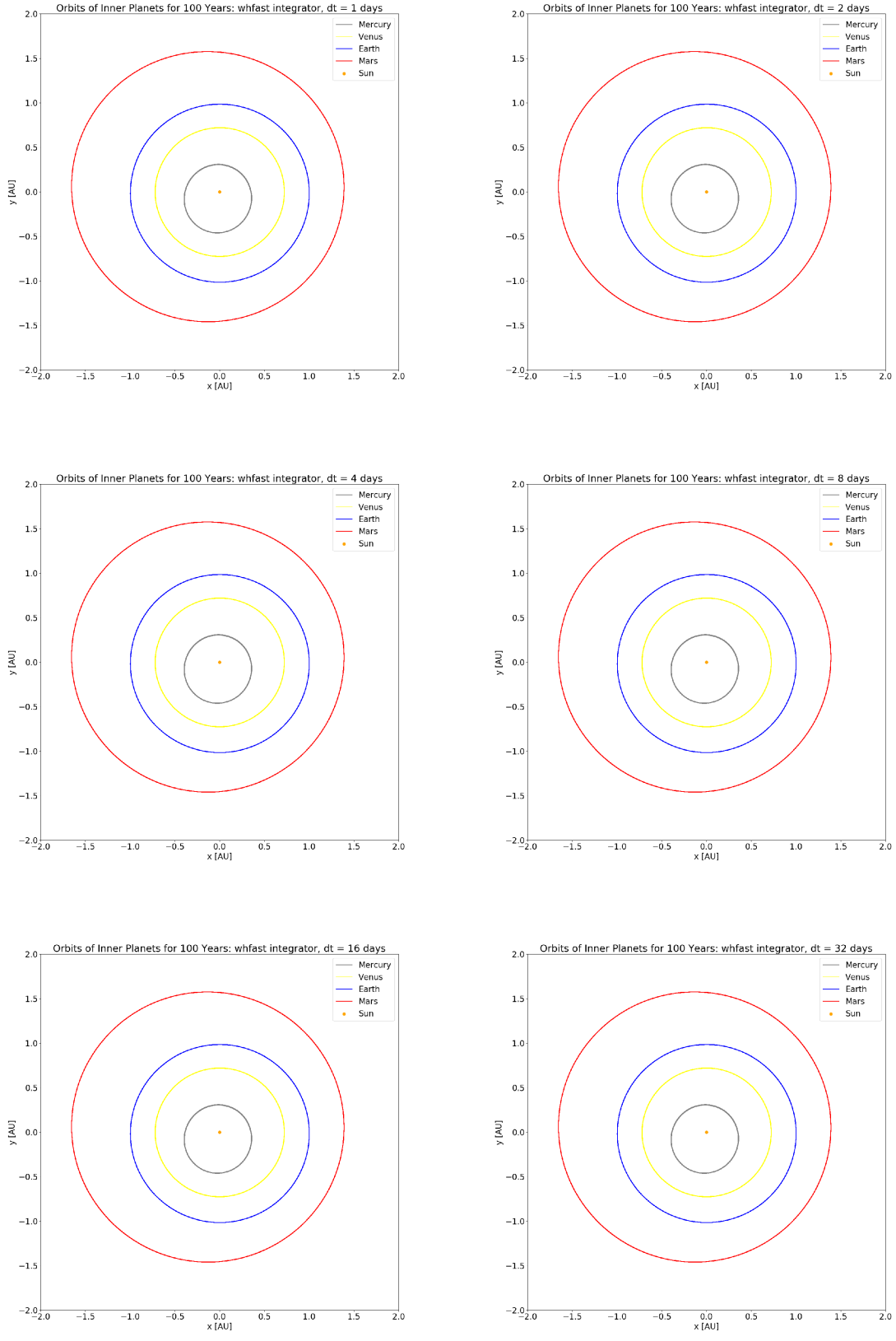


Figure 5: Orbits of the inner planets for 100 years with the WH integrator. The time step ranges over 1, 2, 4, 8, 16, 32 days. Unlike leapfrog, WH has no trouble handling a 32 day step size.

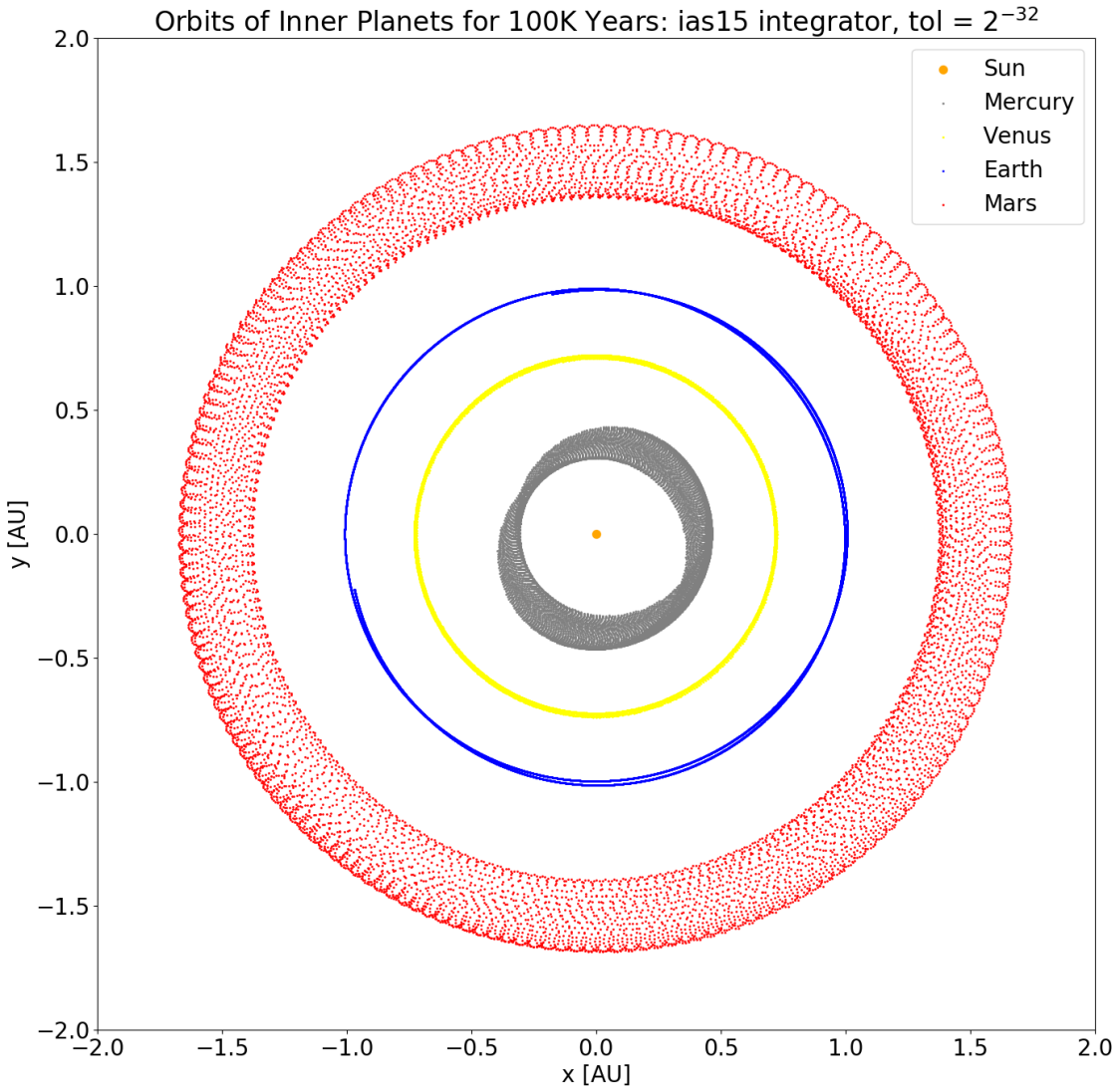


Figure 6: Venus and Earth hardly budge, but Mercury and Mars drift a fair amount, while maintaining stable orbits.

