Numerical Investigation of Chaotic Motion in the Asteroid Belt

Danya Rose

An essay submitted in partial fulfillment of
the requirements for the degree of
B.Sc. (Honours)

Applied Mathematics
University of Sydney

November 2008

Acknowledgements

Many thanks to my supervisor, Dave Galloway. Also to David Ivers, who supervised me while Dave was overseas, without whose numerical expertise my project might have remained Earthbound.

CONTENTS

Introduction

Many systems and processes that appear in nature can be modelled by differential equations. Some can be solved explicitly, so that their solutions are known for all times that the conditions are valid. Most, however, are not so tractable; analytic approximations that are valid near certain regions can allow us to extract some useful and important information such that much of the overall dynamics can be pieced together. With the advent of modern desktop computing, however, it has become possible to perform numerous calculations that approximate the full equations of motion directly.

One such complicated system is the N-body gravitational problem - essentially that of the Solar System. A common feature of such complicated systems is chaos and the emergence of unexpected structures from often elegant systems of equations. The Solar System is rife with examples of unexpected, interesting and beautiful structures, from the (seemingly, but not really) clockwork motion of the elliptical orbits of the planets to the majesty of Saturn's rings. Among such structures are the Kirkwood gaps, little known outside those who study the Solar System in detail, whose origins are now thought to lie in overlapping resonances with Jupiter resulting in unstable chaotic trajectories that ultimately lead to the ejection of asteroids from these small bands of the total asteroid belt between Mars and Jupiter.

Numerical methods allow us to use computers as experimental apparatus to study systems whose natural scales are beyond our ability to observe easily - in the case of the Solar System and its long term evolution, and more particularly here understanding the formation of the Kirkwood gaps - and to test the models we have constructed. To do so, however, requires relevant tools, such as numerical integration techniques that preserve fundamental properties of the system of interest. For a system that can be written as a Hamiltonian, like the N-body problem, a symplectic integrator will preserve fundamental geometric properties of the system's phase space and remain stable for integrations over remarkably long time spans (millions to tens of millions of years), unlike other integration schemes such as Runge-Kutta (which can in fact be more accurate for shorter-time integrations, for a given number of time steps) ([**1**]).

CHAPTER 1

Background and Motivation

1.1. The Kirkwood Gaps

First discovered in the 1860s by Daniel Kirkwood, the Kirkwood gaps are "underpopulated" regions of the asteroid belt between Mars and Jupiter that occur in the vicinity of certain small-number mean motion resonances with Jupiter.

The main Kirkwood gaps occur at the 4:1, 3:1, 5:2, 7:3 and 2:1 mean motion resonances with Jupiter, respectively corresponding to semi-major axes of 2.06, 2.5, 2.82, 2.95 and 3.27 AU. Weaker gaps appear at 1.9 AU (9:2 resonance), 2.25 AU (7:2 resonance), 2.33 AU (10:3 resonance), 2.71 AU (8:3 resonance), 3.03 AU (9:4 resonance), 3.075 AU (11:5 resonance), 3.47 AU (11:6 resonance), 3.7 AU (5:3 resonance). Figure 1 shows the spacing of the major Kirkwood gaps.

Originally, the Kirkwood gaps were thought to be sufficiently explained by simply an extra gravitational tug at the point in an asteroid's orbit where it passed closest to Jupiter. For example, in the 2:1 resonance, the asteroid makes two revolutions about the sun for every one of Jupiter. In this model, the extra strong tug every two revolutions of the asteroid was believed to add up over a long time to such a degree that the 2:1 resonance and the narrow region around it would be depleted of asteroids. However; detailed analysis has shown that this effect is insufficient to account for the depletion we see.

More recent studies ([**2**], [**3**], [**4**], [**5**], [**6**], [**7**], [**6**], [**8**]) have shown that unstable chaotic regions may form in the location of the major resonances, particularly the 3:1 resonance, though the effects of Saturn appear to be necessary to account for the level of depletion we observe by "mixing out" stable pockets that would otherwise remain [**2**].

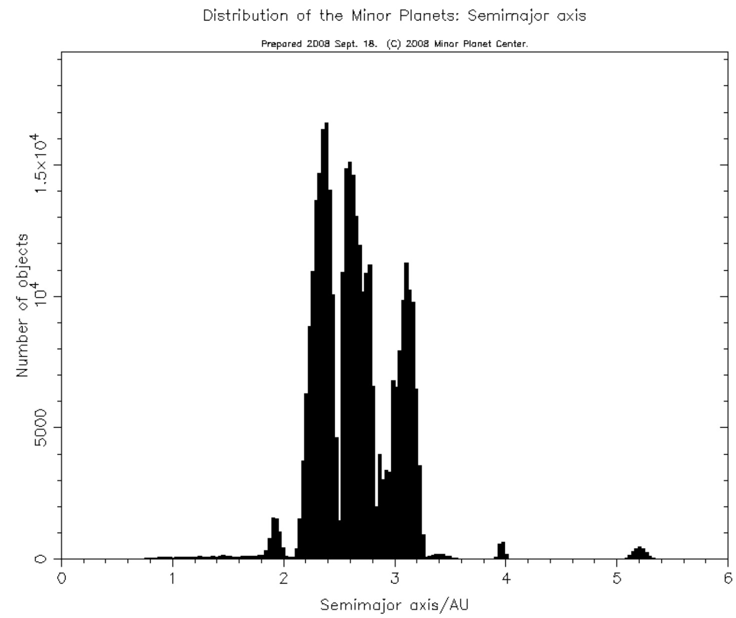


FIGURE 1. Histogram of asteroids by semi-major axis. The major Kirkwood gaps are clearly visible. Image courtesy of the MPC: *http://www.cfa.harvard.edu/iau/lists/MPDistribution.html*

Studies by Wisdom ([**9**], [**10**], [**11**]) suggest that while chaos induced by Jupiter at the 3:1 resonance may be responsible for large excursions of eccentricity after several thousand years of apparently regular behaviour, it is Mars that actually removes the asteroid by direct perturbation.

1.2. Resonance

A *mean motion resonance* occurs when the ratio of the orbital periods of two co-orbiting bodies is itself rational; i.e., the orbital period of a body is $T_1 = \frac{p}{q}T_2$, where p and q are integers. Resonances are abundant in the Solar system and require special treatment to be understood properly from an analytical standpoint, as naïve approaches often fail. Murray & Dermott give an excellent treatment of the topic in Chapter 8 of *Solar System Dynamics*, [**12**].

As the rationals are dense in $\mathbb{R}$, it seems odd, perhaps, that the Kirkwood gaps only appear near small integer resonances with Jupiter. Murray and

Holman in [**13**] provide a summary of how overlapping resonances produce large scale chaos, which can expose the asteroid to a larger volume of phase space than would otherwise be available - possibly leading to encounters with other bodies.

1.3. Chaos

Chaos is defined as sensitive dependence on initial conditions. This sensitive dependence results in an exponential divergence between trajectories with “nearby” initial conditions. Any system with enough coupled degrees of freedom may express chaotic effects, such as the coupling implicit in mutual gravitational attraction for the N-body problem, with $N > 2$.

Chaotic systems can be characterised by a specific time scale, called the Lyapunov time, which is defined as the time taken for two nearby trajectories to diverge exponentially by a factor of e.

1.4. Statement of problem

This project is a numerical investigation of the chaotic behaviour of asteroids in the asteroid belt between Mars and Jupiter, using symplectic aglorithms that do not become catastrophically inaccurate over long term integrations (on the order of millions to hundreds of millions of years). Specifically, I wish to discover if I can replicate the effect of unstable chaotic zones believed to be responsible for the Kirkwood gaps at specific mean motion resonances with Jupiter and investigate the significance of Saturn in the formation of the Kirkwood gaps.

A Hamiltonian approach to the equations of motion of the system will be used, as that is the basis of symplectic integration. Second and fourth order symplectic routines will be implemented in an appropriate language and tested on a simple Hamiltonian system, and then used to integrate the 3- and 4-body problems in 3 spatial dimensions.

1.4.1. The asteroid's dynamics. To understand the dynamics of the asteroid, it is useful to calculate its osculating orbital elements (described in appendix A) - that is, its orbital elements (eccentricity, semi-major axis, inclination, argument of perihelion, ascending node and true anomaly) as if in any instant it exists only in a two-body system composed of the asteroid and the Sun (also called the Kepler problem). By tracking the changes of eccentricity e and semi-major axis a with time it is possible to tell whether the asteroid is in a stable orbit (e often librates like small-amplitude, high-frequency sinusoids superpositioned with large-amplitude, low-frequency sinusoids and a is nearly constant) or unstable (e varies widely and aperiodically, a can vary visibly, sometimes resulting in a new stable orbit).

1.4.2. Accuracy. As the computer acts as an experimental laboratory for this problem, it cannot be assumed that the output of a program that numerically integrates the equations of motion truly represents the dynamics of the real system. Roundoff error is unavoidable and must be estimated and accounted for, and the accuracy of the integration routines themselves must also be tested.

CHAPTER 2

Methods

2.1. Hamiltonian Representation

The Hamiltonian is of the separable form $H(p, q, t) = T(p) + V(q)$ and is independent of t. We have

$$T = \frac{1}{2}\sum_{i=1}^{n}\frac{p_i^2}{m_i}$$

and

$$V = -\sum_{i=2}^{N}\sum_{j=1}^{i-1}\frac{Gm_im_j}{|\mathbf{q}_i - \mathbf{q}_j|},$$

where N is the number of bodies, m_i, p_i and q_i are respectively the mass, momentum and position of body i. m is scalar, $\mathbf{p}_i$ and $\mathbf{q}_i$ are 3-vectors parametrised by time, while p_i and q_i represent the magnitudes of these vectors.

The Hamiltonian formulation gives $6N$ coupled ODEs for the equations of motion, and the system has $3N$ degrees of freedom. Each body has three degrees of freedom and my simulations will have four bodies (the sun, an asteroid, Jupiter and Saturn), so 24 equations in total (x, y, z, p_x, p_y, p_z for each body) - 18 when Saturn is neglected.

We have

$$\dot{\mathbf{q}}_i = \nabla_{\mathbf{p}_i}H = \frac{\mathbf{p}_i}{m_i}$$

and

$$\dot{\mathbf{p}}_i = \nabla_{\mathbf{q}_i} H = -Gm_i \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{m_j(\mathbf{q}_i - \mathbf{q}_j)}{|\mathbf{q}_i - \mathbf{q}_j|^3}.$$

The vector differential operator $\nabla_{\mathbf{x}}$ works the same as ∇, but specifically applies only to the vector variable $\mathbf{x}$.

2.2. Symplectic Mapping and Geometric Integration

Symplecticity is a geometric property of Hamiltonian systems. A symplectic matrix M has the property that $M^* J M = J$, where $J = \begin{pmatrix} 0 & I_{3N} \\ -I_{3N} & 0 \end{pmatrix}$, I_{3N} is the $3N \times 3N$ identity matrix (to be consistent with the number of degrees of freedom above) and $M^* = M^{-1}$ is M's adjoint.

An important feature of symplectic mapping/phase space structure is that volume in phase space is conserved under the flow of solutions (i.e. Liouville's theorem is a consequence of symplecticity).

2.3. Integration Schemes

2.3.1. First order approach. To derive the integrator begin with Euler's method:

Approximate the time derivatives in the equations of motion by

$$\begin{aligned} \frac{\mathbf{q}_{i_{n+1}} - \mathbf{q}_{i_n}}{\tau} &= \nabla_{\mathbf{p}_{i_n}} H \\ \frac{\mathbf{p}_{i_{n+1}} - \mathbf{p}_{i_n}}{\tau} &= \nabla_{\mathbf{q}_{i_n}} H, \end{aligned}$$

where $\mathbf{q}_{i_n} = \mathbf{q}_i(t_n)$, $\nabla_{\mathbf{q}_{i_n}} H = \nabla_{\mathbf{q}_i} H|_{t=t_n}$ (similarly for $\mathbf{p}_{i_n}$) and $\tau = t_{n+1} - t_n$. This gives us

$$\mathbf{q}_{i_{n+1}} = \mathbf{q}_{i_n} + \tau \nabla_{\mathbf{p}_{i_n}} H$$
$$\mathbf{p}_{i_{n+1}} = \mathbf{p}_{i_n} - \tau \nabla_{\mathbf{q}_{i_n}} H.$$

As this stands it is not symplectic, but it becomes so if in the second equation $\mathbf{q}_{i_n}$ is replaced by $\mathbf{q}_{i_{n+1}}$ (the map M obtained by this change preserves symplectic structure: i.e. $M^* J M = J$). This is still only a first order algorithm, but the composition of this map with its adjoint (swap n with $n+1$ and replace τ by $-\tau$ and solve for $\mathbf{q}_{i_{n+1}}$ and $\mathbf{p}_{i_{n+1}}$ to get the adjoint map) creates a second order method called the Störmer-Verlet, or leapfrog routine, which is symplectic (compositions of symplectic maps are symplectic: $M_{comp}^* J M_{comp} = M_2^* M_1^* J M_1 M_2 = M_2^* J M_2 = J$, if M_1 and M_2 are symplectic).

2.3.2. Derivation of leapfrog algorithm. Let the symplectic Euler map with timestep τ be Φ_τ, and let its adjoint be Φ_τ^{-1}.

$$\begin{aligned}
\Phi_\tau : \mathbf{q}_{i_{n+1}} &= \mathbf{q}_{i_n} + \tau \nabla_{\mathbf{p}_{i_n}} H \\
\mathbf{p}_{i_{n+1}} &= \mathbf{p}_{i_n} - \tau \nabla_{\mathbf{q}_{i_{n+1}}} H \\
\Phi_\tau^{-1} : \mathbf{p}_{i_{n+1}} &= \mathbf{p}_{i_n} - \tau \nabla_{\mathbf{q}_{i_n}} H \\
\mathbf{q}_{i_{n+1}} &= \mathbf{q}_{i_n} + \tau \nabla_{\mathbf{p}_{i_{n+1}}} H.
\end{aligned}$$

To compose these maps, introduce a "half timestep" $n + \frac{1}{2}$ and use a step size of $\frac{\tau}{2}$. Now we compose $\Phi_{\frac{\tau}{2}} \circ \Phi^*_{\frac{\tau}{2}}$:

$$\begin{aligned}
\mathbf{q}_{i_{n+\frac{1}{2}}} &= \mathbf{q}_{i_n} + \frac{\tau}{2} \nabla_{\mathbf{p}_{i_n}} H \longrightarrow (1) \\
\mathbf{p}_{i_{n+\frac{1}{2}}} &= \mathbf{p}_{i_n} - \frac{\tau}{2} \nabla_{\mathbf{q}_{i_{n+\frac{1}{2}}}} H \longrightarrow (2) \\
\mathbf{p}_{i_{n+1}} &= \mathbf{p}_{i_{n+\frac{1}{2}}} - \frac{\tau}{2} \nabla_{\mathbf{q}_{i_{n+\frac{1}{2}}}} H \longrightarrow (3) \\
\mathbf{q}_{i_{n+1}} &= \mathbf{q}_{i_{n+\frac{1}{2}}} + \frac{\tau}{2} \nabla_{\mathbf{p}_{i_{n+1}}} H \longrightarrow (4).
\end{aligned}$$

Sub (2) into (3) to get $\mathbf{p}_{i_{n+1}} = \mathbf{p}_{i_n} - \tau \nabla_{\mathbf{q}_{i_{n+\frac{1}{2}}}} H$ and we have

$$\mathbf{q}_{i_{n+\frac{1}{2}}} = \mathbf{q}_{i_n} + \frac{\tau}{2}\nabla_{\mathbf{p}_{i_n}} H$$
$$\mathbf{p}_{i_{n+1}} = \mathbf{p}_{i_n} - \tau\nabla_{\mathbf{q}_{i_{n+\frac{1}{2}}}} H$$
$$\mathbf{q}_{i_{n+1}} = \mathbf{q}_{i_{n+\frac{1}{2}}} + \frac{\tau}{2}\nabla_{\mathbf{p}_{i_{n+1}}} H.$$

The composition $\Phi_\tau^* \circ \Phi_\tau$ produces a similar map

$$\mathbf{p}_{i_{n+\frac{1}{2}}} = \mathbf{p}_{i_n} + \frac{\tau}{2}\nabla_{\mathbf{q}_{i_n}} H$$
$$\mathbf{q}_{i_{n+1}} = \mathbf{q}_{i_n} - \tau\nabla_{\mathbf{p}_{i_{n+\frac{1}{2}}}} H$$
$$\mathbf{p}_{i_{n+1}} = \mathbf{p}_{i_{n+\frac{1}{2}}} + \frac{\tau}{2}\nabla_{\mathbf{q}_{i_{n+1}}} H.$$

The calculation of force in $\nabla_{\mathbf{q}_{i_{n+1}}} H$ is an $\mathrm{O}(N^2)$ operation (where N is the number of bodies) over all bodies i per time step, while $\nabla_{\mathbf{q}_{i_n}} H$ is $\mathrm{O}(N)$ over all i. Thus when this is taken into account, the former version of the leapfrog algorithm is more efficient. Both are accurate to second order.

The leapfrog scheme derives its name from the fact that it computes a "substep" $n + \frac{1}{2}$, from which one can complete the full time step.

2.3.3. Fourth order Forest & Ruth. This routine was independently discovered and published by Forest & Ruth [**14**], Candy & Rozmus [**15**] and Yoshida [**16**] circa 1990. Yoshida in particular gives an elegant way to derive the integration coefficients for higher even-order symplectic routines for separable Hamiltonians, though none are used in this project. Higher order routines would only be of value with higher numerical accuracy or much larger time steps.