Mercury and Mars will both wander around a fair amount. Nevertheless all four planets maintain stable orbits. The WH integrator is visually indistinguishable at all time steps considered. The leapfrog integrator blurs the orbits as dt increases, and eventually ejects Venus as well with $dt = 64$.

3.3 Comparing the Accuracy of These Integrators

In this section we will review the accuracy of these integrators. We will consider accuracy at the same two scales, 100 and 100,000 years. 100 years is intended to be at the longer end of simulations where we desire a high degree of accuracy. NASA for example does its asteroid simulations out 100 years. 100,000 years was chosen somewhat arbitrarily to be long enough to show large time scale dynamics, but short enough not to require undue amounts of computer time for the most expensive IAS15 integrator.

All error analysis below uses the `ias15` integration at the tightest tolerance 2^{-32} as the assumed correct answer. I believe this is a reasonable conclusion for this problem in light of Rein's papers and the results we will see below. Errors for the IAS integrator with this setting are not reported as zero. Instead, a simple

extrapolation is performed using IAS15 with the two closest tolerances, 2^{-28} and 2^{-24}. The linear fit on the log-log plot of error vs. log step size is good enough that this is a reasonable model.

An alternate approach to error estimation would be to compare the outputs of the model to another system, e.g. Horizons, that is assumed to be correct. I have done this on an ad-hoc basis enough to have confidence that my results are broadly correct. It is very difficult to make a tight numerical error estimate when comparing different systems though because what you are trying to measure will quickly get swamped by differences in assumptions or initial configurations. That is why papers like Rein's do cross-platform comparisons for problems with mathematically known answers such as the energy change and the phase change of a round trip with opposite signs on the time step, both of which should be zero. I have deliberately chosen not to take this approach to error estimation, because it doesn't answer the bottom line question I have: how close do these simulators got to the right answers for the positions of the planets? I already know that the energy errors of all the integators considered will be miniscule, and while the phase error on a round trip is probably quite informative, it's not the same thing as knowing how many AUs the integrator drifts off course over time.

For the first visualization, I plot the error in the individual planets for the next 100 years. I show these on a log-log scale because that is most effective at conveying the overall drift to higher errors. There are some sinusoidal patterns whose frequencies are more visible in the linear scale. These are interesting plots with a lot going on. The first key point is that the scales are very different. The bottom plot is scaled from 2^{-60} to 2^{-38} (the units are A.U., so these errors are close to relative errors). Typical errors are in the range of 2^{-42} AU. To put these scales in context, the radius of the earth in AUs is about $2^{-14.5}$. The IAS15 integrator is tremendously accurate on this time scale, easily accurate enough to resolve whether bodies will hit the earth. It doesn't have an obvious bias where one planet does worse than others.

The leapfrog integrator on the other hand has a scale from 2^{-32} to 2^{0}. By the end of the simulation, even with a time step of 1 day, leapfrog is making material errors. We can easily see the overall slope with all the errors moving up more or less in parallel. Leapfrog has a clear problem with the inner planets: the closer a planet is to the sun, the worse it does. This is not surprising, because the leapfrog has to work hard to model an elliptic orbit a series of parabolic curves. There is also a clear sinusoidal pattern to the leapfrog errors. Typical errors for leapfrog are in the range of 2^{-8} AU.

The WH integrator stacks up in between leapfrog and IAS15 for accuracy, with typical errors around 2^{-25} AU. WH also has a clear order of which planets it struggles with, but it's not the same as leapfrog. Looking at the chart the WH integrator has the toughest time with Earth, followed by Venus and Mars. The pattern of sinusoidal wobbles is similar to those for leapfrog, suggesting that it is some kind of resonant effect.

For our next visualization, we will compare the median errors made by the three integrators over the next 100,000 years. We can see the same order of accuracy. It's also clear that the IAS is losing accuracy faster than the other two.

For our next study we can compare the accuracy of methods with a fixed time step to the length of the time step. The accuracy of the `ias15` integrator seems to have a much less persistent impact on its performance, until it falls off a cliff when the tolerance is set to 2^{-4}.

Of course the fairest comparison between algorithms is wall time and accuracy. On this metric, WH still looks very strong. It is a much quicker way to get moderately high accuracy, which is probably more than good enough for most problem. If the highest accuracy is required it is hard to beat IAS15.

There is a bit of a disconnect here between what I was expecting after reading the Rein-Spiegel paper on IAS15, which was that IAS15 would have a clear superiority to WH in the sense that it could do the calculation faster with sensible tolerances. That is not the experience I had. I think a key difference is that Rein and Spiegel simulated only the *outer* solar system, starting with Jupiter, whereas I simulated the full solar system. One intuition I have is that `IAS15` is almost like leapfrog on steroids. It's very good at doing brute force calculations, but knows nothing about elliptical orbits, and needs to stitch them together with parabolic shapes over small enough time steps. For planets starting at Jupiter with a 12 year period, this