2.4. The Integrator

In general, the routine for an even-order, symmetric symplectic integrator suitable for separable Hamiltonians has two arrays of integration coefficients (a and b, say), whose lengths are m and $m-1$. The routine calculates $2m-1$ substeps (in both $\mathbf{p}$ and $\mathbf{q}$) in going from step n to step $n+1$.

For example, the leapfrog algorithm has arrays of integration coefficients $a_1 = \frac{1}{2}, a_2 = \frac{1}{2}$ and $b_1 = 1$ (thus $m = 2$) and has the form

- $\mathbf{q}_{n+\frac{1}{2}} = \mathbf{q}_n + a_1 \tau \nabla_{\mathbf{p}} H|_n$;
- $\mathbf{p}_{n+1} = \mathbf{p}_n - b_1 \tau \nabla_{\mathbf{q}} H|_{n+\frac{1}{2}}$
- $\mathbf{q}_{n+1} = \mathbf{q}_{n+\frac{1}{2}} + a_2 \tau \nabla_{\mathbf{p}} H|_{n+1}$,

exactly as before. Note that $a_j = a_{m-j+1}$ and $b_j = b_{m-j}$ for integer $1 \leq j < m$.

This generalises for arbitrary m:

- $\mathbf{q}_{n+\frac{1}{m}} = \mathbf{q}_n + a_1 \tau \nabla_{\mathbf{p}} H|_n$;
- $\mathbf{p}_{n+\frac{1}{m-1}} = \mathbf{p}_n - b_1 \tau \nabla_{\mathbf{q}} H|_{n+\frac{1}{m}}$
- $\mathbf{q}_{n+\frac{2}{m}} = \mathbf{q}_{n+\frac{1}{m}} + a_2 \tau \nabla_{\mathbf{p}} H|_{n+\frac{1}{m-1}}$

  $\vdots$
- $\mathbf{p}_{n+1} = \mathbf{p}_{n+\frac{m-2}{m-1}} - b_{m-1} \tau \nabla_{\mathbf{q}} H|_{n+\frac{m-1}{m}}$
- $\mathbf{q}_{n+1} = \mathbf{q}_{n+\frac{m-1}{m}} + a_m \tau \nabla_{\mathbf{p}} H|_{n+1}$.

Alternatively, p and q can be swapped to produce a different algorithm of the same order.

More generally, however, a and b are of the same length m, but the first or last element of b or a (respectively) is in fact 0. McLachlan and Atela in [**17**] show that this is not optimal in terms of theoretical accuracy. The optimal second order routine has coefficients $a_1 = \frac{1}{\sqrt{2}}, a_2 = 1 - \frac{1}{\sqrt{2}}$ and $b_1 = \frac{1}{\sqrt{2}}, b_2 = 1 - \frac{1}{\sqrt{2}}$. This algorithm is not used here, however, as it is not symmetric.

2.4.1. Error Testing. Because the system is Hamiltonian, it is exactly time reversible. Thus it is possible to replace t by $-t$ everywhere in the equations of motion with the new solution representing the flow of the former solution backwards in time. Also, both the leapfrog and Forest & Ruth integration algorithms are symmetric in the time step, so the same routine can be used to integrate both forwards and backwards in time. Therefore, a good way to discover the amount and effect of numerical error may be to run the integrator forward for a given length of time, T, say, set $\mathbf{p} = -\mathbf{p}$

and allow the integrator to continue until $t = 2T$. Because of the aforementioned symmetry, this has the same effect as reversing the time flow and returning to $t = 0$; ideally, when $t = 2T$ the system will be in the same spatial configuration as it was when $t = 0$. Finite-precision arithmetic will prevent this from ever being the case, but the closer the agreement between configurations the better.

Estimations of the amount of truncation error are discussed in section 3.1.

2.4.2. MATLAB implementation. A complete listing of the MATLAB source code is given in appendix F1. Initial conditions are found in [**18**], but are given in appendix B for convenience.

The main integrations routines are *asteroid_integrate*, *asteroid_resume_run*. The former begins with the initial conditions and integrates forward a given number of steps (storing data to a buffer every *storefrequency* steps and dumping the buffer to disk when full) and finishes. However, a flag may be set that reverses the flow (as discussed above) and continues to integrate until the system reaches $t = 2T$, equivalent to $t = 0$.

It is important to be able to choose arbitrary (within reason) initial conditions for the asteroid, in particular specifying its mean motion relative to Jupiter and its eccentricity. Limitations on the exact arbitrariness of the asteroid's initial conditions are discussed in section 3.2.2, and the means of determining its exact position and velocity from the desired relative mean motion and eccentricity, given the necessary restrictions, are shown in appendix C.

The routine *asteroid_resume_run* scans through the data stored on disk and attempts to resume an integration run by taking the last completely recorded set of data (positions and momenta for each body) and uses this to resume a run that has been interrupted part way through. It is designed to resume a run at any possible stage - during its forward part and continue its reverse run (if any) or during the reverse stage of a run.

If the eccentricity of the asteroid at any stage exceeds 0.8, the run is terminated, as it has certainly become a Mars (or even Earth or Venus) crosser for any given semi-major axis within the main asteroid belt and is likely to be removed from the zone of interest due to close encounters.

MATLAB, being an interpreter for its code, takes several days to compute 10^8 time steps, as we routinely wish to do, so a faster solution is much desired.

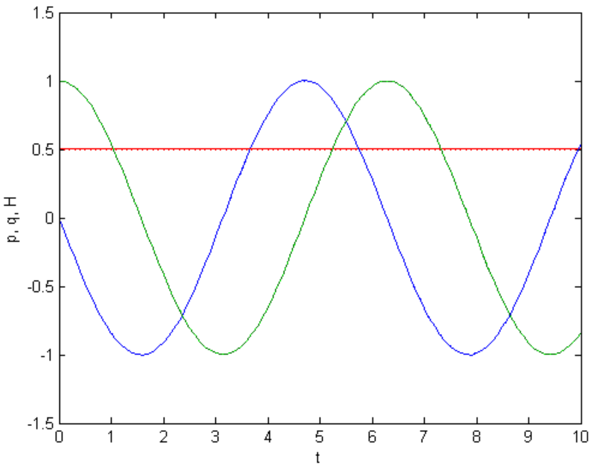
2.4.3. Simple tests. A test to make sure the algorithms work correctly is the simple harmonic oscillator, whose Hamiltonian is $H = \frac{1}{2}(p^2 + q^2)$ with general solution $q(t) = A\cos(t) + B\sin(t)$ and $p(t) = -A\sin(t) + B\cos(t)$. Given initial condition $q_0 = 1$, $p_0 = 0$, the particular solution is $q(t) = \cos(t)$, $p(t) = -\sin(t)$.

This was implemented in MATLAB, and short tests were conducted for 100 time steps of $\tau = 0.1$. Figures 1a and 1b show the calculated evolution of the system (solid lines) along with the exact solutions (dotted lines), while 1c and 1d show the differences between the calculated solutions and the exact solutions. The fourth order integrator performs approximately two orders of magnitude better than the leapfrog integrator over the short time investigated.

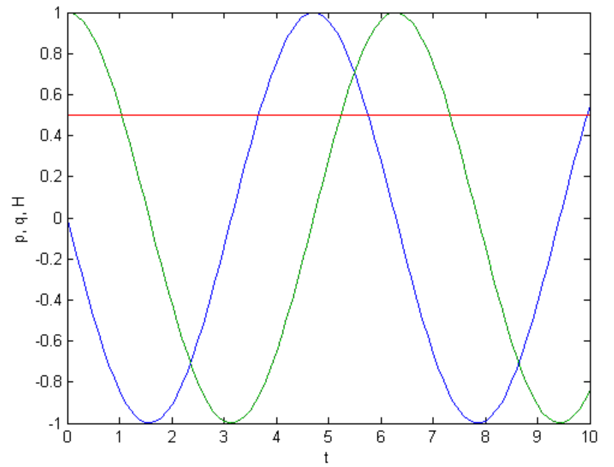
2.4.4. Fortran implementation. The Fortran implementation is similar to the MATLAB version, but integrates the functions of the two routines above into one executable. It is also capable of reading through batches of initial conditions and parameters (number of steps, buffering frequency, buffer size, etc.) to ease the process of doing large numbers of integrations. It also reduces the time taken to integrate 10^8 time steps from nearly a week down to less than a day, thanks in part to the efficiency of its compilers as a mature language.

Although efficiency was already much improved by porting the integrator to Fortran, further improvement could have been made by recognising that the matrix of forces between bodies is antisymmetric on the main diagonal (expected from Newton's third law of motion). The force matrix has elements $\mathbf{F}_{ij}$, the force exerted on body i by body j, where $1 \leq i, j \leq N$ (N being the number of bodies) and $i \neq j$. It would have been possible to calculate only the forces $\mathbf{F}_{ij}$ with $1 \leq i < j \leq N$ and then set $\mathbf{F}_{ji} = -\mathbf{F}_{ij}$, effectively halving the time taken to calculate all the forces.

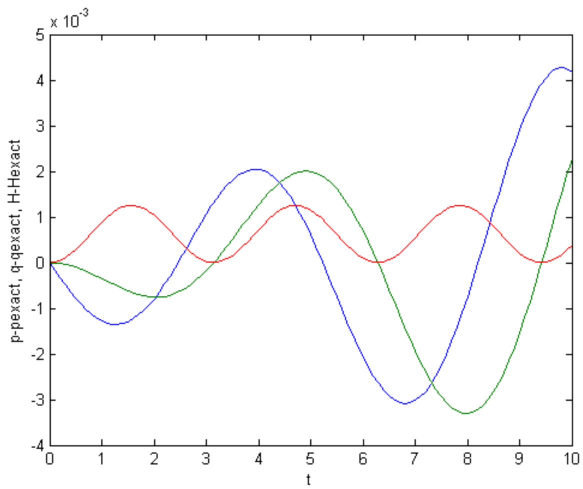
The source for the Fortran implementation is given in in appendix F2, while the data file containing the initial conditions is given in appendix F2 and an example parameters file in appendix F2. The initial conditions are arranged in arrays of velocity and position for each spatial dimension indexed



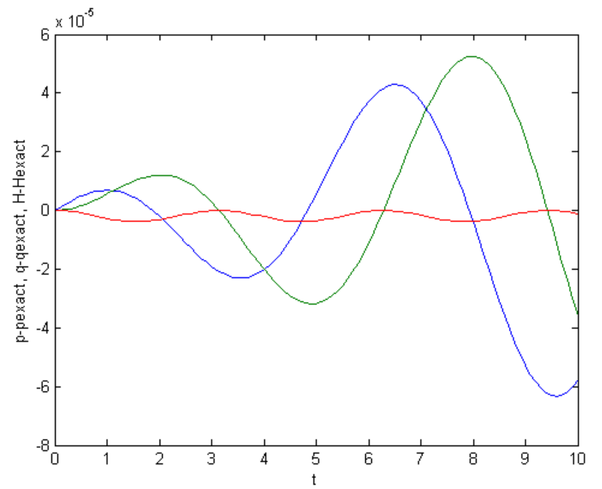
(A) Leapfrog result



(B) Fourth order result



(C) Leapfrog difference



(D) Fourth order difference

FIGURE 1. Results of a simple test of the leapfrog and fourth order routines integrating the simple harmonic oscillator.

by body (Sun = 1, asteroid = 2, Jupiter = 3, Saturn = 4). This may at first seem perverse (more straightforward to have a three-element vector for each body, indexed by the x, y and z dimensions), but it is in fact easier to accommodate an arbitrary number of bodies this way and associate it with the correct mass, given as an array set in the parameters file.

2.5. Interpreting Output in MATLAB

There are two main routines to interpret output data from the integrator: *asteroid_plot* and *asteroid_compare_runs*. Both read position and momentum data from files (along with metadata that contain details such as the time step size, the buffering frequency, whether to look for reversed flow data, etc.) and constructs arrays which contain the osculating orbital elements for each body (although they are not well defined for the Sun, being the primary body), making it possible to plot the orbital elements against time and examine how the orbits change. The latter routine is useful for plotting the differences between two orbits that begin close to one another and determining the rate of divergence between them.

CHAPTER 3

Discussion

3.1. Numerical Error

A consequence of finite-precision arithmetic is numerical error, beyond any approximations inherent in the computational routine itself. A large source of error for the symplectic routines is roundoff. Double precision arithmetic has about sixteen digits of accuracy, so when adding or subtrating a number 10^n smaller than another, n digits of the smaller are truncated. If $n \geq 16$, adding the smaller number is the same as adding nothing at all without resorting to higher levels of precision (and consequently lower speeds if the computational architecture is not built to suit). Even if $n \sim 10$, in double precision, the number may be small enough that its effects can be swamped by numerical error elsewhere in the routine.

An estimation of the error of the leapfrog and fourth order routines follows.

3.1.1. Estimating roundoff in leapfrog. To calculate the new position and momentum at each timestep, the leapfrog routine calculates a middle value of the position as a "stepping stone". In each integration step, the velocity of each body is multiplied by half the time step and added to the position, the total force on each body is multiplied by the time step and added to the momentum, and finally the updated velocity is multiplied by half the time step and added to the position. Roundoff is most likely to occur in the calculation of the force (especially if two bodies become near one another[1]) or in the addition step as position or momentum is updated.

[1]Though should this happen, typically the system will no longer be interesting and the asteroid will no longer be in the main belt. Jupiter and Saturn are almost certainly not going to interact too closely; nor are they likely to make a such close approach to the Sun at any point that this aspect of numerical error should rear its head.

During the calculation of the force, provided the bodies are "reasonable" distances apart, the greatest source of roundoff will come from vastly differing masses. Index the bodies by Sun $= 1$, asteroid $= 2$, Jupiter $= 3$ and Saturn $= 4$. Now $m_1 \sim 1$, $m_2 \sim 10^{-15}$, $m_3 \sim 10^{-3}$, $m_4 \sim 10^{-4}$ and $p_1 \sim 10^{-5}$, $p_2 \sim 10^{-17}$, $p_3 \sim 10^{-5}$, $p_4 \sim 10^{-6}$. Also, $G \sim 10^{-4}$.

Gravity between each body is Newtonian, i.e. $F_g = \frac{Gm_1m_2}{r_{12}^2}$, where r_{12} is the distance between the bodies (relativity is neglected, but is discussed in section 3.2.6). For the Sun-asteroid-Jupiter-Saturn system, approximate average distances are

Asteroid	Jupiter	Saturn	
4	5	10	Sun
	4	10	Asteroid
		10	Jupiter.

Therefore the magnitudes of the forces between bodies are approximately averaged

Asteroid	Jupiter	Saturn	
10^{-20}	10^{-8}	10^{-10}	Sun
	10^{-23}	10^{-25}	Asteroid
		10^{-13}	Jupiter.

Compare these values to momenta and we find that the force between the Sun and the asteroid results in a change of momentum 10^{-15} relative to the Sun's momentum at the prior time step and 10^{-3} relative to the asteroid's momentum if the timestep is of order 1. This means that all but one digit of the Sun-asteroid force is truncated when added to the Sun's momentum per the calculations, but only three are digits truncated when the force is applied to the asteroid. The following array is generated showing how much truncation takes place:

Sun	Asteroid	Jupiter	Saturn	
0	15	3	5	Sun
3	0	6	8	Asteroid
3	18	0	8	Jupiter
4	20	7	0	Saturn

This unfortunately means that there will be times during the asteroid's orbits where it effectively exerts no force on Jupiter or on Saturn if its mass is too low or the timestep too small, as force $\times$ time step results in a relative change of momentum smaller than machine precision.

Because it computes more substeps per timestep, the fourth order routine potentially suffers more roundoff error, as its integration coefficients are smaller for each substep.

Roundoff could be reduced by increasing the mass of the asteroid (which is justification in itself to do so). However, the total mass of the asteroid belt is $\sim 10^{-9}$ Earth masses, with $\sim 80\%$ of that mass contained in Ceres, Pallas and Vesta, the three largest asteroids ([**19**]), leading to the decision to use such a small mass.

3.1.2. Tradeoff between energy and angular momentum conservation. A dichotomy exists between the desire to minimise error in the energy (as the true system evolves on surfaces of constant H - achieved by using a smaller timestep) and minimising roundoff error (increasing timestep, for a given mass of the asteroid). Exact conservation of angular momentum is proved for the leapfrog algorithm in appendix D, so any variation from the initial value during a run is due to numerical error only.

Table 1 shows values for variation in energy and angular momentum for eight runs for two different time steps and two different initial mean motion ratios with Jupiter (the ~ 2.8 value is far enough from any Kirkwood gap not to experience any major resonance phenomena). There is a clear difference in performance regarding energy conservation between the second and fourth order routines, though there is little appreciable difference in angular momentum conservation for a given time step, though the runs with the timestep smaller by a factor of 100 show a corresponding increase by a factor of 100 in angular momentum error - roughly linear growth in the error is observed when plotted against time - as they had to integrate 100 times as many time steps for the given length of time.

It is also worth noting that the smaller time steps improve energy conservation by at least two and up to four orders of magnitude. Moreover, with the smaller timestep, energy conservation differs between the routines by merely a factor of five, whereas with the larger timestep the fourth order routine clearly outperforms the second order: it is one hundred times better.