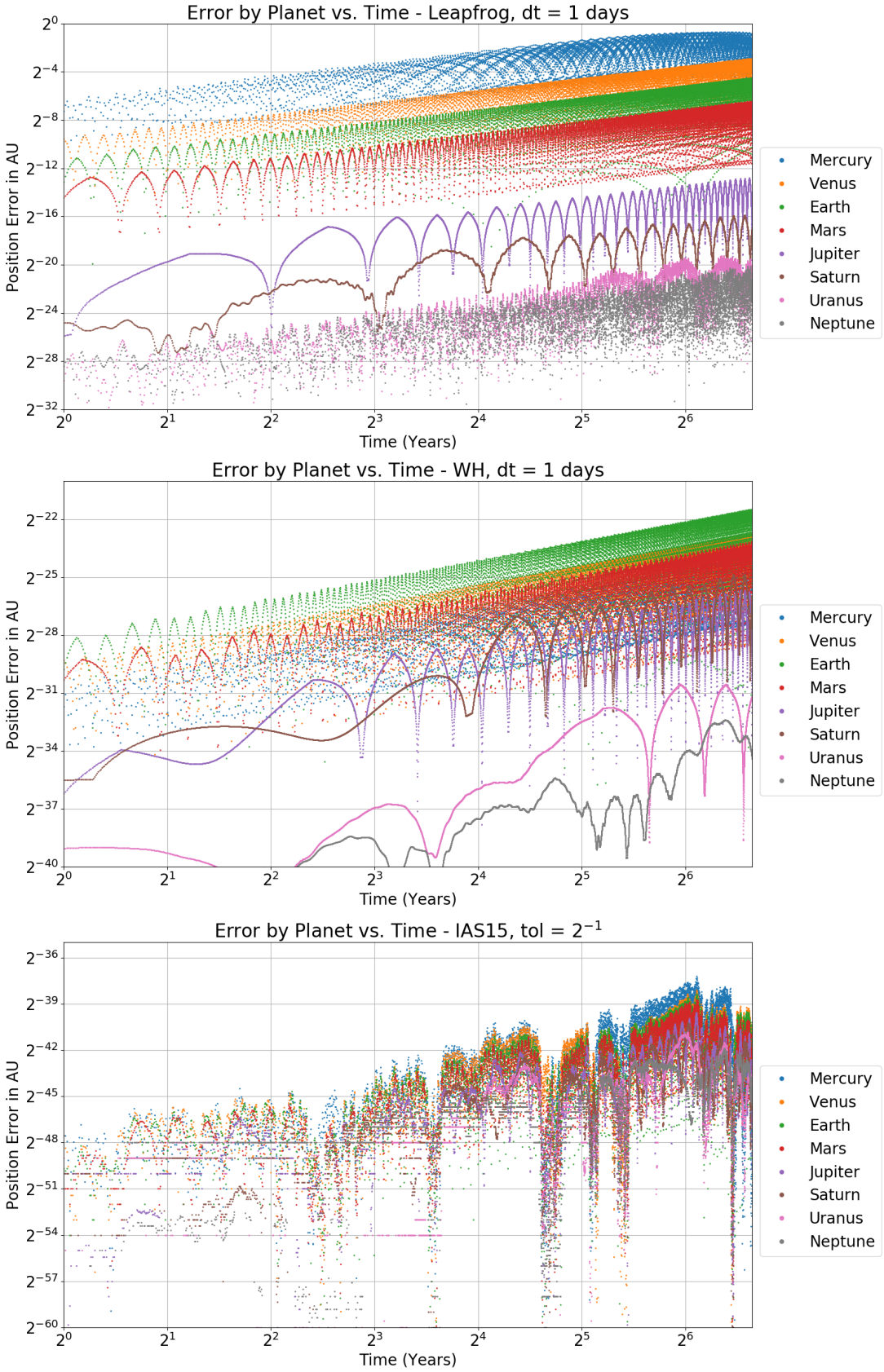


Figure 7: Errors by planet over the next 100 years for leapfrog, WH, and IAS15.

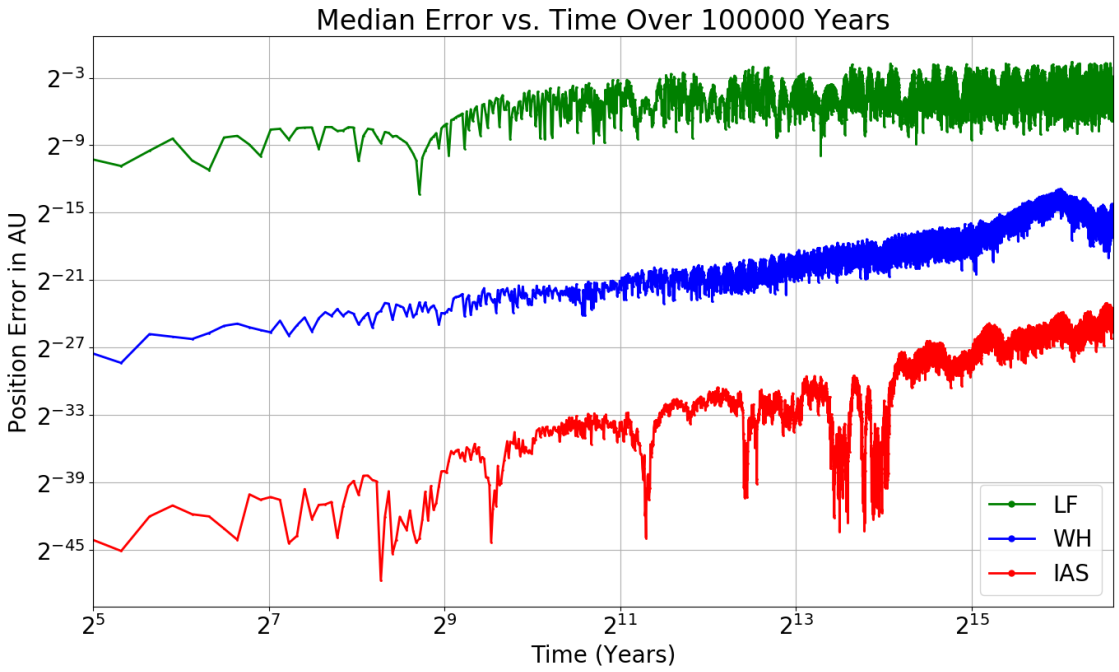


Figure 8: Errors vs. time for leapfrog, WH, and IAS15 over the next 100,000 years.
IAS is the most accurate but is also losing ground faster.

isn't much of a handicap, but Mercury and its 88 day period are not flattering for `IAS15`.

Doing this exercise has given me a new appreciation for why John Chambers named his integrator `MERCURY`! If I had to do this project over again, I might have also put the `Mercurius` hybrid integrator into the mix. This is the `REBOUND` version of the Chambers Mercury integrator that uses `whfast` on most time steps and then switches over to `ias15` when a near approach is detected.

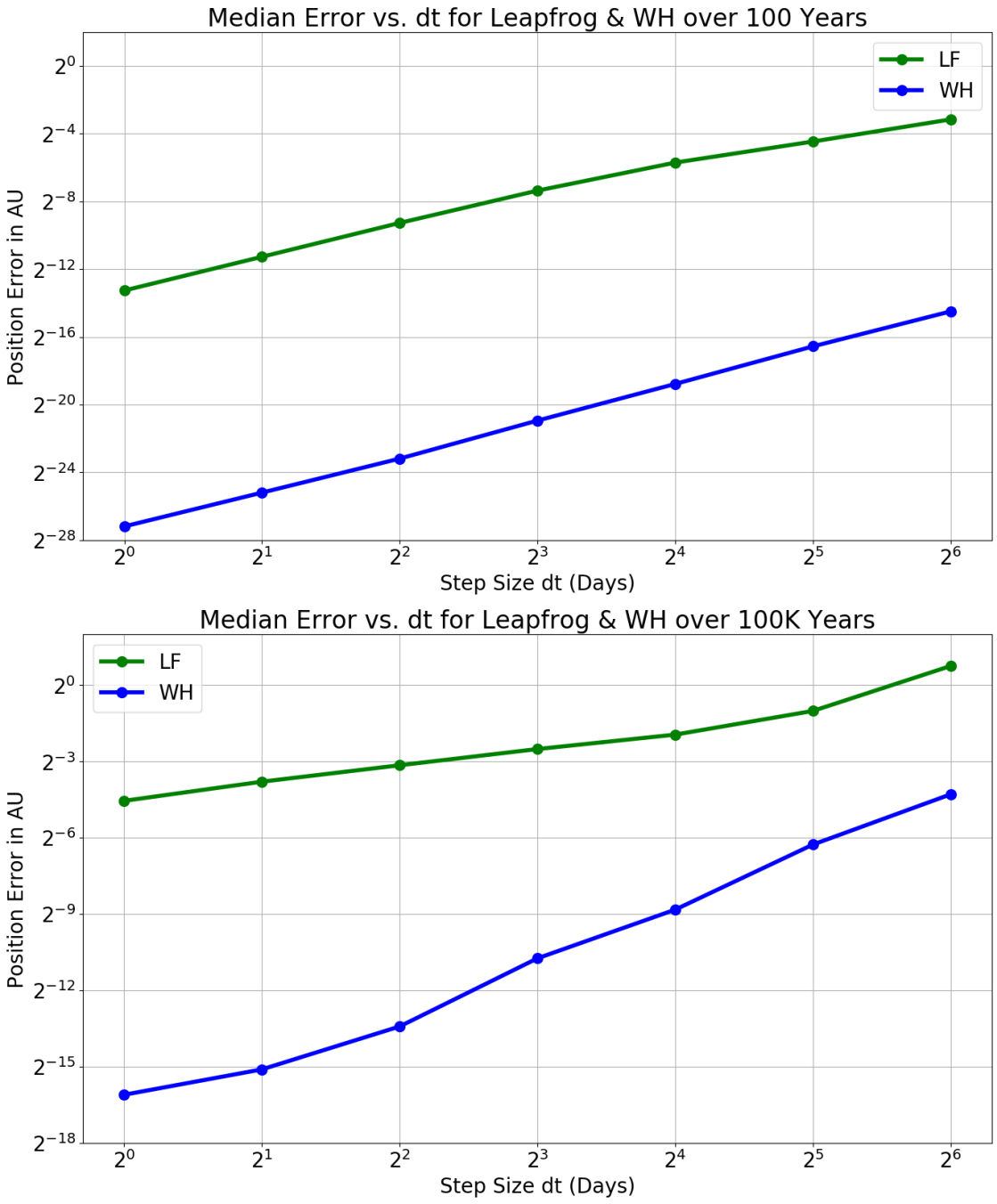


Figure 9: Accuracy of leapfrog and WH vs. the timestep for 100 and 100,000 year time scales.
WH has a consistent advantage of about 2^{13} for the first 100 years.
Runtimes are roughly comparable for a given time step.