τ	$\frac{n_a}{n_j}$	Method order	$\frac{\Delta H}{\langle H\rangle}$	$\frac{\Delta h}{\langle h\rangle}$
43.31572	3.0	2	4.2237×10^{-5}	9.2900×10^{-11}
		4	4.9482×10^{-7}	7.5109×10^{-11}
	2.846542263	2	4.2237×10^{-5}	5.6950×10^{-11}
		4	4.9474×10^{-7}	5.7681×10^{-11}
0.4331572	3.0	2	5.5113×10^{-9}	3.9557×10^{-9}
		4	8.7233×10^{-10}	4.0091×10^{-9}
	2.846542263	2	5.2549×10^{-9}	3.8633×10^{-9}
		4	1.1982×10^{-9}	3.8752×10^{-9}

TABLE 1. Maximum relative variation in energy (H) and angular momentum (h) over one megayear for two different step sizes and two different initial mean motions for each integration method.

3.1.2.1. *Observation on variation in the angular momentum error.* A common feature of each run has been a more or less linear growth in variations in the angular momentum about a mean value, close to the initial value. Of particular interest (and a source of some consternation, as it is unexpected[2]) was the fact that in runs which reversed the flow to test the accuracy of the integrator (finding how close the system returned to its original state) the variance of $\mathbf{h}$ converged to nearly zero as time returned to zero.

However, the nonzero momentum results in secular growth of each component of $\mathbf{q}$ (with oscillatory variations about the centre of mass of the system, which moves with a speed of approximately 6.5×10^{-6} AU per day when the system consists of the Sun, an asteroid, Jupiter and Saturn). This is an obvious source of truncation error as the integration continues for long times; the system can travel tens or hundreds of thousands of AU in millions of years, while the velocities remain of order 10^{-1} or smaller.

The size of the truncation error grows linearly with time, but bias in the algorithm may determine how close the mean angular momentum over long time will remain to the angular momentum at $t = 0$. In fact, for some initial conditions and time steps there is a bias to increase the angular momentum, while in some it will decrease, and a few show almost no bias at all.

[2]Error normally grows with the number of steps integrated. Thus even as time for the system is essentially going backwards, the integration is still in a practical sense going forward.

A further observation is that if the flow is reversed, the system will tend (approximately, with the accumulated errors) towards the origin, so truncation will fall off as the positions get smaller, and only the sum of any bias will remain as error in h. Figure 1 shows a plot of the angular momentum as it evolves forward (blue) and backward (red) in time. The difference between the two values at $t = 0$ is of order 10^{-10} and represents the sum of the error over both branches of the integration.

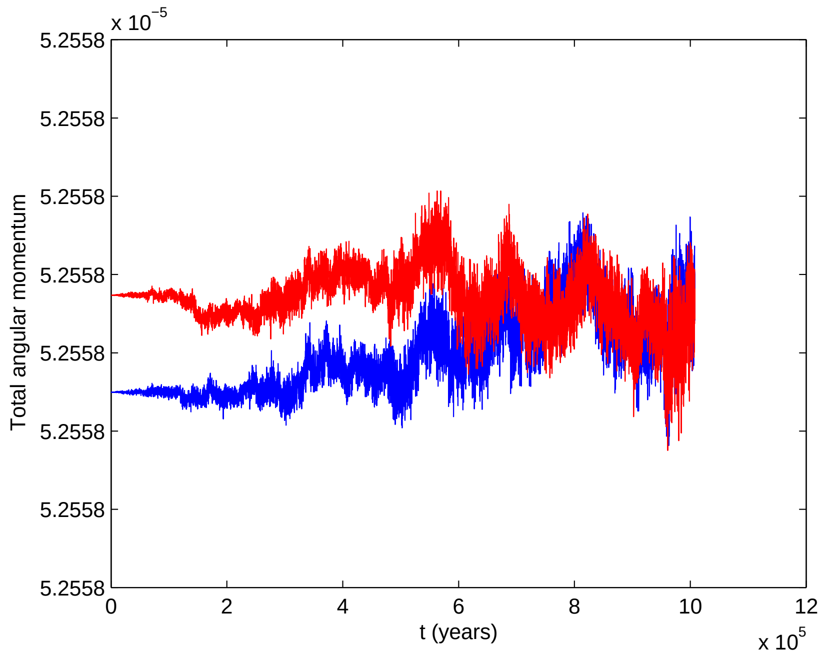
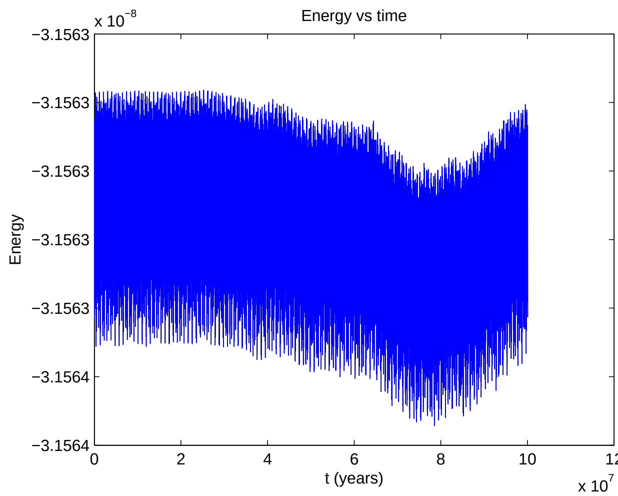


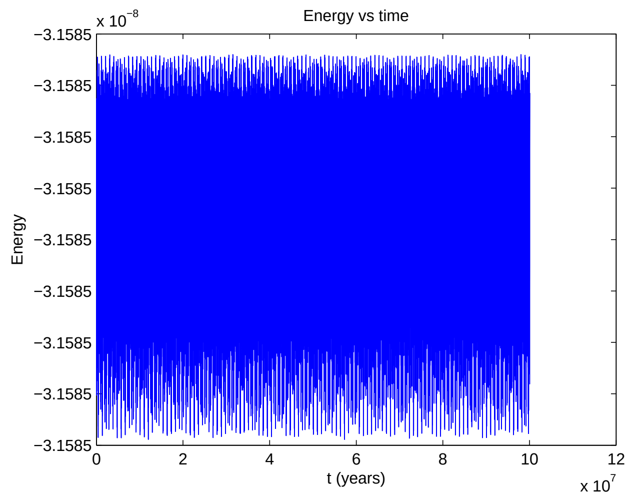
FIGURE 1. An example of the increasing variance of angular momentum as the system moves away from the origin (blue) and how the variance decreases again as $t \to 0$ (red).

This source of numerical error can be controlled by subtracting the drift velocity from the velocity of each of the bodies, though this explanation for the anomaly was unfortunately not discovered until too late in the day to redo most of the runs. Some comparisons were possible, however, between long-time runs with and without this correction, in order to ascertain its effect on the dynamics of the system in the long term; that is, whether the results might still be true to the real system. For the majority of runs, the relative error remained smaller than 10^{-8}, which is hopefully small enough that the dynamics are not far off.

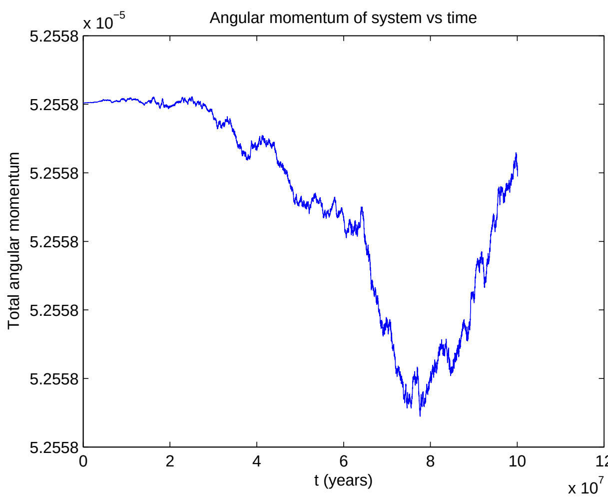
When the drift was corrected, relative angular momentum deviations remained within 10^{-12}, with no apparent growth over long time. Thankfully, resonant and non-resonant regions behave similarly to runs without the correction. However, as deviations in the angular momentum become large (around 10^{-7} relative error - which takes on the order of 10^{7} years) the energy is seen to no longer remain bounded but instead its mean value follows a curve of the same shape as the the angular momentum's evolution in time, as shown in figure 2. Clearly, if the angular momentum error becomes too large the energy will drift a long way from the original value and it may be that the trajectory will cross a separatrix in the true phase space that otherwise it wouldn't.



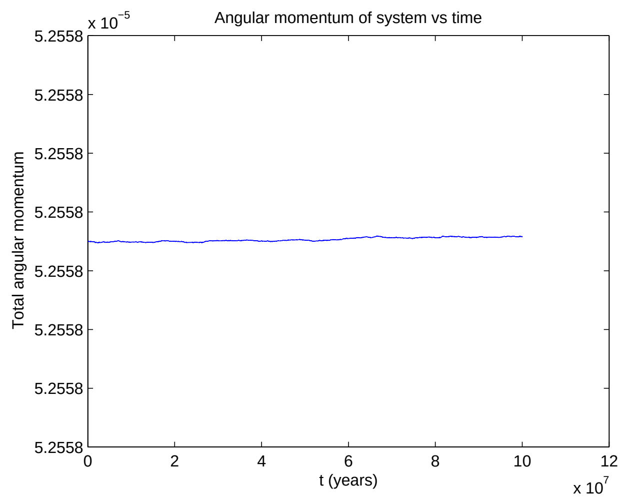
(A)



(B)



(C)



(D)

FIGURE 2. Comparison of: (A) and (B) energy; (C) and (D) angular momentum. The former of each pair is for the system with drift included, while the initial momentum is neutralised in the latter. Both runs started with the asteroid at perihelion, directly opposite Jupiter's IC, initial eccentricity $e = 0.15$, inital mean motion ratio with Jupiter being $\frac{n_{ast}}{n_{jup}} = 1.666666666666667$ and $\tau = 43.31572$ and ran for 100 Myears. The amplitude of the oscillations in energy at any time are the same in both plots, but it is clear that the accumulated errors in the angular momentum can significantly alter the energy if the system drifts too far from the origin.

3.2. Neglected Influences

3.2.1. Inner planets. The masses of the inner planets are of order 10^{-7} to 10^{-6} $M_\odot$ at most. The average force between the asteroid and Earth (as it is of a moderate mean distance from the asteroid and the most massive of the inner planets at 3.0034901×10^{-6} $M_\odot$) will be of order 10^{-26}, resulting in the truncation of 10 digits in calculation of the change in the asteroid's momentum, with a step size of order 1. While not negligible (and Mars will certainly approach close enough to the asteroid to have a significant effect on inner-belt asteroids, closer in than the 3:1 Kirkwood gap), the inner planets' combined mass is added to the sun and their orbital perturbations ignored. This is because the focus of this study is on the resonant effects of Jupiter and Saturn, the former of whose gravitational effects on the asteroid are only swamped by the Sun itself.

3.2.2. Arbitrary ICs for asteroid. The number of possible initial conditions for asteroids in the main belt is vast. The main asteroid belt has semi-major axes ranging from 2.1 to 3.3 AU and eccentricities concentrated between 0.05 and 0.35 (cite MPC), with the peak near 0.15. Inclinations range from $0\,^\circ$ to over 40°, though the bulk of the asteroids have inclinations less than 20° (though an interesting cluster exist between 20° and 30°, representing several families that exist at high eccentricities between several Kirkwood gaps, exhibiting quite distinct structure in a vs i scatter plots, as seen in figure 3). Arguments of perihelion are approximately evenly distributed around the circle. In other words, the space of possible initial conditions is prohibitively large.

Eccentricity and semi-major axis are chosen to be arbitrary, as semi-major axis determines orbital period (and thus resonance), and eccentricity is important to choose arbitrarily, as it plays an important role in resonant dynamics. Figure 4 shows the extent of the asteroid belt in its semi-major axis and its eccentricity.

By restricting the argument of perihelion to be opposite Jupiter's initial condition (which is not at that point at its node of perihelion), however, the dynamics may not be appreciably biased. Simulations show that the arguments of perihelion precess for both the asteroid and Jupiter (as expected for the ($n > 2$)-body problem), and the asteroid at a much greater rate than Jupiter. Therefore it is arguable that the initial argument of perihelion has little influence in determining the secular dynamics of the asteroid, as the histogram in figure 5 shows.

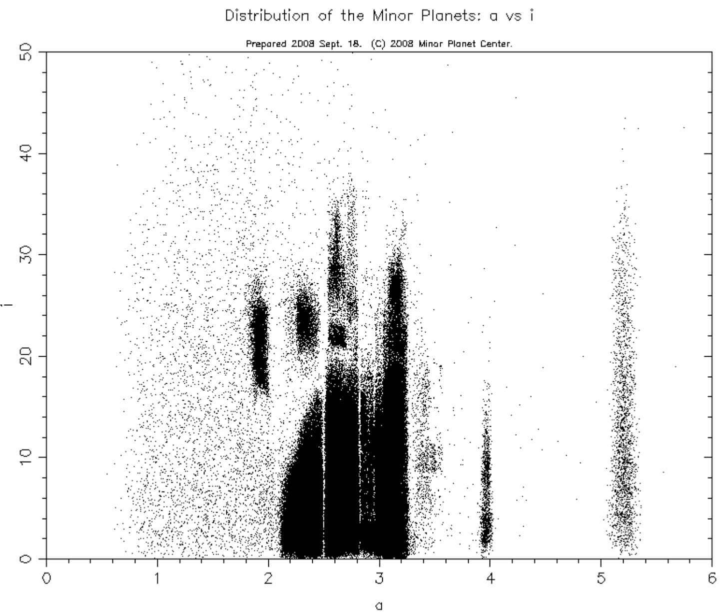


FIGURE 3. Scatter plot of inclination i vs. semi-major axis a, clearly showing the Kirkwood gaps and several major families of asteroids at high inclinations, distinctly above the main body of asteroids. Image courtesy of the MPC: *http://www.cfa.harvard.edu/iau/lists/MPDistribution.html*

The only orbital elements of concern, then, are inclination and, closely associated with it, the ascending node. Observations show there are actually very few asteroids in the plane of the ecliptic ($0°$ inclination), but numbers rise sharply with inclination to tens of thousands of observed asteroids (with likely many more unknown) with less than $5°$ inclination, with peak numbers in a small interval just below $4°$. Simulations show inclination tends to vary slightly either side of Jupiter's mean, while some (relatively few out of the sample of integrations) show large excursions of inclination up to $20°$ either side of Jupiter's. These orbits in particular tend to coincide with the major Kirkwood gaps and very fast chaotic divergence of nearby trajectories. What we can conclude is that a more complete numerical survey should include the ability to arbitrarily choose inclination.

While most of the planets have longitudes of ascending node between $70°$ and $130°$, the asteroids show a distinct pattern in the distribution of their ascending nodes, shown in figure 6, almost certainly associated with Jupiter's argument of perihelion ($275.066°$) and its ascending node ($100.492°$), suggesting (along with numerical results) that asteroids are actively perturbed

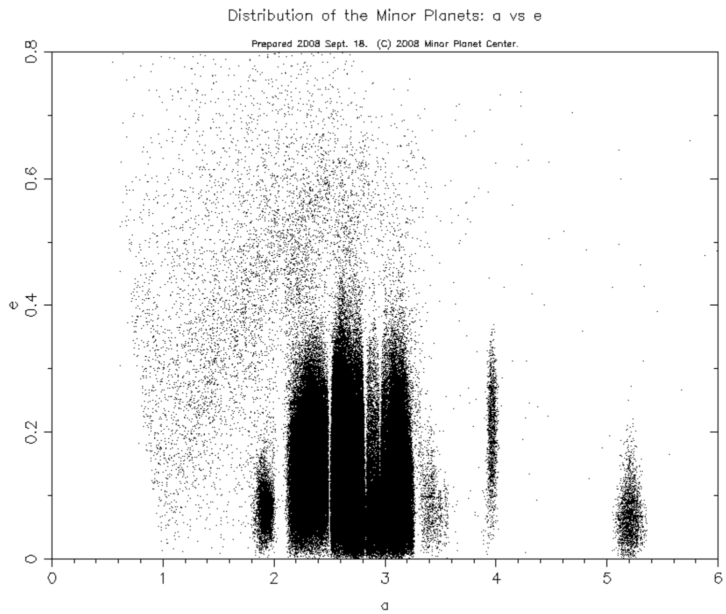


FIGURE 4. Scatter plot of eccentricity e vs. semi-major axis a, including the Trojans and the Greeks in a 1:1 resonance with Jupiter, which orbit near its L_4 and L_5 Lagrange points. Image courtesy of the MPC: *http://www.cfa.harvard.edu/iau/lists/MPDistribution.html*

out of Jupiter's exact plane of orbit, even if they instead oscillate around it. This suggests it may also be worthwhile to be able to arbitrarily choose the argument of ascending node, as simulations show that the asteroid closely follows (albeit with greater, librating amplitude) the argument of ascending node of Jupiter.