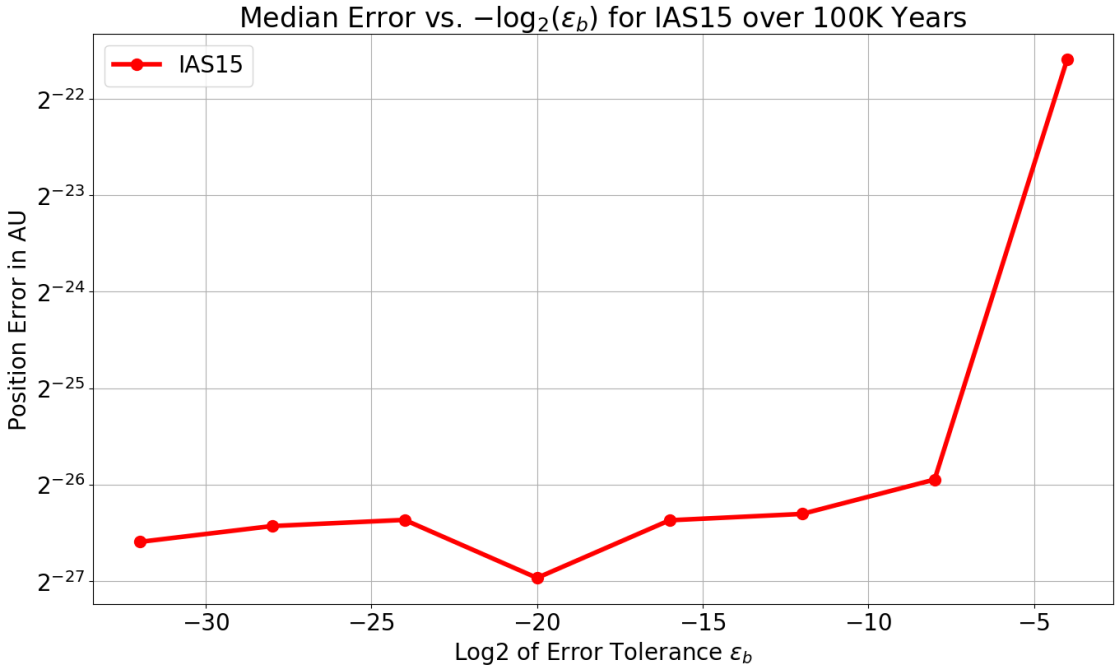


Figure 10: Accuracy of IAS15 vs. the tolerance for 100,000 year time scale.
Accuracy is steady until it degrades with $\epsilon = 2^{-4}$

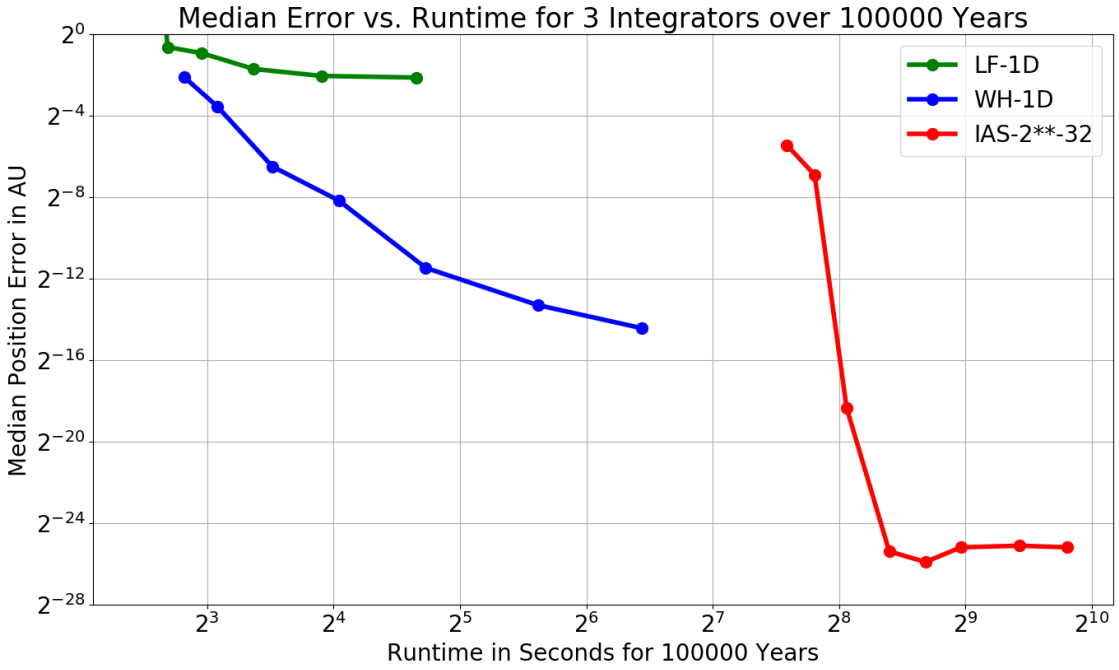


Figure 11: Accuracy vs. runtime for all 3 integrators on 100,000 year period.

4 Simulating the Near Approach of Asteroid Apophis to Earth

The asteroid Apophis was discovered in 2004. I have chosen to model it for this project because it achieved the highest ever rating on the Torino Scale, which is used to communicate impact hazard of near-earth objects (NEOs) to the public. [14] Apophis was assigned a rating of 4 on the Torino scale in December 2004, when the estimated risk of a strike peaked at 2.7% (approximately 1 in 37). Apophis was nominally named after the Egyptian god Apep, a.k.a Apophis, the god of chaos and darkness and opponent of the sun god Ra. However it was "really" named after a character on the Sci-Fi TV show Stargate SG-1, which includes in its fan base the astonomers who discovered it and thus got the privilege of naming it under IAU rules.

The NASA Sentry system estimates that if Apophis were to strike the earth, it would release approximately 750 megatons of energy. [15] To put this in context, the largest ever hydrogen bomb tested released 57 megatons, while the eruption of Krakatoa released 200 megatons. A strike by Apophis would unleash regional devastation in an area on the order of perhaps thousands of square miles, but would not pose a major hazard to the entire planet. This also gets to the point of planetary defense and close monitoring of asteroids. Perhaps you may share a healthy skepticism that humanity could replicate the plot of Armageddon and successfully deflect an object with a mass of $6.1 \cdot 10^{10}$ kg. Given that the U.S. still doesn't have a working ballistic missile defense system, this skepticism would be well placed. A 10 year plan to evacuate a radius of say 50 miles around a projected impact site, on the other hand, would be an eminently feasible undertaking. [16]

4.1 Simulating the Current Trajectory of Apophis

The calculations of Apophis's near earth approach are performed in the Python program `sim_apophis.py` and the visualizations are generated in `plot_apophis.py`. The function `make_sim_apophis` sets up the initial configuration. There are a few subtleties here. When a planet is queried by name, Horizons defaults to returning the barycenter (center of mass) of the entire planetary system, consisting of the planet and its moons. This is usually what you want. In this case however we want to distinguish between the Earth and the Moon, so we load them separately. A convenient way to do this is to query the Horizons web interface and click on the box next to "Target Body." A quick search gives the IDs for the Earth, Moon and Apophis.