3.2.3. Tidal forces and viscous fluid effects. Most numerical studies of the Solar System involve modelling the planets as point bodies, rather than as the extended objects they actually are. In addition to this, the most important bodies (in terms of gravitational presence) are not even rigid; the Sun is a ball of fluid plasma, Jupiter, Saturn, Uranus and Neptune are gaseous and fluid. Even the earth has a liquid core, which affects its dynamics. Extended bodies do not just experience the gravitational force as a vector pulling bodies together, but tidal forces act to deform bodies by squeezing them inwards in the plane perpendicular to the gravitational force itself and outwards on the vector of the force. It is for this reason that we have tides in our oceans, and because of fluid friction of the oceans against

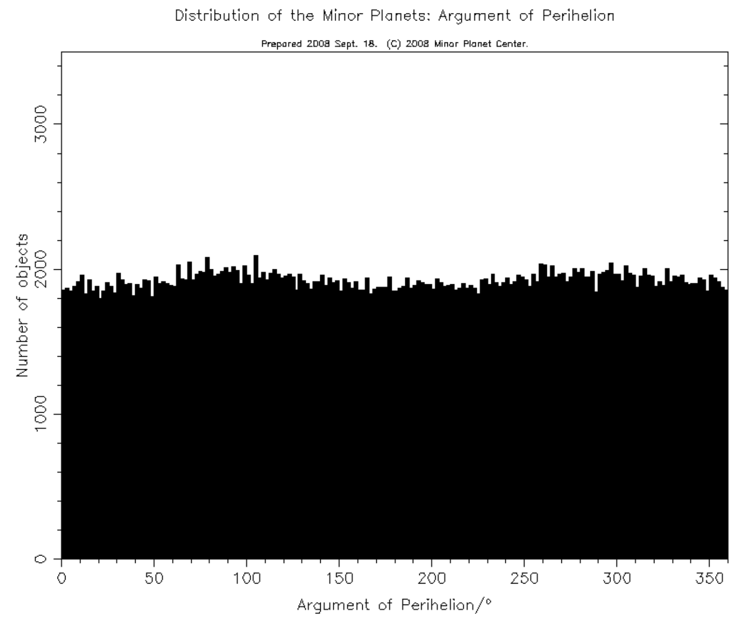


FIGURE 5. Histogram of asteroids by argument of perihelion. Note the approximately flat distribution. Image courtesy of the MPC: *http://www.cfa.harvard.edu/iau/lists/MPDistribution.html*

Earth's continents the Earth's days are getting longer and the moon is receding. Fluid viscosity also tends to be a stabilising factor in dynamical systems, so could tidal forces and viscous dissipation be having an effect on the dynamics of the asteroids?

In reality, this is almost certainly the case. However; if this effect is too small it is negligible due to truncation. Tidal forces go as $\frac{1}{R^3}$, where R is the distance between the bodies, therefore falling off much more quickly than the attractive force between them. Further, it is proportional to the product of the masses and the radius of the body in question. A simple estimation yields this force to be on the order of 10^{-30} between the asteroid and Jupiter for an asteroid assumed to be several kilometres in diameter (10^{-7} AU). It is safe to say without further estimations that the effect of viscosity on the dynamics of the asteroid is even smaller than the limit of double precision arithmetic.

3.2.4. Aspherical bodies. A common simplification often found in studies of the Solar System is that every body is assumed to have a spherical

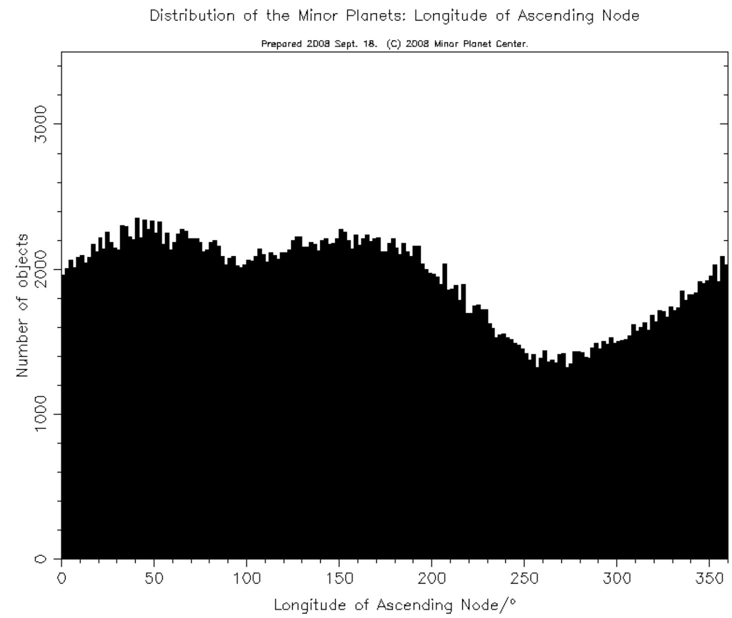


FIGURE 6. Histogram of asteroids by longitude of ascending node. Note the dips around $100°$, near Jupiter's ascending node, and $275°$, near Jupiter's argument of perihelion. Image courtesy of the MPC: *http://www.cfa.harvard.edu/iau/lists/MPDistribution.html*

gravitational potential. The justifications for maintaining this simplification are: (a) convention and simplicity in the context of the scope of the project; and (b) most of the interesting motion happens in or near a single plane of motion and all the bodies (except possibly the asteroid) are "pretty close" to spherical.

3.2.5. Loss of mass from the Sun through radiation. Solar mass is estimated to be lost at a rate of five million tonnes per second. This is equivalent to approximately 2×10^{-16} solar masses per day, a negligible amount per time step.

3.2.6. Relativity. The article by Benito & Gallardo [**20**] discusses numerical simulations of relativistic versus classical models of the solar system. Their finding is that relativistic effects from the Sun play an important role in the secular dynamics of the inner planets. The relativistic correction factor to the acceleration due to the Sun used in [**20**] was proposed in 1975 by Anderson et al. [**21**] is

$$\Delta\ddot{\mathbf{r}} = \frac{G\mathrm{M}_\odot}{r^3c^2}\left[\left(\frac{4G\mathrm{M}_\odot}{r} - \mathbf{v}^2\right)\mathbf{r} + 4(\mathbf{v}\cdot\mathbf{r})\mathbf{v}\right].$$

In units of AU per day, the speed of light is approximately 173.1446, and substituting a typical distance $r = 2.25$ and speed $v = 0.0120$, $\mathrm{M}_\odot = 1$ and $G = 2.95912208286$, the correction is $\Delta\ddot{\mathbf{r}} = -8.8198e - 012$. This is not below the limit of precision, but certainly negligible for short integrations and prone to strong truncation using only double precision arithmetic. [**20**] shows that over the course of megayears relativistic effects may change the dynamics of an asteroid's orbit, but it is simply beyond the scope of this project to incorporate relativity.

3.2.7. Outer planets beyond Saturn. Although it is certain that perturbations from Uranus and Neptune contribute to the asteroids' dynamics in reality, it follows from the estimations in section 3.1.1 that truncation and accuracy will be a huge problem when updating the momentum based on the forces between the massive outer bodies and the minuscule asteroid.

3.3. Continuing Discussion: Drift of the Solar System

Using the initial conditions for the outer planets and the Sun given in Hairer, Lubich and Wanner [**18**], it is observed that the centre of mass of the system moves with constant velocity. Indeed, on checking, the initial momentum of the system is nonzero, and as expected the total momentum remains constant in time (barring numerical errors which mirror those observed in the angular momentum). As discussed in section 3.1.2.1, this was responsible for a great deal of roundoff error later in most runs. For runs less than about 10 megayears, direct comparison shows this error does not appear to affect the particular dynamics for any set of initial conditions.

Even out to 50 megayears, keeping in mind the chaotic divergence of trajectories and perturbations from numerical error in the drifting system, the way that the orbital elements evolve (regular or chaotic variations) is similar in both systems. Figures 1 through 11 in Appendix E illustrate this for several resonant and nonresonant orbits.

3.4. Desired Integrations

In order to get some idea of the chaotic structure of the asteroid belt, it is necessary to do a large number of integrations from a large sample of initial mean motions and eccentricities. Ideally it would be possible to do enough runs from enough initial conditions and account for Mars crossings do statistical calculations on the numerically determined asteroid "belt" (since only one asteroid's orbit is evaluated at a time) comparing its structure to that of the real asteroid belt.

Given the restrictions imposed, however, we will do a series of short (1 Myear) integrations for initial Jovian mean motion resonances ranging from 4.3 to 1.2 in increments of 0.1 and initial eccentricities of 0.05, 0.15, 0.25 and 0.35, both with and without Saturn. This will offer some idea of how much both direct and indirect (through modifications to Jupiter's orbit) perturbations from Saturn affect the asteroids' orbits, though it is not a fine enough sample space to cover many of the Kirkwood gap resonances. Longer integrations will examine the long term behaviour of the asteroid in various resonant and nonresonant orbits, while other runs will examine what happens to orbits with nearby starting conditions both with and without Saturn.

Runs must also be conducted so that error can be tested; to find out how badly numerical error affects the reversibility of the system (and thus projections for its accuracy in general). Unfortunately many runs were completed with the roundoff-inducing drift in place and not enough time remained to redo these integrations with the initial momentum compensated for. The vast majority of these runs, however, were short (~ 1 Myear), and, as discussed, comparisons show the dynamics are not significantly altered over such time spans.

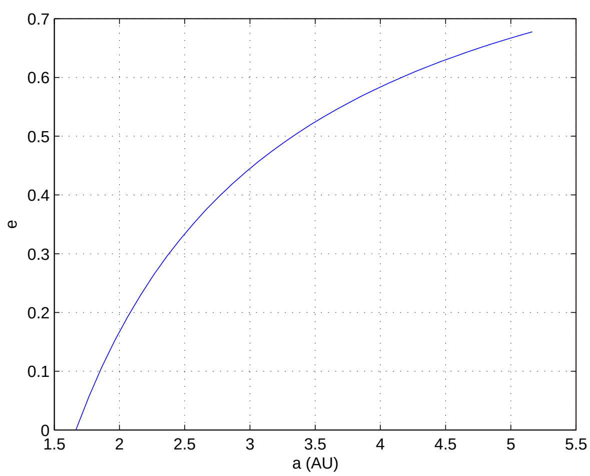
3.4.1. Expected results. Integrating the full equations of motion for the system of the Sun, an asteroid, Jupiter and Saturn should result in chaotic motion with short Lyapunov times ($\sim 10^5$ years) in regions like the 3:1 resonance, exhibiting periods of seemingly regular evolution of the orbital elements interspersed with periods where the eccentricity rises into a region where an interaction with Mars or Jupiter is probable.

In nonresonant orbits, linear or polynomial divergence of nearby trajectories is expected, with entirely regular orbits; showing only regular variation in the orbital elements.

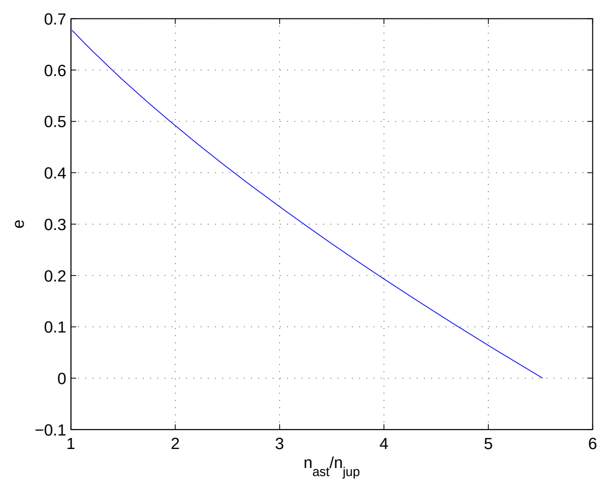
When Saturn is excluded from the integrations, Jupiter's orbit will be almost Keplerian: the force exterted on it by the asteroid will result in only a tiny change in Jupiter's momentum, so precession will be near (if not) negligible. On the other hand, the asteroid will still experience perturbations from Jupiter, but these perturbations will be more constant in strength, due to its orbital elements changing less in time. Thus is is expected that asteroids in resonant orbits will experience weaker chaos or remain bounded in smaller pockets of chaotic motion (which may or may not lead to ejection from the resonance).

Meanwhile, orbits away from Kirkwood gaps are expected (naïvely, from knowledge of the actual distribution of asteroids in the asteroid belt) to be similarly less perturbed and more stable, even over very long time spans.

3.4.2. Mars/Jupiter Interaction Thresholds. Figure 7 shows the threshold line for an asteroid of a given semi-major axis (7a) or mean motion ratio (7b) to become a Mars crosser - that is, to have eccentricity large enough that part of its perihelion distance within the orbit of Mars (which has semi-major axis 1.52 AU and eccentricity 0.093).



(A) Eccentricity threshold by semi-major axis.

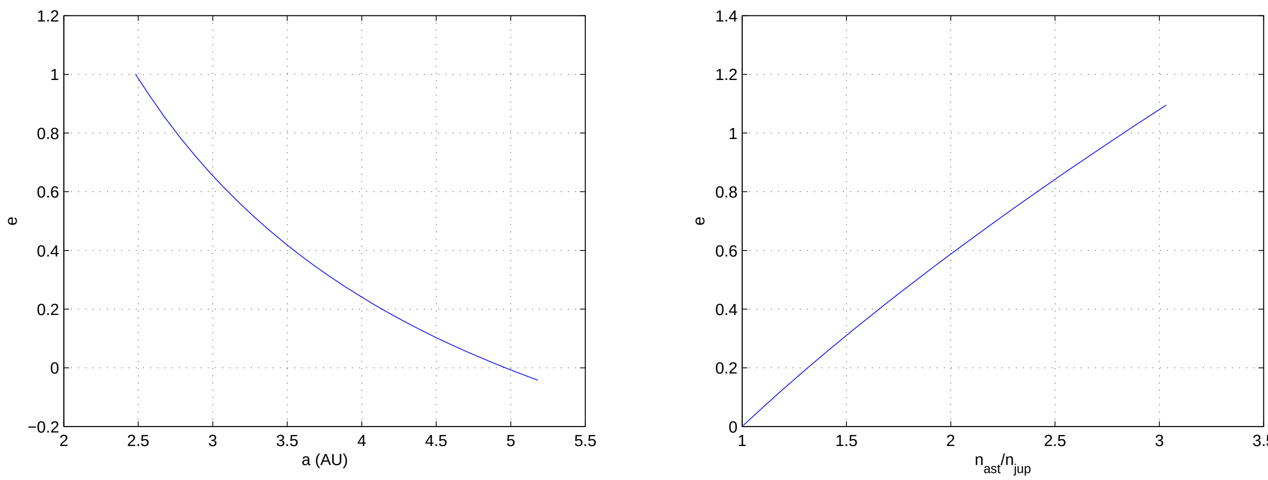


(B) Eccentricity threshold by Jovian mean motion ratio.

FIGURE 7. The eccentricity threshold, above which an asteroid will become a Mars crosser and will probably be ejected from its resonance by direct perturbations from Mars.

Similarly, if the aphelion distance of the asteroid becomes far enough out, there is the possibility of a large perturbation from Jupiter. Figure 8 shows the threshold eccentricity at which the aphelion distance of the asteroid's orbit will pass beyond Jupiter's perihelion distance. Depending on the nature of the resonance, close approaches with Jupiter may or may not be possible (but there are more orbits in total where they are), and approaches need not be as close to Jupiter to have the same effect as an approach to Mars.

Perihelion distance can be determined from the orbital elements by $a(1-e)$. Similarly, aphelion distance is given by $a(1+e)$. These simple relationships come from the geometry of an ellipse with one focus at the origin.



(A) Eccentricity threshold by semi-major axis

(B) Eccentricity threshold by Jovian mean motion ratio

FIGURE 8. The eccentricity threshold, above which an asteroid's aphelion distance will exceed Jupiter's perihelion distance and become likely to be swept out by Jupiter's large gravity.

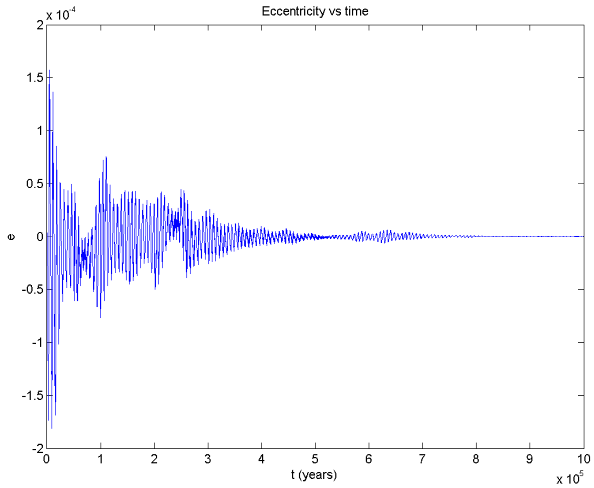
τ	Method order	Drift	$\lvert\mathbf{q}_a(0) - \mathbf{q}_{a,r}(0)\rvert$	$\lvert\mathbf{q}_j(0) - \mathbf{q}_{j,r}(0)\rvert$
10.00	2	y	2.5050503×10^{-2}	3.2386714×10^{-4}
		n	1.0236731×10^{-4}	7.1825833×10^{-6}
	4	y	8.0377675×10^{-3}	2.1732859×10^{-3}
		n	5.4588679×10^{-6}	5.9034100×10^{-6}
1.00	2	y	2.4001855×10^{-3}	1.9625345×10^{-3}
		n	3.2216039×10^{-6}	2.5399603×10^{-6}
	4	y	2.3510936×10^{-3}	9.4165100×10^{-4}
		n	1.9090611×10^{-5}	2.1089969×10^{-5}

TABLE 2. Accuracy of the two algorithms for step sizes $\tau = 1.00$ and $\tau = 10.00$ both including and excluding drift induced by nonzero initial momentum in terms of how close the system returns to its initial configuration when the flow is reversed. $\mathbf{q}_{i,r}(t)$ denotes the position of body i at time t for the reversed flow. Only Jupiter and the asteroid are considered; the other larger bodies are roughly comparable to Jupiter for the sake of judging accuracy.

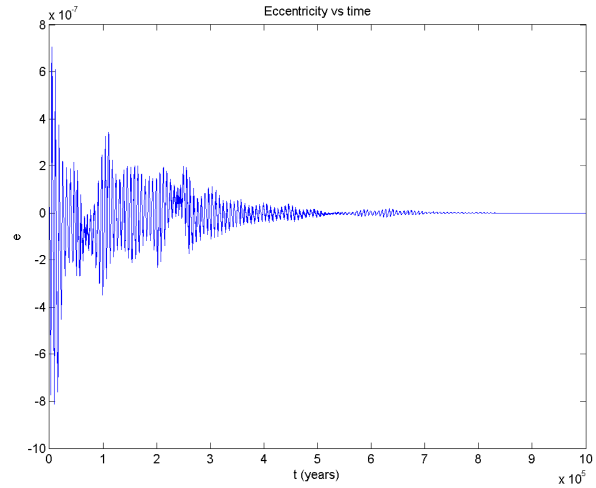
3.4.3. Error-testing runs. Several runs were conducted wither their flow reversed after they reached 1 Myear. This distance between their final position and their initial position indicates the total of both the accuracy of the integration routine and the accumulated effect of roundoff over effectively 2 Myears (complicated in the case where the system drifts because the roundoff reduces again as $t \to 0$ and the system tends back to the origin).

The runs illustrated here all have initial conditions $e = 0.35$ and $\frac{n_{ast}}{n_{jup}} = 2.00$ and step sizes of $\tau = 1.00$ and $\tau = 10.00$. Note that under normal circumstances this orbit would start already as a Mars crosser, but this is less relevant as here all we want is to sample the accuracy.

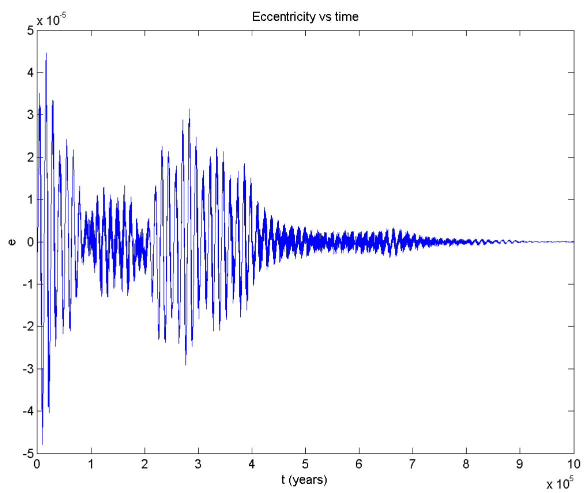
Table 2 shows the differences between positions for Jupiter and the asteroid at $t = 0$ between the initial condition and the "final" value of the reversed run. There is a clear advantage when the initial momentum is neutralised, generally an improvement of 2-3 orders of magnitude. Step size also plays a role: when τ is smaller the leapfrog routine tends to perform better than the fourth order routine and vice versa.



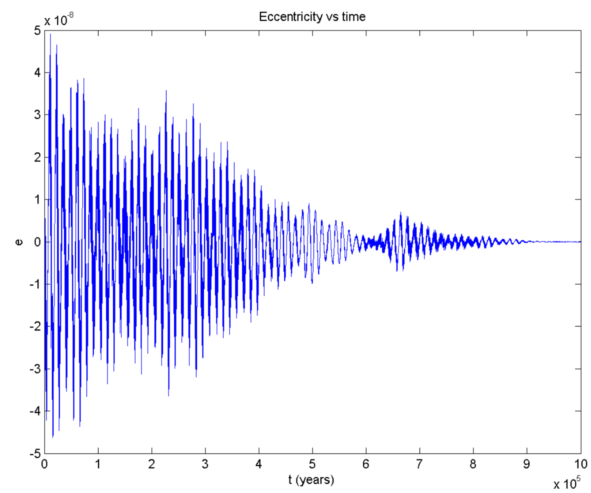
(A) $\tau = 10$, second order, drift.



(B) $\tau = 10$, second order, no drift.



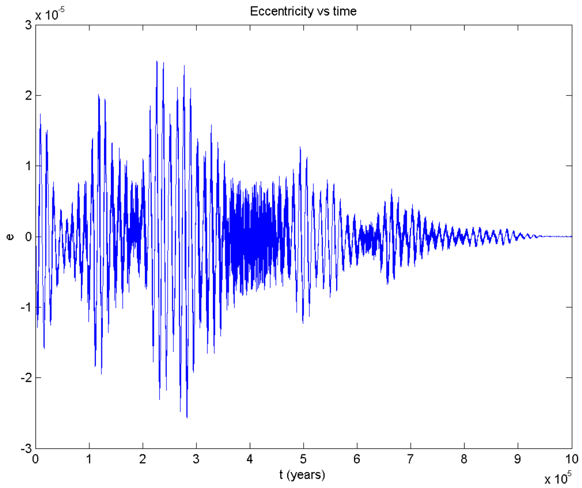
(C) $\tau = 10$, fourth order, drift.



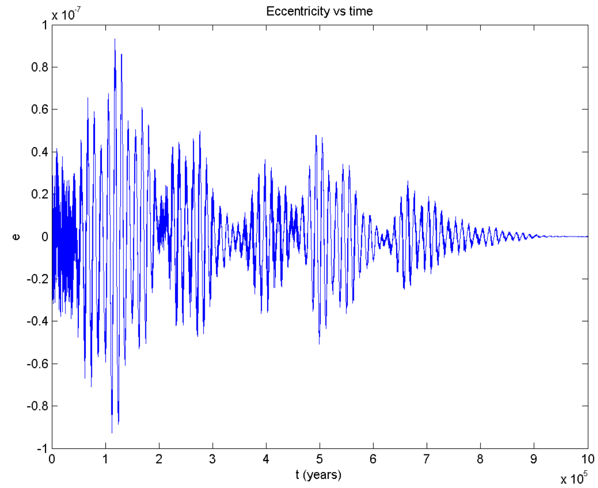
(D) $\tau = 10$, fourth order, no drift.

FIGURE 9. Difference in asteroidal eccentricity between forward and reversed flows for $\tau = 10$. Initial conditions are identical, given in paragraph 2 of section 3.4.3.

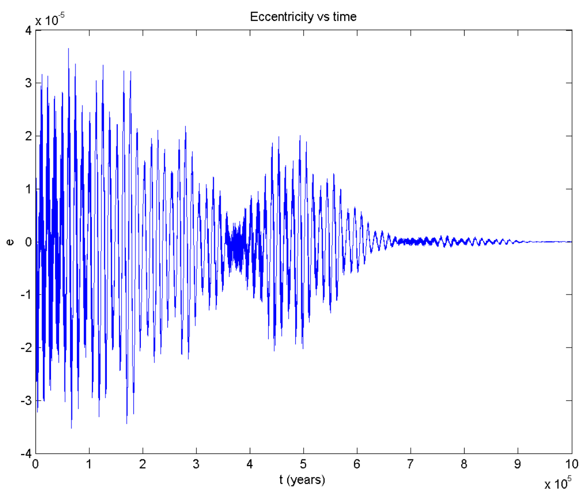
Figures 9 and 10 show the divergence in eccentricity of the asteroid for the same set of runs as in table 2. Note that the divergence appears polynomial for this resonance over this time scale.



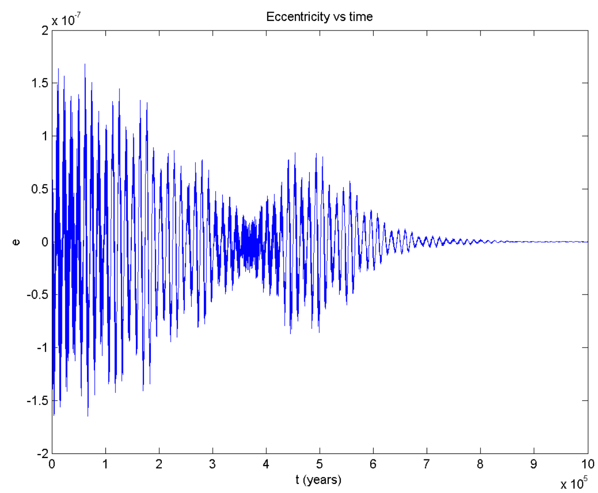
(A) $\tau = 1$, second order, drift.



(B) $\tau = 1$, second order, no drift.



(C) $\tau = 1$, fourth order, drift.



(D) $\tau = 1$, fourth order, no drift.

FIGURE 10. Difference in asteroidal eccentricity between forward and reversed flows for $\tau = 1$. Initial conditions are identical, given in paragraph 2 of section 3.4.3.

3.4.4. Megayear Runs. Runs integrated for one megayear were started at grid points over the two dimensional initial condition space as oulined above, both with and without Saturn. Each run is named by its initial condition: “e15n33” is the run starting with $e = 0.15$ and $\frac{n_{ast}}{n_{jup}} = 3.3$. A prefix “ns” indicates the run neglected Saturn. These runs were begun before the analysis of error regarding step size and the order of the routines was completed, so for consistency's sake they were continued with the same step size ($\tau = 1.0$ days) with the fourth order algorithm. A prefix "ls" indicates a larger step size of $\tau = 43.31572$ days[3] was taken, as the shorter integration time with this step size made it feasable to do a set to compare the dynamics between the step sizes across a broad spectrum of the initial condition space, though this was not possible for the runs without Saturn.

While there are too many megayear runs (84 with Saturn and 84 without) to present all of them, some results of particular interest emerge. Results when Saturn is neglected will be discussed after the 4-body results.

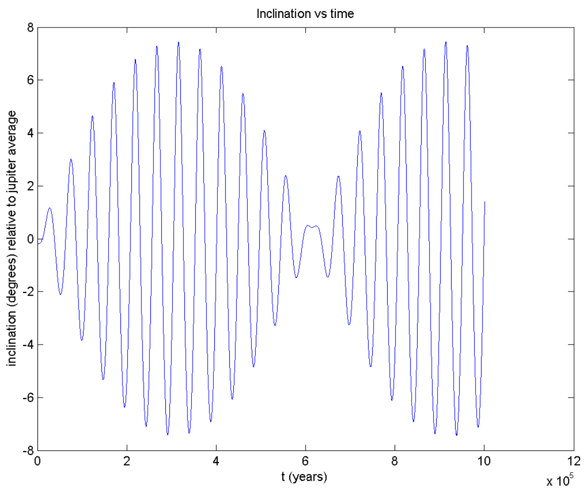
3.4.4.1. *Mean motion ratio* $4.0 \leq \frac{n_{ast}}{n_{jup}} \leq 4.3$. This region is closer to the Sun than what is largely considered the inner edge of the main asteroid belt (the border is often given to be at the 4:1 resonance, but sometimes the 5:1 resonance is considered the inner boundary of the whole asteroid belt). Semi-major axes range from approximately 1.97 to 2.01 AU, implying that the eccentricities for such small orbits must remain less than about 0.15 to 0.19 to avoid crossing Mars' orbit.

Common features of the orbits with Saturn were that the inclinations would vary periodically on a time scale of approximately $25,000$ years inside an envelope with a much longer period, seeming to depend on both the mean motion ratio and the initial eccentricity, as shown in figure 11. Note that the variations in inclination are regular.

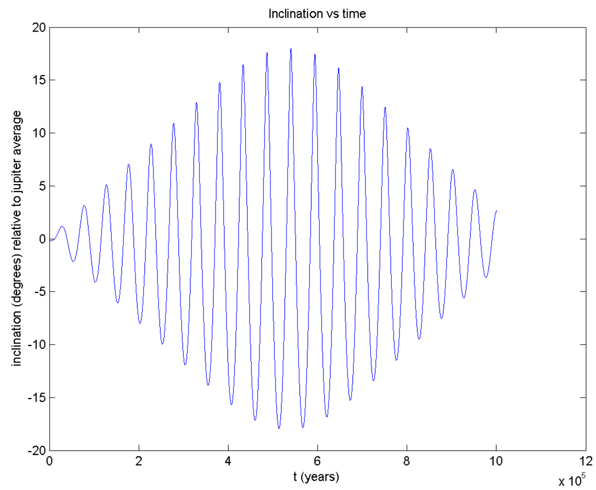
The same runs computed with the smaller step size behave mostly the same, but with a striking difference at the 4:1 resonance: the eccentricity varies irregularly for about $200,000$ years in the e05n40 run and then spikes upwards past the Mars-crossing threshold and finally passes the 0.8 threshold just before $500,000$ years. It reaches the $e = 0.8$ threshold even more quickly when the initial eccentricity is higher. Contrast this with the eccentricity calculated for the corresponding run with the larger step size, as in figure 12, which remains stable for the duration of the integration for all initial eccentricities.

Which dynamic is correct? Figure 13 shows the evolution of both the energy and total angular momentum of the system with time. The energy error in run e05n40 is comparable to its angular momentum error, three orders of

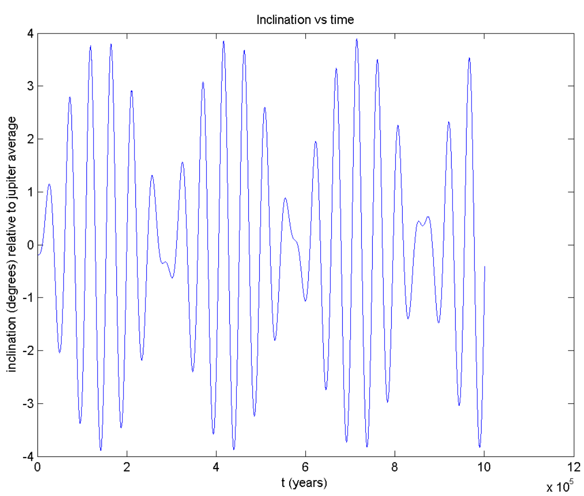
[3]This step size is chosen because it is fractional to Jupiter's orbital period $T_j = 4331.572$ days.



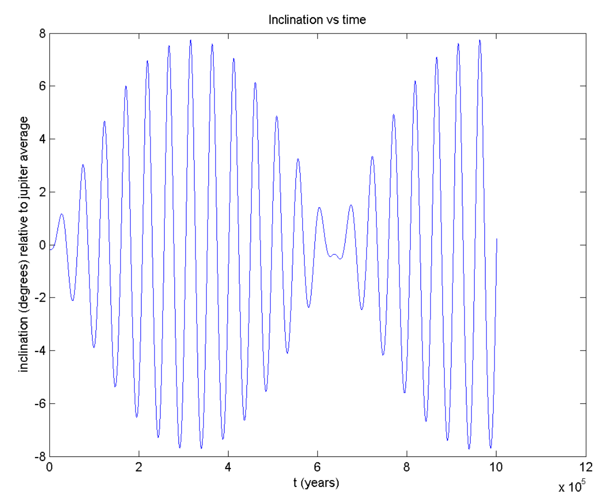
(A) lse05n40.



(B) lse05n42.



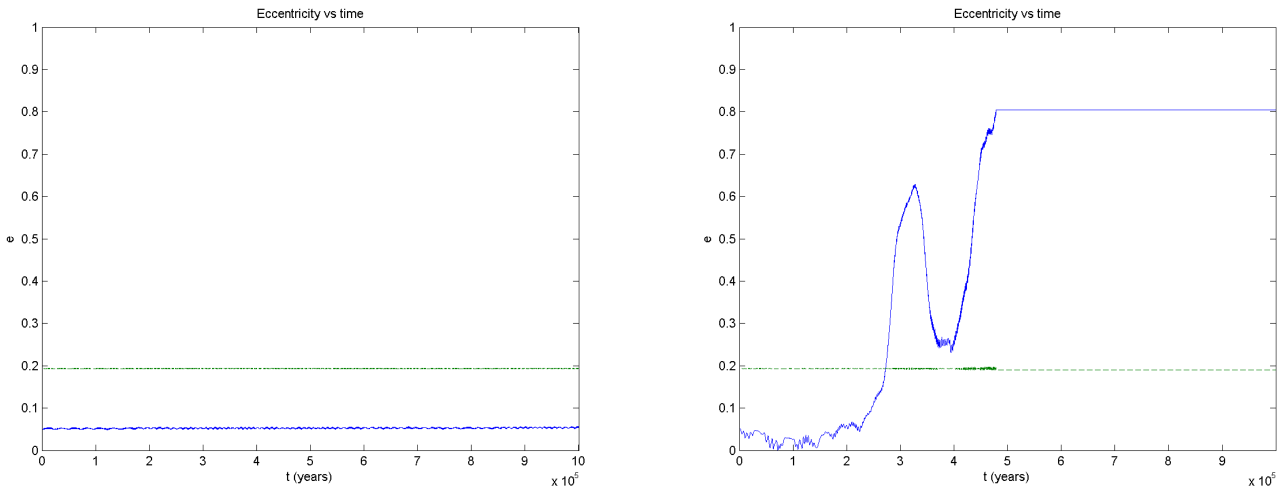
(C) lse15n40.



(D) lse15n42.

FIGURE 11. Inclinations for a sample of initial conditions in the $4.0 \leq \frac{n_{ast}}{n_{jup}} \leq 4.3$ range. The motion appears regular, even at the 4:1 resonance ((A) and (C)).

magnitude smaller than the energy error in run lse05n30, though the angular momentum error in that run is comparable (if slightly smaller). Given that the energy error is smaller (at least over this time scale), e05n40 may be the more accurate; it is possible that the 4:1 resonance exists close to some separatrix in phase space, allowing lse05n40 to cross into a pocket of regular



(A) lse05n40. (B) e05n40.