[14] The Palermo Scale is a logarithmic risk scale that is better suited to scientific work. It compares the modeled probability of an asteroid strike over the simulation period to the estimated probability of a strike by an object of the same or greater energy from a background distribution of asteroid strikes.

[15] A megaton is the energy released by one million tons of TNT. Though hardly an SI unit, it has been the customary unit for communicating the destructive energy of nuclear weapons, which is why it has been adopted for communicating asteroid hazards.

[16] Here is an interesting and not very fun thought experiment. If NASA calculations showed that Cambridge, MA would be the epicenter an Apophis strike in 2029, would you sell your house and move? Would SEAS relocate not just to Allston but to, say, New York? I would probably want to run the simulation myself at least once before listing my house!

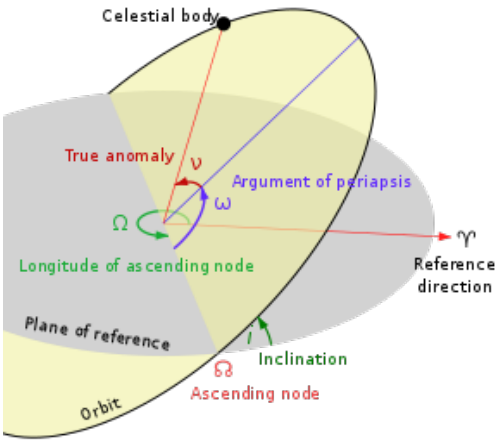


Figure 12: Definition of the traditional Keplerian orbital elements orbital elements, courtesy of Wikipedia. Two parameters define the shape and size of the ellipse; two define the orientation of the orbital plane; and the last two orient the ellipse in its plane and the phase of the body on its ellipse.

Perhaps the most interesting technical aspect of this problem is the details of combining multiple observations into a full uncertainty estimate for the distribution of possible paths. The Sentry system provides estimates of the values and uncertainties of the orbital elements that can be accessed at the JPL Small-Body browser. Like the Horizons system, this is an excellent resource that is free to the public. Crucially for this exercise, CNEOS supplies not only current estimates and uncertainties for the orbital elements, but a full 6x6 covariance matrix parameterized in the following six orbital elements:

- a, the semi-major axis; named `a` in JPL and REBOUND
- e, the eccentricity; named `e` in both systems
- i, the inclination; named `i` in JPL and `inc` in REBOUND
- Ω, the longitude of the ascending node; named `node` in JPL and `Omega` in REBOUND
- ω, the argument of perihelion; named `peri` in JPL and `omega` in REBOUND
- t_p, the time of perihelion passage; named `tp` in JPL and `T` in REBOUND

Distances are in A.U. in both JPL and REBOUND. Angles are quoted in degrees in JPL and in radians in REBOUND. The time t_p is measured in Julian Days at JPL. I switched the simulation units from years to days for this exercise to match JPL and work more easily with Python dates..

The next important point is understand exactly what the Horizons systems is returning when you query it. The detailed printout reveals that the orbital elements for Apophis were measured as of the epoch with Julian Date 2454733.5, corresponding to 2008-09-24. When we query Horizons for orbital elements, it generally gives us the output of its own orbital simulators evaluated as of the query date. A corollary is that if we queried Horizons as of the projected near approach date April 13, 2029, we would trivially match their answer. Here we query Horizons for the configuration as of the epoch, which in practice means we are only asking Horizons for the original orbital fit, then running our own simulation forward from that time.

If we wanted to really go hard core, we would need to go one step further back on the chain from pixels to orbits. If we carefully read the orbital solution, we can see that the "producer" Giorgini estimated this orbit on August 9, 2017 using 4481 total observations spanning 10.80 years between 2004-03 and and 2015-01. This is an interesting question I would like to investigate further, but is beyond the scope of this paper.

Orbital Elements at Epoch 2454733.5 (2008-Sep-24.0) TDB
Reference: JPL 199 (heliocentric ecliptic J2000)

Element	Value	Uncertainty (1-sigma)	Units
e	.1911953048308701	5.3461e-09	
a	.9224383019077086	4.1547e-10	au
q	.7460724295867941	4.7439e-09	au
i	3.331369520013644	3.5025e-07	deg
node	204.4460289189818	2.1065e-05	deg
peri	126.401879524849	2.0643e-05	deg
M	180.429373045644	5.4642e-06	deg
t_p	2454894.912519503203 (2009-Mar-04.41251950)	5.0014e-06	TDB
period	323.596949048484 0.89	2.1863e-07 5.986e-10	d yr
n	1.112495037603281	7.5161e-10	deg/d
Q	1.098804174228623	4.9491e-10	au

Orbit Determination Parameters

# obs. used (total)	4481
# delay obs. used	17
# Doppler obs. used	29
data-arc span	3946 days (10.80 yr)
first obs. used	2004-03-15
last obs. used	2015-01-03
planetary ephem.	DE431
SB-pert. ephem.	SB431-N16
condition code	0
norm. resid. RMS	.2745
source	ORB
producer	Giorgini
solution date	2017-Aug-09 01:08:15

Additional Model Parameters

Parameter	Value	Uncertainty (1-sigma)
A2 [EST]	-5.592840054057059E-14	2.201E-14
ALN [SET]	1.	n/a
NK [SET]	0.	n/a
NM [SET]	2.	n/a
R0 [SET]	1.	n/a

Additional Information

Earth MOID = .000315683 au
Jupiter MOID = 4.12582 au
T_jup = 6.466

Orbit Covariance (7×7)