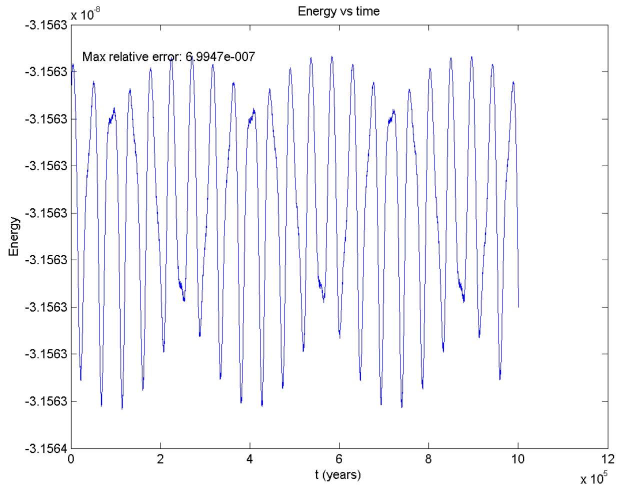
FIGURE 12. Eccentricities for runs with identical initial conditions but different step sizes at the 4:1 resonance. The green dotted line represents the Mars-crossing threshold.

motion, or the smaller step size produces a modified Hamiltonian that has a different phase space structure to the larger step size.

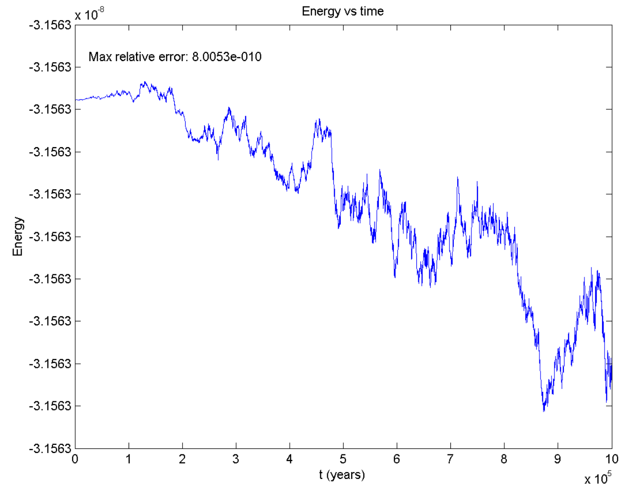
Extra runs for this were computed with the same step sizes using the leapfrog routine. The results are shown in figures 14 and 15: the large step size produces dynamics that look completely stable and very regular in the eccentricity (figure 15a, while the small step size shows irregular behaviour and spends most of its time as a Mars crosser (figure 15b). The energy and angular momentum do not show anything surprising (figure 13).

3.4.4.2. *Mean motion ratio* $3.1 \leq \frac{n_{ast}}{n_{jup}} \leq 3.9$. This range of mean motion ratios correspond to semi-major axes from 2.10 to 2.45 and is considered the inner asteroid belt, divided as it is by the 3:1 resonance.

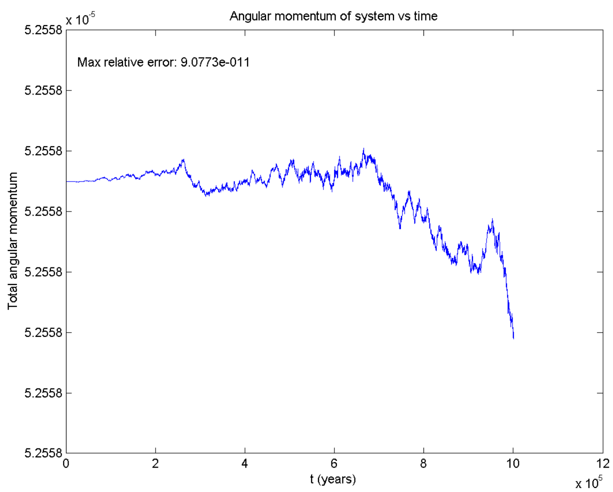
Similar to above, a consistent difference between step sizes for almost this whole region is apparent between $\tau = 1$ and $\tau = 43.31572$. Figure 16 shows this difference for two runs near the middle of this region, with initial eccentricity $e = 0.15$ (approximately the median eccentricity for bodies in the asteroid belt).



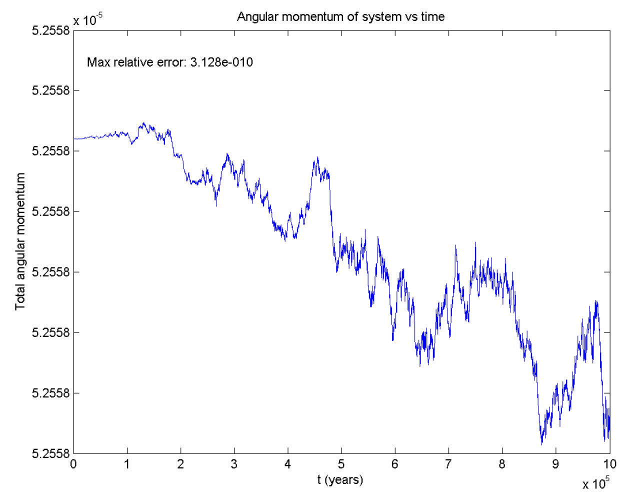
(A) Energy for run lse05n40.



(B) Energy for run e05n42.



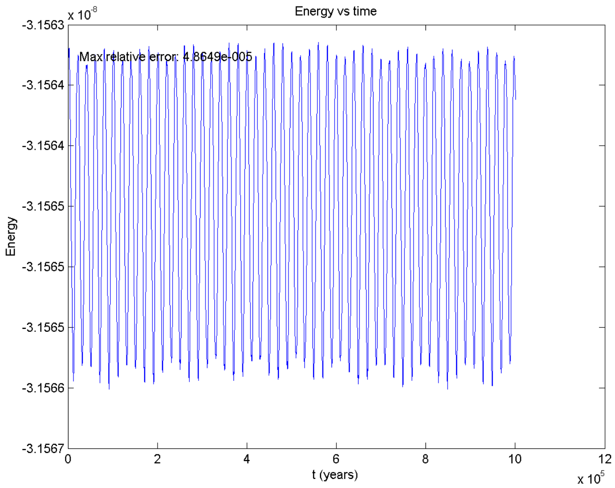
(C) Angular momentum for run lse05n40.



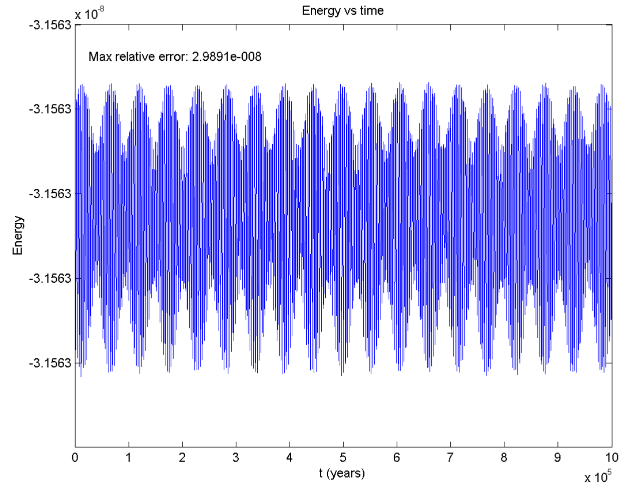
(D) Angular momentum for run e05n42.

FIGURE 13. Energy and angular momentum for for runs with identical initial conditions but different step sizes at the 4:1 resonance. Maximum errors are: (A) 6.9947×10^{-7}; (B) 8.0053×10^{-10}; (C) 9.0773×10^{-11}; and (D) 3.128×10^{-10}.

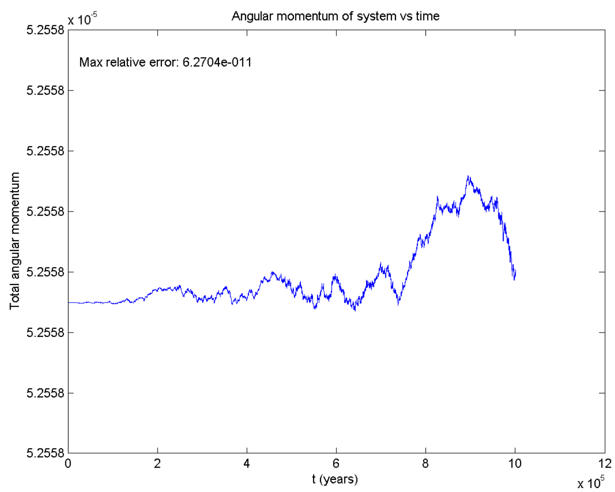
Again the question arises: where does this discrepancy come from? Figure 17 shows the results of two more megayear runs with initial conditions $e = 0.15$ and $\frac{n_{ast}}{n_{jup}} = 3.3$ calculated with a medium step size of $\tau = 20$



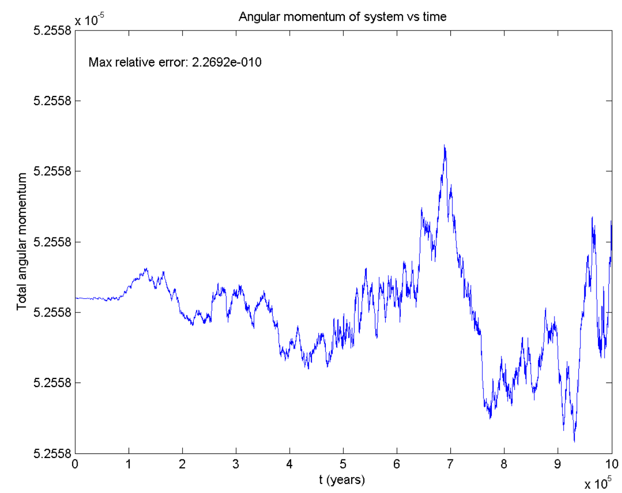
(A) Energy for run 2olse05n40.



(B) Energy for run 2oe05n40.



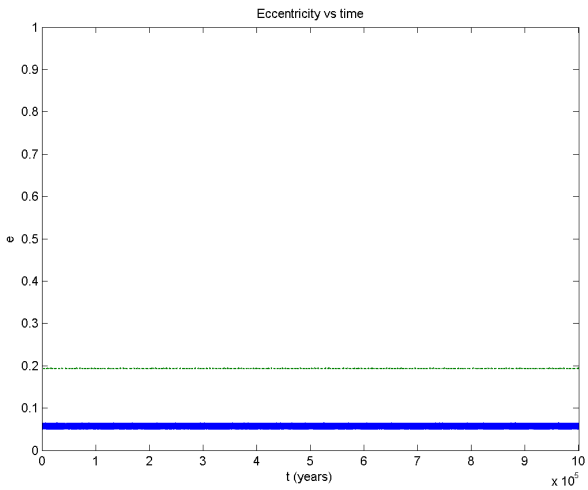
(C) Angular momentum for run 2olse05n40.



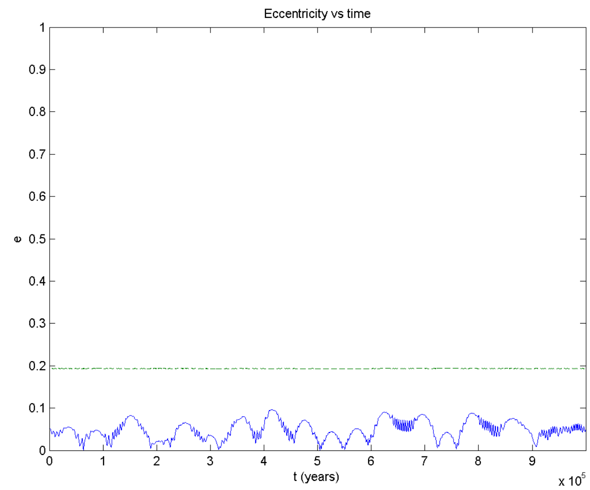
(D) Angular momentum for run 2oe05n40.

FIGURE 14. Energy and angular momentum for for runs with identical initial conditions but different step sizes at the 4:1 resonance, calculated using the leapfrog algorithm. Maximum errors are: (A) 4.8649×10^{-5}; (B) 3.0137×10^{-8}; (C) 6.2704×10^{-11}; and (D) 3.7114×10^{-10}.

days. Figure 17a was calculated with the fourth order routine, while figure 17b was calculated using leapfrog. In this case, the two routines show



(A) 2olse05n40.



(B) 2oe05n40.

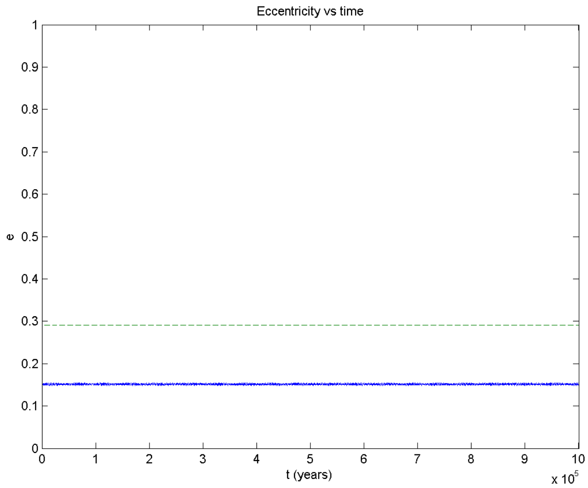
FIGURE 15. Eccentricities for runs with identical initial conditions but different step sizes at the 4:1 resonance as in figure 12, but calculated using leapfrog instead. The green dotted line represents the Mars-crossing threshold.

completely different behaviour, even though everything else was the same between the runs. Interestingly, this discrepancy between the two routines is not always so apparent. The cause of this discrepancy could be the fact that $\frac{n_{ast}}{n_{jup}} = 3.3$ is near the 10:3 minor Kirkwood gap, though other runs closer to the exact resonance make this unlikely.

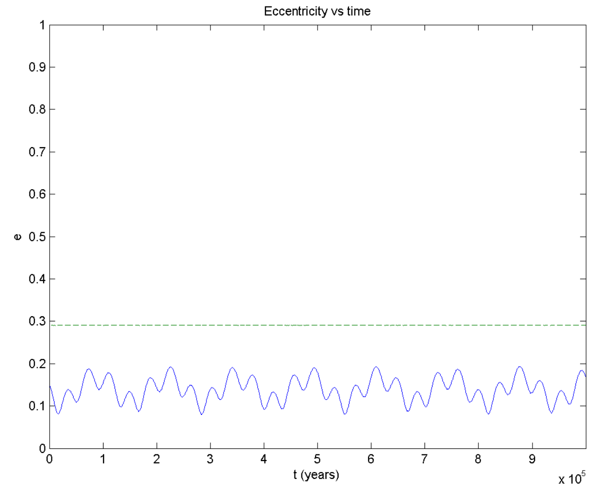
3.4.4.3. *The 3:1 resonance*. This resonance marks the middle of the asteroid belt at $a \approx 2.50$ and is perhaps the most studied of the Kirkwood gaps ([**22**], [**23**], [**2**], [**13**], [**4**], [**9**], [**10**], [**11**] and [**24**], for example).

The charateristic behaviour for an asteroid initially placed in this resonance with low eccentricity is seemingly regular for thousands of years up to tens of thousands of years interspersed with spikes of increased eccentricity high enough that direct perturbations from Mars should remove it from resonance. Murray & Holman in [**24**] summarise much of the work done by Wisdom in [**9**], [**10**] and [**11**]

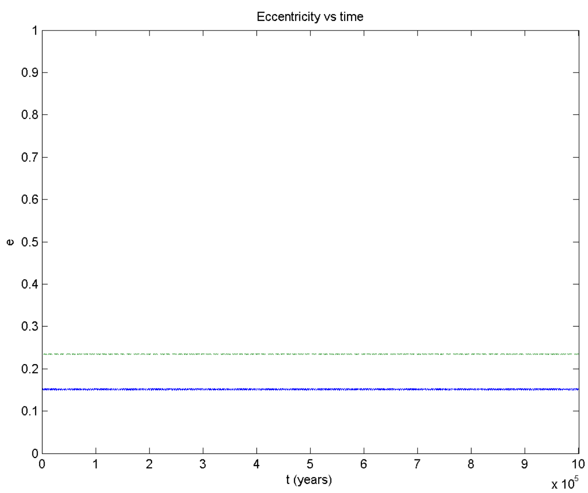
The behaviour of the asteroid in the 3:1 resonance again seems to depend on the step size used. In figure 18 the eccentricity is plotted against time



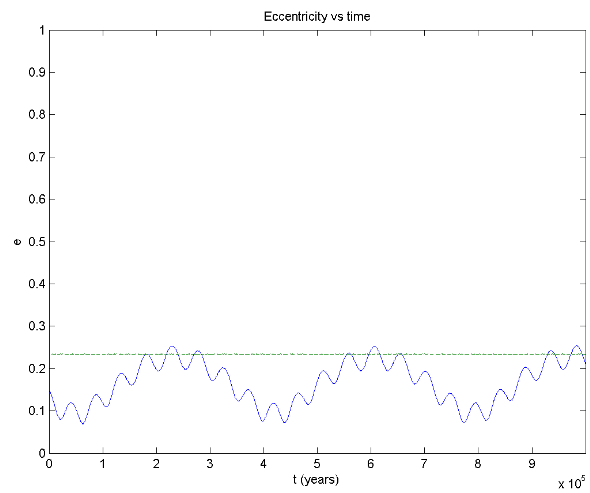
(A) lse15n33.



(B) e15n33.



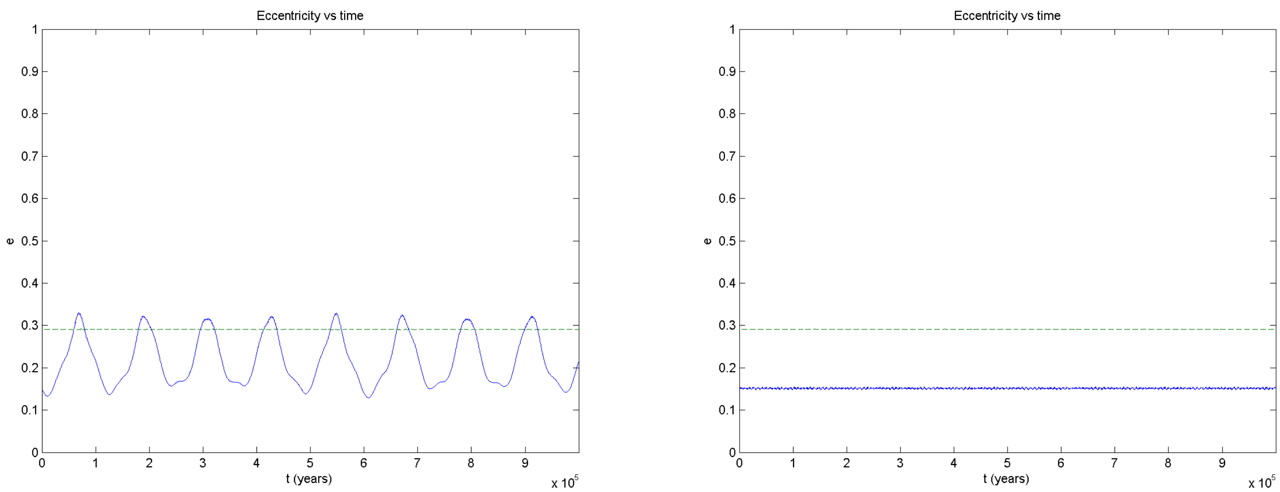
(C) lse15n37.



(D) e15n37.

FIGURE 16. Eccentricity vs time for orbits calculated with $\tau = 43.31572$ ((A) and (C)) and $\tau = 1.0$ ((B) and (C)). The green dotted line represents the Mars-crossing threshold.

for initial conditions $e = 0.15$, $\frac{n_{ast}}{n_{jup}} = 3.0$ with step sizes $\tau = 1$ day (18a) and $\tau = 43.31572$ days (18b). Though the latter shows what looks like unstable behavior, the former shows behaviour that is much more in accord with other studies: periods of thousands of years with low eccentricity and chaotic peaks which cause its orbit to become Mars crossing.

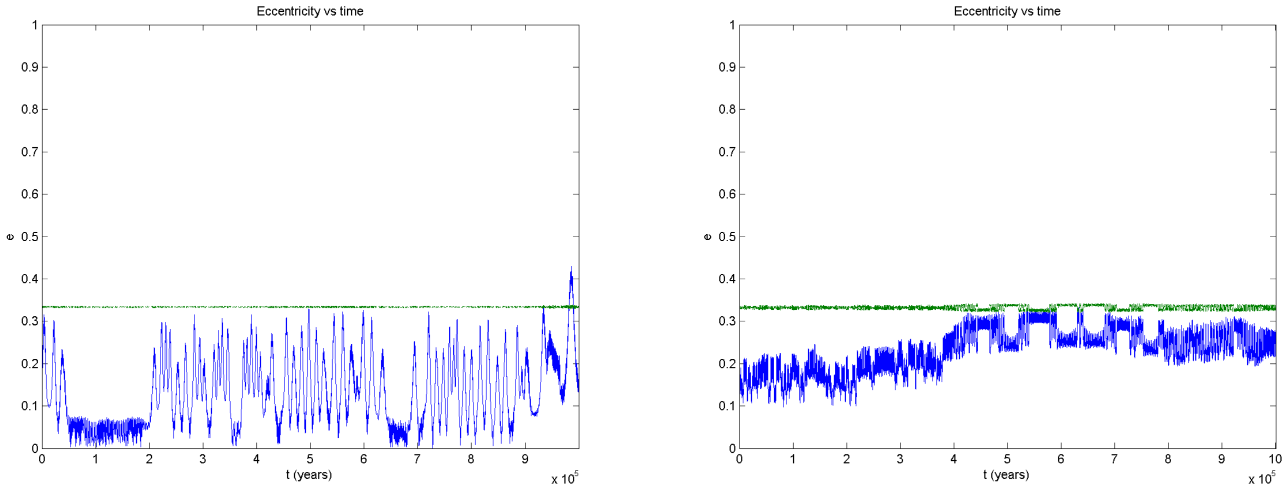


(A) mse15n33. (B) 2omse15n33.

FIGURE 17. Eccentricity vs time for orbits calculated with a medium step (ms) of $\tau = 20.00$ days. (A) is calculated using the fourth order routine, while (B) uses leapfrog. The green dotted line represents the Mars-crossing threshold.

3.4.4.4. *Mean motion ratio* $2.1 \leq \frac{n_{ast}}{n_{jup}} \leq 2.9$. The outer half of the main belt has semi major axes from 2.56 to 3.17. Most behaviour within this region is regular on a one megayear time scale, with the exception of the 5:2 resonance, whose eccentricity is plotted in figures 19 and 20. This resonance shows behaviour that is in some respects like the 3:1 resonance, though much more time tends to be spent as a Mars crosser. The large step plots show irregular behaviour, but for some reason lack the radical jumps in eccentricity.

This “damping” appears consistent in all cases when the large time step is compared to the small time step. As the large time step is a simple fraction of Jupiter’s orbital period, it is possible that some resonant effects are actually damped or averaged out in some manner. A fundamental point to keep in mind when using symplectic integrators is that they do not integrate the original Hamiltonian but a modified one that depends on the choice of time step. As such, the system could have a different phase space structure - and a given orbit may fall on one side of a separatrix or another for a given time step, even for the same initial conditions.



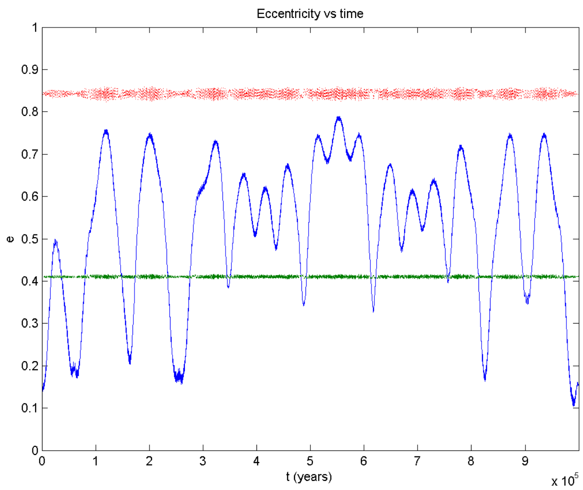
(A) e15n30. (B) lse15n30.

FIGURE 18. Eccentricity vs time for orbits in the 3:1 resonance with small time step $\tau = 1.0$ in (A) and $\tau = 43.31572$ in (B). The green dotted line represents the Mars-crossing threshold.

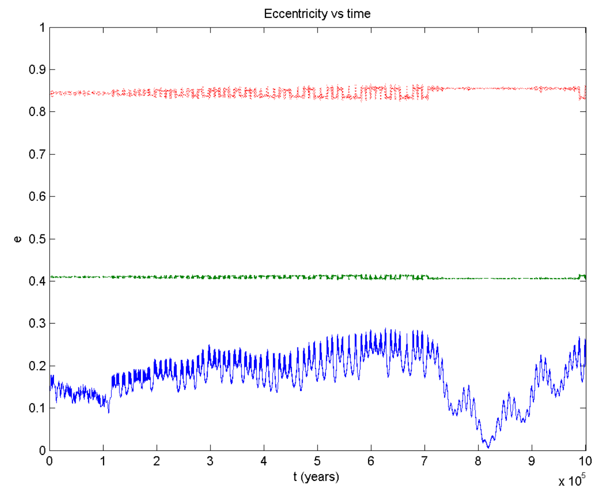
3.4.4.5. *The 2:1 resonance.* Located at 3.28 AU, the 2:1 resonance marks the outer edge of the main belt. Of all the orbits examined so far, this one shows the greatest accord between the two time steps used to calculate its orbit (figure 21). Moons gives particular treatment to the 2:1 resonance in [**23**], where the formation of the gap is attributed to slow diffusive processes, rather than sudden spikes in the orbital elements. Indeed, longer integrations of this resonance suggest that diffusion from this gap could take hundreds of millions of years.

3.4.4.6. *Mean motion ratio* $1.2 \leq \frac{n_{ast}}{n_{jup}} \leq 1.9$. This is the region between the 2:1 resonance and Jupiter itself. The overwhelming result for asteroids placed in this region is removal through perturbations from Jupiter. Some asteroids can survive with low eccentricities (generally less than 0.15) between $\frac{n_{ast}}{n_{jup}} = 1.9$ and $\frac{n_{ast}}{n_{jup}} = 1.7$, but otherwise they have short life spans less than $500,000$ years.

The exception to this is the $3:2$ resonance, known to be home to a pocket of asteroids called the Hildas, which tend to exist with eccentricities mostly between 0.1 and 0.3. Integrations showed orbits that looked chaotic, but



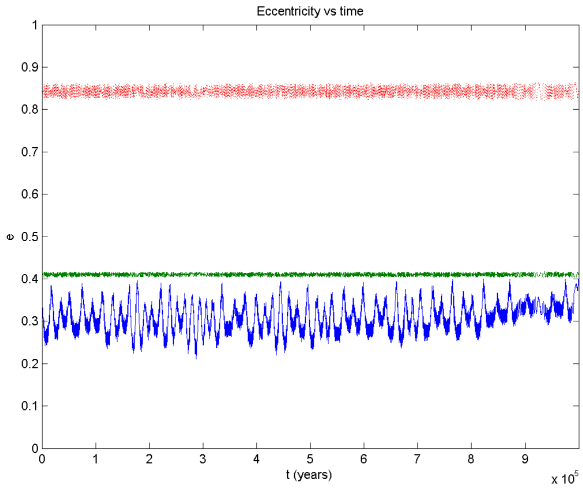
(A) e15n25.



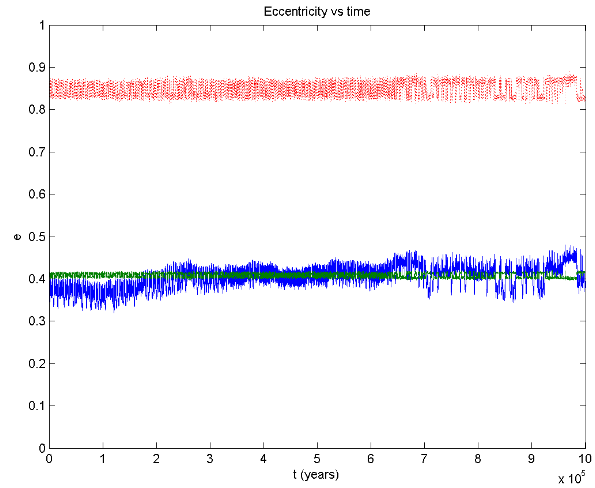
(B) lse15n25.

FIGURE 19. Eccentricity vs time for orbits in the 5:2 resonance with small time step $\tau = 1.0$ in (A) and $\tau = 43.31572$ in (B) with initial eccentricity $e = 0.15$. The green dotted line represents the Mars-crossing threshold, while the red dotted line represents the Jupiter crossing threshold.

were often stable for low initial eccentricities. Figure 22 shows two examples; most orbits with higher eccentricity resulted in the asteroid being ejected so quickly that the plots over this time scale are uninteresting.

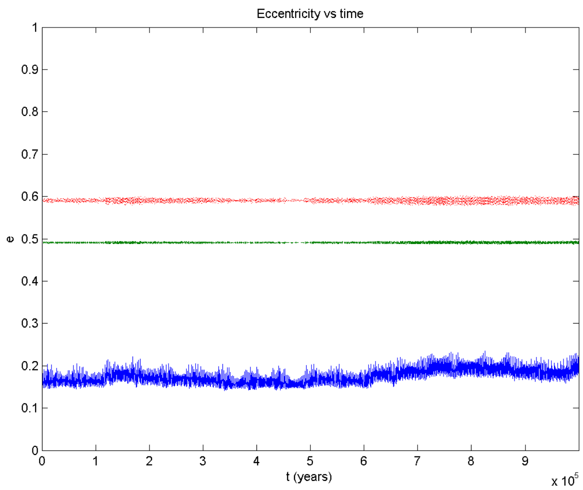


(A) e35n25.

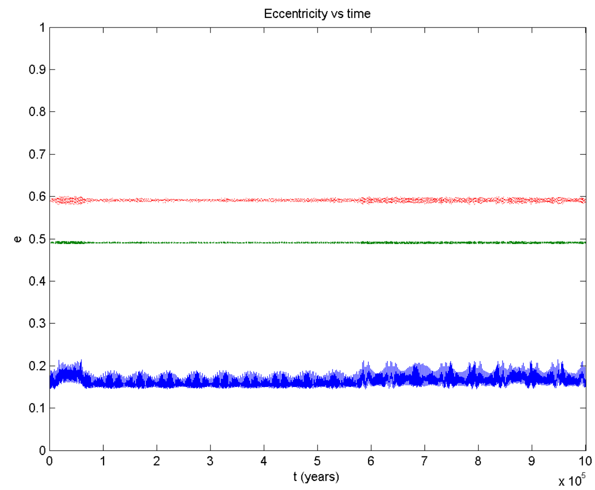


(B) lse35n25.

FIGURE 20. Eccentricity vs time for orbits in the 5:2 resonance with small time step $\tau = 1.0$ in (A) and $\tau = 43.31572$ in (B) with initial eccentricity $e = 0.35$. The green dotted line represents the Mars-crossing threshold, while the red dotted line represents the Jupiter crossing threshold. Note in (A) that the eccentricity does not spike as erratically over this time span, though it for this eccentricity still puts it in danger of removal by close encounter, while in (B) the behaviour does not look substantially different from 19b, though it quickly reaches past the Mars crossing threshold and stays there long enough that removal is practically guaranteed.

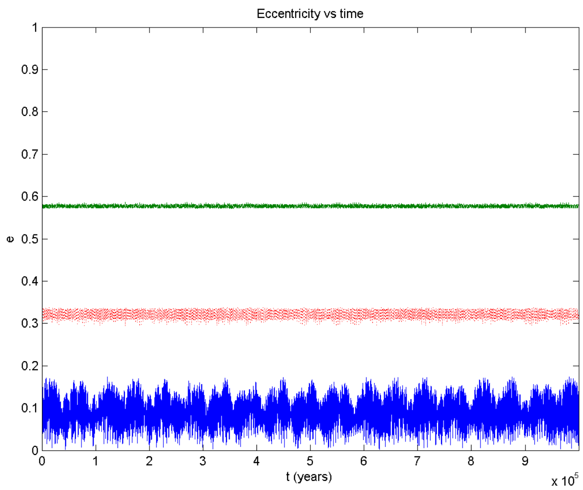


(A) e15n20.

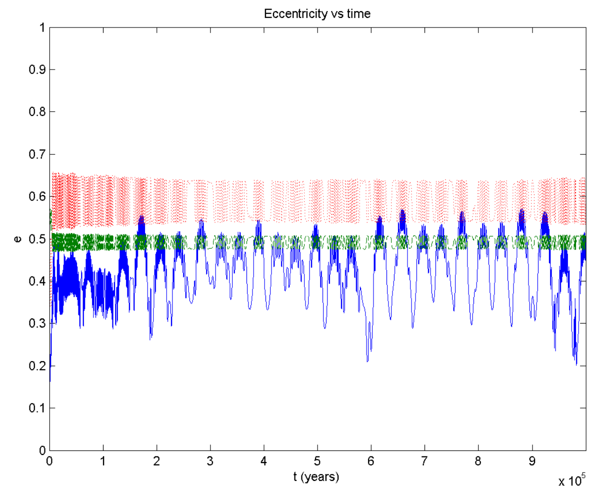


(B) lse15n20.

FIGURE 21. Eccentricity vs time for orbits in the 2:1 resonance with small time step $\tau = 1.0$ in (A) and $\tau = 43.31572$ in (B) with initial eccentricity $e = 0.15$. The green dotted line represents the Mars-crossing threshold, while the red dotted line represents the Jupiter crossing threshold. Unlike other orbits, these two seem to show much more similar dynamics to each other.



(A) e05n15.



(B) e15n15.

FIGURE 22. Eccentricity vs time for orbits in the 3:2 resonance with small time step $\tau = 1.0$. Initial eccentricity in (A) is $e = 0.05$ and in (B) $e = 0.15$. The green dotted line represents the Mars-crossing threshold, while the red dotted line represents the Jupiter crossing threshold. These orbits appear chaotic, but stable on a time scale of 1 Myear, though (B) shows the orbit frequently becoming a Mars crosser - even at times reaching out past Jupiter.

3.4.5. Long term behaviour in and out of resonance. Most of the non-resonant orbits in the main belt show regular behaviour over time scales longer than a megayear, with only small librations in the orbital elements. This does not appear to depend too strongly on the choice of time step - important to note, since most longer runs were computed with $\tau = 43.31572$ before the discrepancy discussed above could become apparent.

The long term behaviour of asteroids initially placed in a resonance depends strongly on the resonance in which it starts. Moons' analysis of the 4:1, 3:1, 5:2 and 7:3 resonances ([**23**]) shows that gravitational interactions and overlapping resonances are enough to account for these gaps, and the long term integrations performed here show signs of chaotic variation in the orbital elements for each of these resonances except for the 4:1 resonance, which resembles that in figure 12a.