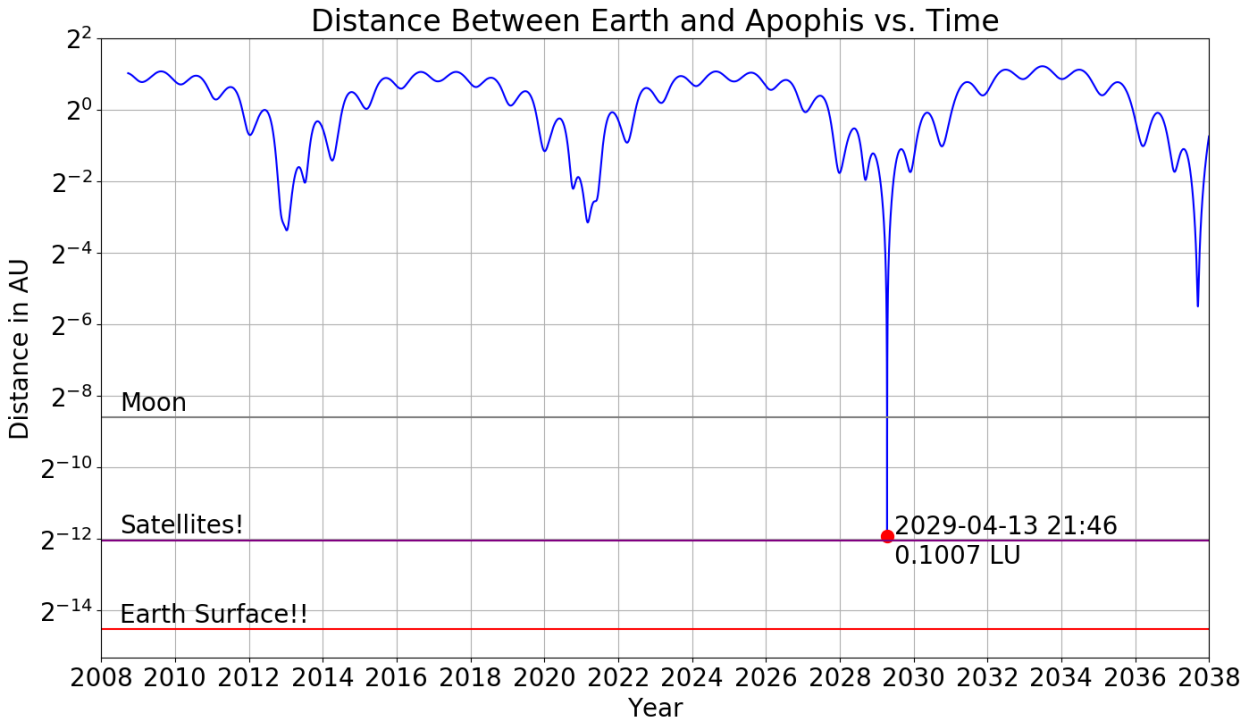
	e	q	tp	node	peri	i	A2
e	2.858062040774483E-17	-2.531903029559859E-17	2.451271598753632E-14	5.341143529000106E-14	-5.064059304794175E-14	-2.316088496620918E-16	-9.159706240530915E-23
q	-2.531903029559859E-17	2.250437378457998E-17	-2.122266051203493E-14	-5.100944859184798E-14	4.826834522209508E-14	2.76636664162405E-16	7.744081722922498E-23
tp	2.451271598753632E-14	-2.122266051203493E-14	2.501437350446983E-11	1.16213242311946E-11	-1.057653022558682E-11	4.597741246907787E-13	-1.026132427113056E-19
node	5.341143529000106E-14	-5.100944859184798E-14	1.16213242311946E-11	4.437388022822618E-10	-4.338840441053813E-10	-6.817498150073878E-12	-4.324000765783613E-21
peri	-5.064059304794175E-14	4.826834522209508E-14	-1.057653022558682E-11	-4.338840441053813E-10	4.261149273549589E-10	6.697032883689781E-12	1.139615135464918E-20
i	-2.316088496620918E-16	2.76636664162405E-16	4.597741246907787E-13	-6.817498150073878E-12	6.697032883689781E-12	1.22672053287042E-13	-2.524941685627351E-21
A2	-9.159706240530915E-23	7.744081722922498E-23	-1.026132427113056E-19	-4.324000765783613E-21	1.139615135464918E-20	-2.524941685627351E-21	4.846398125111792E-28

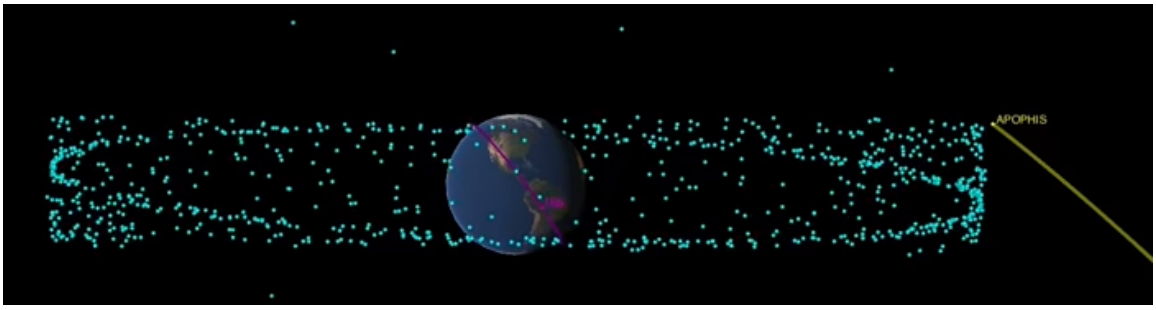
Figure 13: Orbital elements of Apophis courtesy of JPL.
This is the fitted orbit used as the starting point for ephemerides returned by the Horizons system.
A full covariance matrix of uncertainties is also included. See JPL Horizons - Apophis

Once the system is initialized, the function `simulate_apophis` performs a simulation over the next 32 years, covering both its near approach in 2029 and its return trip on 2037-09-27. The joke is that it takes more conceptual effort to synchronize the simulation time to the Gregorian calendar than it does to simulate the motion. This is done by adding the simulation time in days as a time delta to the base datetime of 2008-09-24 0:00:00. I performed the simulation at an interval of every 15 minutes, finding by trial and error that was sharp enough to resolve the close approach accurately. The function `calc_xyz` extracts the Cartesian coordinates of Earth and Apophis at every time snap taken by the simulation. These are guaranteed to be at most 15 minutes apart, but may be closer. `calc_dist` computes the distance and `plot_approach_dist` creates a visualization of the approach.

My simulation showed that Apophis will approach to within 0.1007 lunar units at 2029-04-14 21:46. As the screenshot from NASA shows, these results tie out to all available significant figures and are accurate to the minute! I wondered for a moment how I was able to match to the nearest minute using `np.argmin` rather than a spline for the time of closest approach. It's because of the adaptive time stepping of `ias15`. This forces the integrator to run a time step that ends very close to the time of nearest approach. It's an excellent example of the benefits of using an adaptive integrator.

I was pleased to have such a tight agreement to NASA. Getting the calculation this accurate is not straightforward. The orbital elements page includes a figure that the MOID (minimal orbital intersection distance) of Earth and Apophis is 0.000316 AU. Note that this is barely accurate to even the first significant figure of the right answer, which is 0.000259 AU. This demonstrates that a Keplerian approximation is good enough to determine if a near approach might occur, but is insufficient to resolve them accurately. The visualization shows the distance between Earth an Apophis on a log scale. I've included three reference lines, depicting the distance from earth's center to the moon, to satellites in geosynchronous orbit, and to the earth's surface. This makes it easy to that Apophis will get much closer than the moon and graze by satellites in near earth orbit. The frame shown underneath comes from an excellent Wikipedia visualization movie. Adding all of the artificial satellites was a great idea and gives a strong visual cue of the proximity of the approach.





Object	Close-Approach (CA) Date	CA Distance Nominal (LD | au)
(2011 ES4)	2020-Sep-01 14:49 ± 7_02:42	0.21 | 0.00053
153814 (2001 WN5)	2028-Jun-26 05:23 ± < 00:01	0.65 | 0.00166
99942 Apophis (2004 MN4)	2029-Apr-13 21:46 ± < 00:01	0.10 | 0.00025

Figure 14: The near earth approach of Apophis projected on April 13, 2029 at 21:46.
My simulation agrees with NASA on the time of approach down to the minute, at 21:46.
I also agree with them to at least 2 significant figures on the distance of closest approach at 0.101 Lunar Units.
Please see CNEOS - NEO Earth Close Approaches for the live predictions from NASA.
The visualization of the Earth its satellites comes from the video on the Apophis Wikipedia page

4.2 Simulating 1000 Sampled Trajectories of Apophis

The idea behind this exercise is that we don't exactly know the configuration (position and velocity) of Apophis. We would like to quantify our level of uncertainty over its future trajectory. We know from the JPL system the covariance matrix of uncertainties for its six orbital elements.

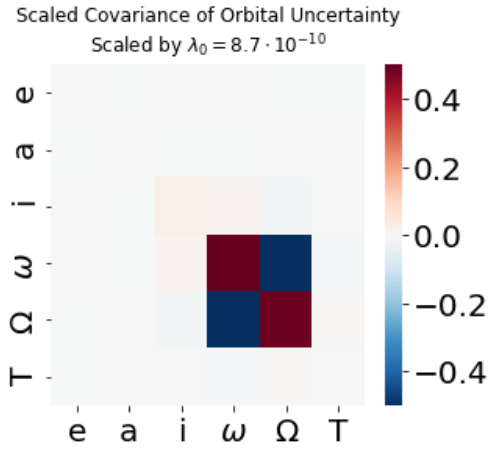


Figure 15: Heatmap for the covariance matrix of the orbital elements of Apophis.
Entries have been scaled up the largest eigenvalue to make it more understandable.
The covariance is dominated by an uncertainty between the angular parameters Ω and ω.
Their sum is well resolved, their difference is more uncertain.

The function `jpl_cov_apophis` returns a hard-coded version of the 6x6 covariance matrix along with a list of the associated field names. A visualization of the covariance matrix reveals that is dominated by an uncertainty between Ω and ω but a much higher confidence in their sum. The largest eigenvalue of the covariance matrix is 8.7E-10, giving an idea of how tightly determined the parameter values are.

The function `sample_orbital_elements` draws a sample of perturbed orbital elements using this covariance matrix. It starts by extracting the orbital elements of the unperturbed Apophis as the starting point. I verified that a round trip of converting back and forth to Cartesian coordinates was extremely accurate. It then uses the numpy function `np.random.multivariate` to draw a sample of perturbations with the specified covariance. Shifts to angular variables i, Ω and ω are transformed from degrees (in the JPL cov matrix and drawn perturbations) into radians before being applied. The elements are returned in a dictionary keyed by variable name rather than an Nx6 array to avoid errors from hard coded variable orders.

The function `sim_times` is used to generate a more parsimonious list of times as which we want snapshots. For the base case, I took uniform snapshots every 15 minutes over a 32 year window. That is too expensive with large numbers of test objects in the system. Instead, this function creates a schedule with 4 snapshots per day most of the time. For a window of 45 days on either side of the projected near approach, snapshots are taken at the same high resolution of every 15 minutes.

The function `simulate_apophis_scenarios` simulates a collection of perturbed Apophis orbits given a dictionary of their orbital elements. They are added using a syntax that allows addition of a new particle with its orbital elements rather than Cartesian coordinates. (This is actually the preferred way to communicate orbits and is how JPL does it.) The particles are specified to be test particles by setting `sim.N_active = 11` after adding them. Otherwise the simulation will bog down with $O(N^2)$ gravitational interactions between simulated copies of Apophis with zero mass. The largest practical difficulty was that my first attempt used simulation archives. The files were way too big, and even after I generated them, iterating through them crashed my program repeatedly. Instead, I compute the distances in situ using the function `scenario_dist`. This in turn uses the method `sim.serialize_particle` to efficiently get at the underlying particle array

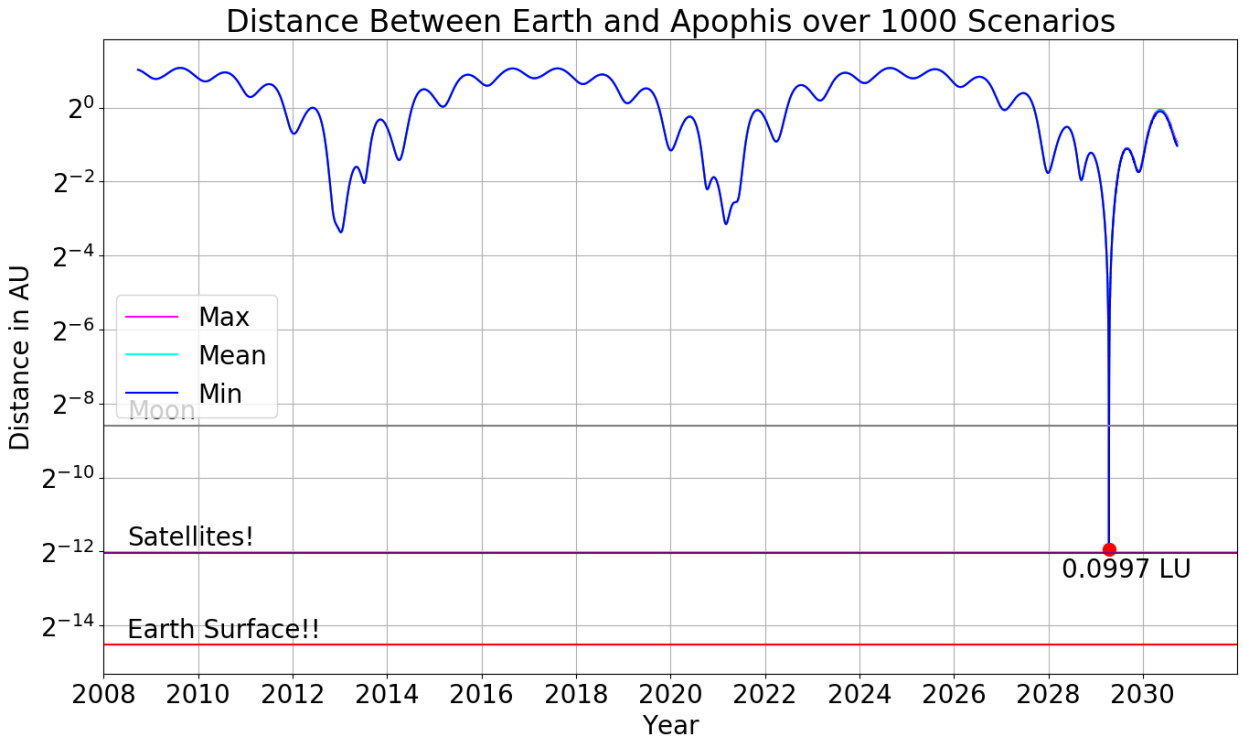


Figure 16: The range of distances for Apophis to Earth over 1000 scenarios drawn with the current orbital covariance matrix. All of the scenarios are visually indistinguishable on a log scale.

purely on a C level. It requires creating a numpy array of the right size ahead of time. This one small change eliminated a painful bottleneck and significantly improved the program's performance. The function returns an array with the distance of Apophis to earth indexed over scenarios and time steps. Simulating over 40,422 snapshots with 1,000 scenarios took between 8 and 13 minutes on my Linux server.

I present a visualization formatted consistently with the original one. Instead of the one distance vs. time, I plot the mean, min and max series. These are all visually indistinguishable, even on a long scale during the near approach. A variance plot (not shown) reveals that the maximum deviation over 1000 samples jumps from around 2^{-18} AU to 2^{-6} AU after the near approach. Our orbital parameter estimates of Apophis are so tight that we know where it's going.

While it's good news that we know where Apophis is going, it doesn't make for a very exciting visualization. To spice things up a bit, I ran three alternate simulations with the covariance boosted by a increasing factors. I chose scaling factors of 1E2, 1E4 and 1E6, corresponding to scaling parameter uncertainties σ by 10, 100 and 1000. With σ increased by 10, the nearest approach of the 1000 samples gets to within 0.0906, lunar units, slightly closer than the 0.1007 in the base case. With σ increased by 100, the nearest approach of the 1000 samples gets to within 0.0154 lunar units... wait a minute–that's a collision! Earth's radius is 0.0166 lunar units. There is just 1 collision out of 1,000 trials. With σ increased by 1000, the nearest approach is 0.0037 lunar units, which a golf player might describe as a "center cut" on a good putt. This time there are 2 collisions out of 1000 trials.

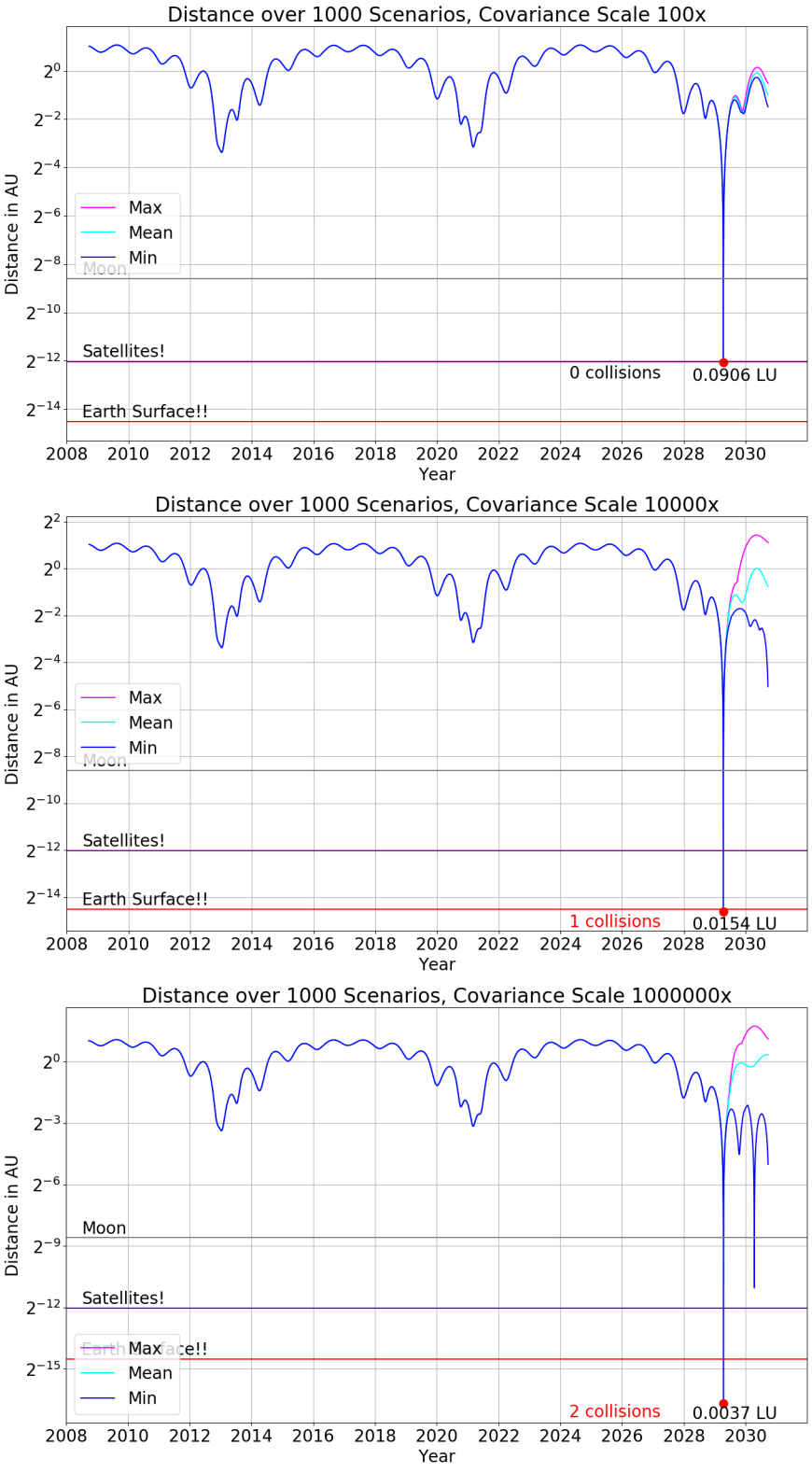


Figure 17: The range of distances for Apophis to Earth over 1000 scenarios drawn with a covariance matrices that have been scaled by 10, 100 and 1000 times.
The latter two simulations reveal 1 and 2 collisions with the earth respectively!

5 Conclusions and Future Work

The N-body problem is a classic that is still of both practical and theoretical interest. In this project I have gained proficiency in using the state of art integrator, the open source project `REBOUND`. I reinforced two lessons about software engineering and life. Before you write a serious piece of software, make a careful study of the best available library. An hour spent reading the documentation of a good library can be worth 100 hours of code you don't write. And don't limit research to online searches. If you know a person with domain knowledge, talk to them, preferably in person!

The `REBOUND` integrator package achieves a high performance with a convenient Python API. Using it, I was able to tackle a full scale solar simulation to a degree of accuracy comparable to that of the JPL at NASA. In experiments with two of the stronger integrators, `ias15` and `whfast`, I determined that `ias15` is very accurate out of the box, and if it's not prohibitively slow for your problem, it's a good starting choice. On some problems, however, `ias15` is prohibitively slow with the default parameter settings. On those problems, good alternatives include using `whfast` or the hybrid integrator `mercurius`, which I didn't have time to try here but which I suspect would be the best option.

Using this familiarity with `REBOUND`, I was able to run an extremely accurate simulation of the near earth approach of the asteroid Apophis, matching NASA's estimate of the projected time of its nearest approach to earth to a tolerance of one minute. Not too shabby for one person sitting at home armed only with a computer, Google, and two semesters of Applied Math fun at Harvard. Finally, I was able to generate a range of realistic scenarios for possible trajectories of Apophis by using the covariance matrix of uncertainty in the estimates of its orbital elements. This analysis agrees with NASA's conclusion that we have a sufficiently tight bound on the orbit of Apophis that we can be assured it will not impact the earth on a 100 year time scale. To prove out this methodology, I also simulated scenarios with larger uncertainty in the orbital parameters. These simulations revealed that if the uncertainty window around the orbital elements were increased by a factor of 100, an earth strike in 2029 is a remote but real possibility, happening once in a sample of 1000 trials.

Matt Holman and Matt Payne had an excellent suggestion for future work which I might take up at some point. In their work at the CFA, it would be useful if they could load a "movie" with a high resolution simulation of the solar system's evolution with all the planets and heavy bodies. They would like to be able to apply the configuration of the heavy objects to quickly integrate a large number of test bodies (assumed to be massless) without needing to re-integrate the heavy bodies. Ideally this should run in parallel; the problem is certainly amenable to an embarrassingly parallel approach. `REBOUND` does not currently support this mode of operation. I believe that adding it would be a nontrivial challenge but one that should be feasible for me or someone similarly situated.

6 Acknowledgments

I would like to thank Matt Holman and Matt Payne at the Center for Astrophysics for generously sharing their time and expertise.
I would like to thank Professor Rycroft and teaching fellow Nick Derr for all their work and knowledge. It has been a great semester and I've learned a lot. [17]
Finally I would like to thank my wife Christie and my children Victor and Renée. The long hours of course work have a heavy strain on our family life, and all three of them have been tremendously patient and supportive of my efforts.

[17] This course has also come close to killing me, mostly in a good way, but I'm relieved to be at the finish line.

References

[1] Jack Wisdom, Matthew Holman.
Symplectic Maps for the N-Body Problem.
The Astronomical Journal, Volume 102, Number 4. October 1991.

[2] J.E. Chambers.
A hybrid symplectic integrator that permits close encounters between massive bodies.
Monthly Notices of the Royal Astronomical Society 304, 793-799 (1999). Accepted December 2, 1998.

[3] Hanno Rein, Shang-Fei Liu
REBOUND: An open-source multi-purpose N-body code for collisional dynamics.
Astronomy & Astrophysics. November 11, 2011.
arXiv: 1110.4876v2

[4] Hanno Rein, David S. Spiegel
IAS15: A fast, adaptive high-order integrator for gravitational dynamics, accurate to machine precision over a billion orbits.
Monthly Notices of the Royal Astronomical Society. Printed 16 October 2014.
arXiv: 1405.4779.v2

[5] Hanno Rein, Daniel Tamayo
WHFast: A fast and unbiased implementation of a symplectic Wisdom-Holman integrator for long term gravitational simulations.
Monthly Notices of the Royal Astronomical Society. Printed 4 June 2015.
arXiv: 1506.01084v1.v2