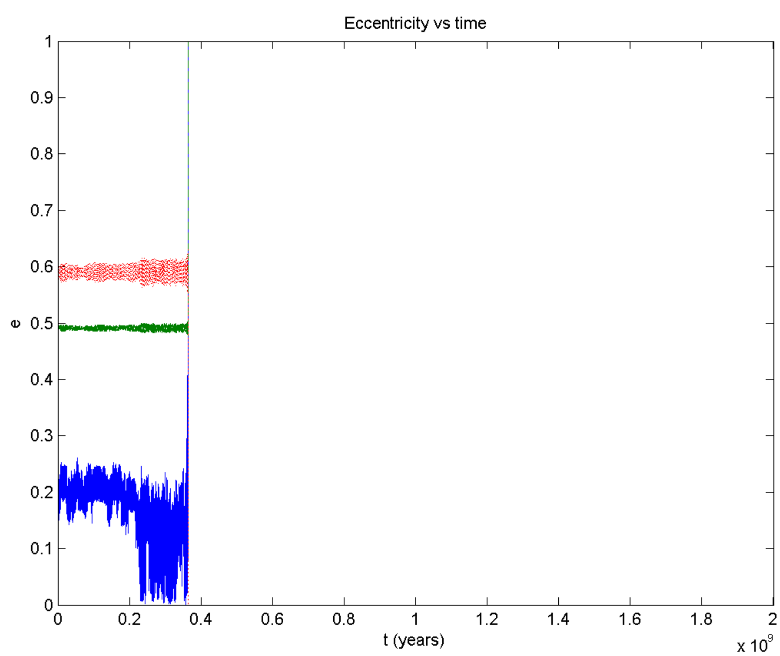


FIGURE 23. Eccentricity vs time for an asteroid orbiting in the 2:1 resonance. Initial eccentricity is $e = 0.15$, and eccentricity spikes sharply greater than 0.8 at approximately 350 Myears. The green dotted line represents the Mars-crossing threshold, while the red dotted line represents the Jupiter crossing threshold.

All runs with the asteroid placed in the 2:1 resonance show irregular (but not particularly unstable) variation in eccentricity. The exception to this rule is one integration out to 2000 Myears, shown in figure 23. The asteroid spikes in eccentricity unexpectedly at around 350 Myears and the run is terminated after the eccentricity exceeds 0.8. This is an interesting result, but unfortunately the error in the angular momentum reaches a relative magnitude of about 4×10^{-7}, which may be enough to seriously impact the accuracy of this result. Sadly, time did not permit this run to be repeated with either a different time step or the drift neutralised to see if a similar result could be repeated.

The general consensus about the 2:1 resonance is that it is not yet well understood. Overlapping resonances do produce chaotic orbits here, but do not result in eccentricities high enough for Mars or Jupiter to remove it; slow diffusive processes are responsible for removing asteroids from the region of the resonance.

3.4.6. Divergence of trajectories. A small sample of trajectories started close to one another shows clearly the difference between resonant and non-resonant orbits. Asteroids were placed with initial $\frac{n_a}{n_j} = 2.0, 2.86358736161824$ and 3.0 and run for up to 10 Myears, and sister trajectories, identical except for a difference in the asteroid's initial position of 10^-14 (approximately 1.5 mm), were integrated for the same length of time. Figures 24-26 show the results of these runs.

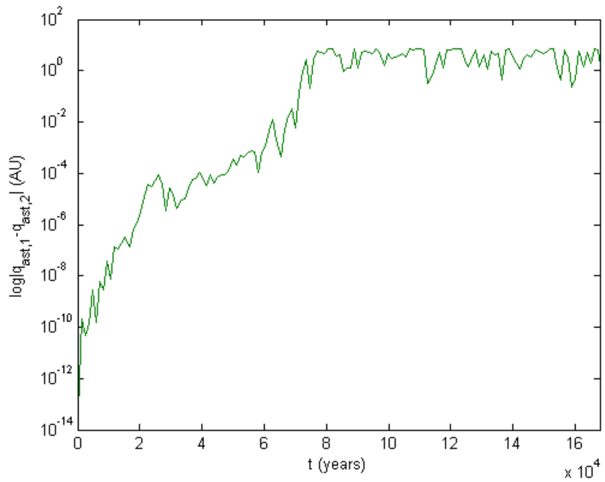


FIGURE 24. Log distance between trajectories started near one another in the 2:1 resonance. Total run time was 10 Myears, but the plot is refocused on the region of divergence. Lyapunov time is approximately 10^2 years.

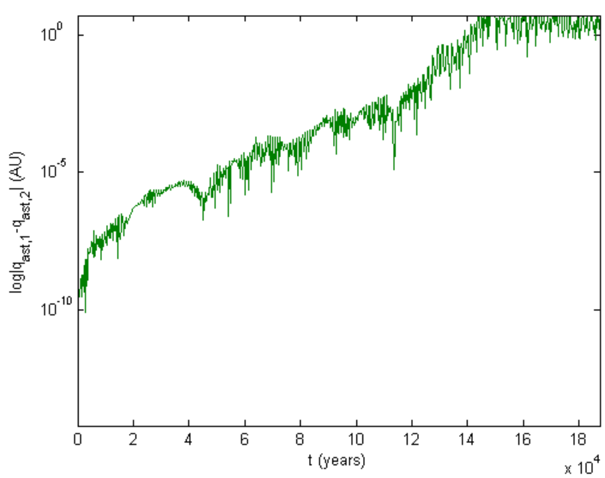


FIGURE 25. Log distance between trajectories started near one another in the 3:1 resonance. Total run time was 10 Myears, but the plot is refocused on the region of divergence. Lyapunov time is approximately 10^3 years.

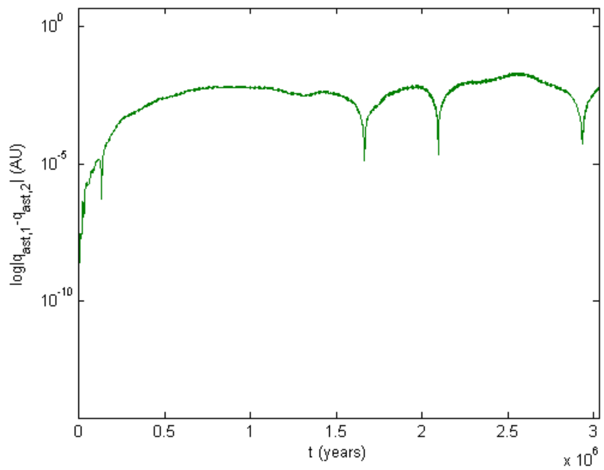


FIGURE 26. Log distance between trajectories started with $\frac{n_a}{n_j} = 2.86358736161824$, far away from any Kirkwood gaps. There is no exponential divergence of the trajectories, at least over this time scale.

3.5. When Saturn is Removed

When Saturn is excluded from the simulations, the system effectively becomes the planar 3-body problem, as Saturn's orbit being inclined slightly to Jupiter's normally results in the asteroid being pulled out of its initial plane of orbit. However, the main Kirkwood gaps still show chaotic motion. Figures 27-fig:divergencesns3 show the difference of orbits from the same initial conditions as figures 24-fig:divergences3 including Saturn, with similar Lyapunov times, where motion is chaotic.

A common feature of nonresonant runs without Saturn is that librations in the asteroid's orbital elements have only one mode - one driving frequency. This is to be expected, as there is only one body perturbing the asteroid's orbit: Jupiter. Resonant orbits that correspond to major Kirkwood gaps look similarly "cleaner", but retain the main features that result in the removal of asteroids.

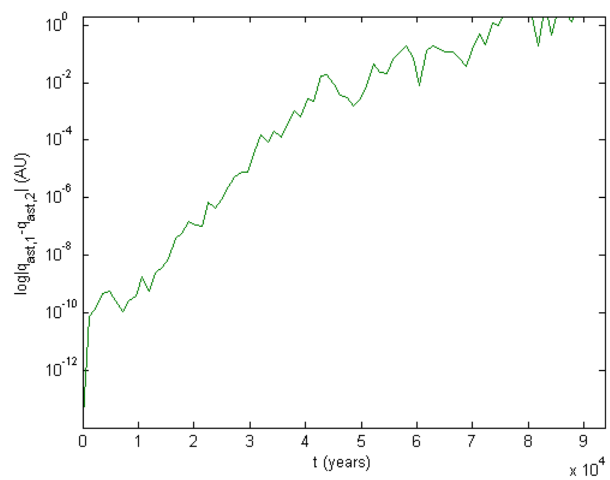


FIGURE 27. Log distance between trajectories started near one another in the 2:1 resonance without Saturn. Total run time was 10 Myears, but the plot is refocused on the region of divergence. Lyapunov time is approximately 10^2 years.

The 3:1 resonance also displays similar chaotic behaviour, with periods of low eccentricity, seemingly regular motion broken by spikes of high eccentricity. The inclination tends to remain almost zero for a long time, but it can in fact jump to over $10°$ relative to Jupiter.

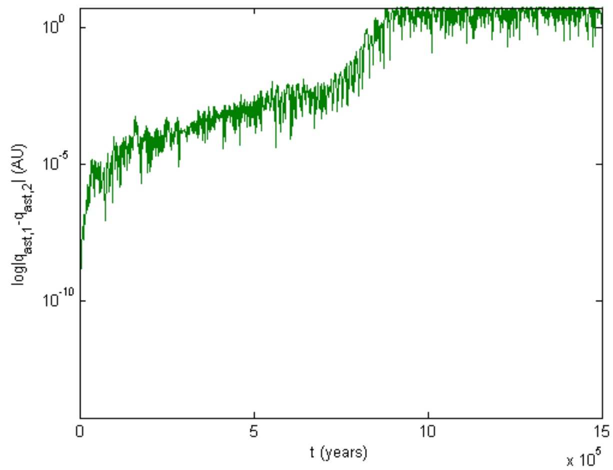


FIGURE 28. Log distance between trajectories started near one another in the 3:1 resonance without Saturn. Total run time was 10 Myears, but the plot is refocused on the region of divergence. Lyapunov time is approximately 10^3 years.

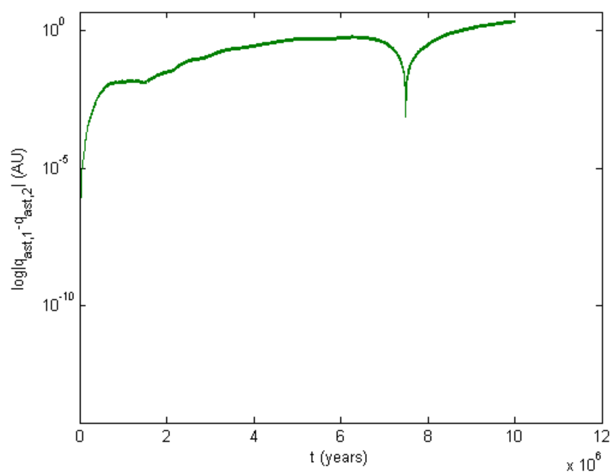
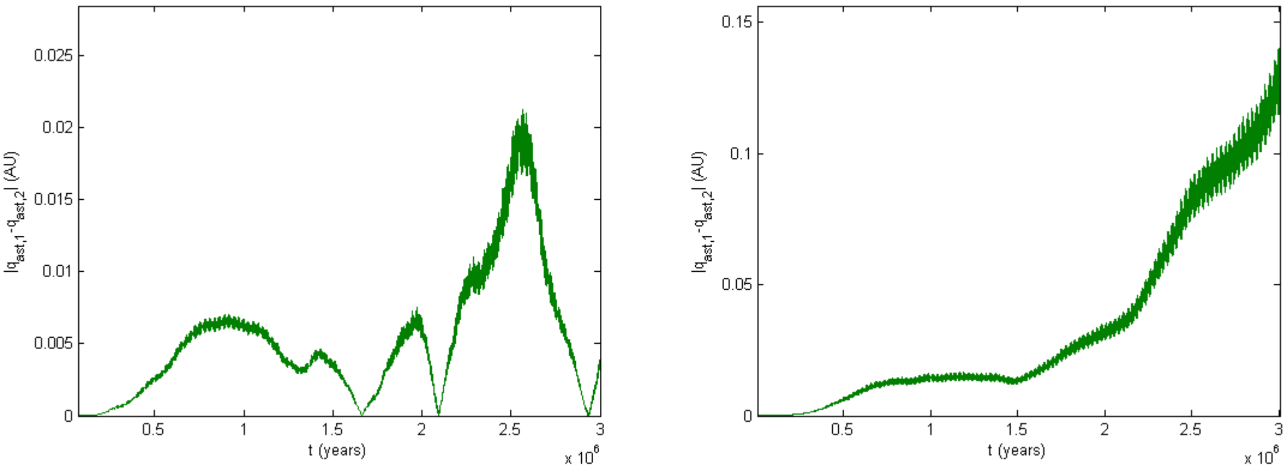


FIGURE 29. Log distance between trajectories started with $\frac{n_a}{n_j} = 2.86358736161824$, far away from any Kirkwood gaps, without Saturn. There is no exponential divergence of the trajectories, at least over this time scale.

Of particular interest was the observation of the divergence between nonresonant trajectories viewed on a linear scale with and without Saturn (figure 30). In particular, it appears (albeit with a sample size of one) that Saturn's presence may help *stabilise* orbits that are out of a major mean motion resonance.



(A) Divergence of nonresonant trajectories with Saturn included.

(B) Divergence of nonresonant trajectories with Saturn excluded.

FIGURE 30. Divergence between trajectories initally separated by approximately 10^{-14} AU (A) with Saturn and (B) without, viewed on a linear scale over 3 Myears. Note the difference in vertical scale over this time span.

Conclusion

The results of this study have been a combination of confirming things well known (for example, the chaotic motion of asteroids in major resonances and non-chaotic elsewhere), the inconclusive (how trustworthy some of the longer runs actually are) and the unexpected (if Saturn indeed does have a stabilising influence on at least some orbits far from major resonances).

Glaring discrepancies in the dynamics between runs with identical initial conditions when only the time step changes suggest that further work is required to understand the role that the choice of time step plays in the symplectic integration of chaotic systems: does a time step close to a resonant frequency of the system affect the dynamics of the numerically integrated system, or was the ~ 43.3 day time step too large to capture finer features of the dynamics of relatively short-period resonant orbits? Certainly it illustrates the care that must always be taken when undertaking numerical work.

On the other hand, runs where the time step had little relation to any natural frequency of the system often showed good agreement with results discovered from previous analytical studies of the mean motion resonances, as well as numerical investigations over the last thirty years of research.

Removing Saturn from the integrations most noticeably results in the asteroid's inclination becoming almost constant and other orbital elements librating more simply. Perturbations from Saturn may cause more rapid removal of asteroids from the 4:1, 3:1, 5:2 and 7:3 Kirkwood gaps by those perturbations adding to the underlying chaos of the resonances with Jupiter. The 2:1 resonance, still not well understood in the literature, appeared to behave chaotically, but did not suffer any large amplitude variations in its orbital elements.

The role of Saturn in the dynamics of the asteroid belt is worth further investigation, however, as the divergence of nearby nonresonant trajectories appeared slower over three megayears when Saturn was present than when it was not. A deeper understanding of the phase space of the asteroid belt assists in showing where the boundaries of chaotic regions actually lie, as would larger surveys, such as have already been done by Saha ([**22**]) and others, to compare against the known distribution of the asteroid belt.

APPENDIX A

Osculating Orbital Elements

The equations of motion for the two-body gravitational problem can be solved exactly. It can easily be proven that the motion is planar, linear momentum of the system is conserved and angular momentum of the system is conserved. The explicit equations of motion reveal that the paths of the bodies correspond to conic sections (ellipse, parabola, hyperbola) depending on the energy of the system, which is also constant.

This fact means that orbits can be characterised by six numbers, called the Keplerian elements:

a) eccentricity e, which sets the shape of the ellipse;
b) semi-major axis a (measured in astronimical units (AU)), which sets the size;
c) inclination i, which sets the angle of deviation of the orbital plane from an arbitrary reference plane (called the ecliptic);
d) longditude of ascending node Ω, which sets the line of intersection between the plane of orbit and the reference plane, measured from an arbitrary reference direction called the vernal node;
e) argument of perihelion (or periapsis) ω, which defines the orientation of the orbit in its plane (at what angle the body passes closest to the centre of mass of the system, measured from the ascending node); and
f) mean anomaly at epoch M_0, which describes how far around the orbit's "auxilliary circle" the body has travelled, measured from perihelion.

The auxiliary circle is a circle of radius a, with its centre at the centre of the ellipse (i.e. when $e = 0$ the mean and true anomalies coincide). The mean anomaly is related to the true anomaly ν (which is in fact more easily calculated from the body's orbital state vectors $\mathbf{r}$ and $\mathbf{v}$, which are position and velocity relative to the primary body), the angle measured from perihelion that the body has travelled about the centre of mass.

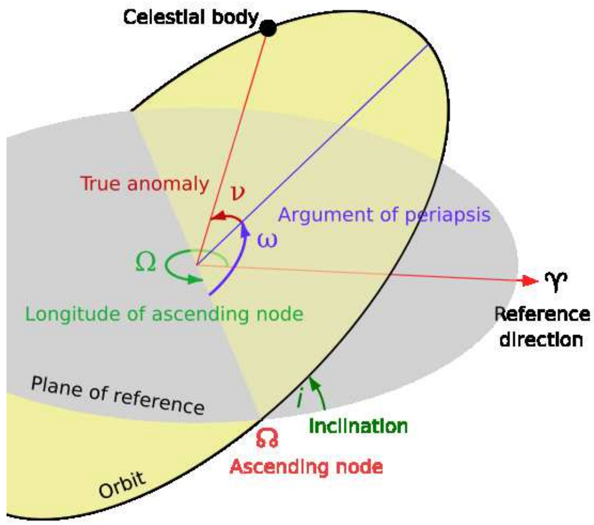


FIGURE 1. Illustration of the meaning of the orbital elements i, Ω, ω and ν. Used under the GNU Free Document License, Version 1.2. Copryright Lucas Snyder. Source: *http://en.wikipedia.org/wiki/Image:Orbit1.svg*.

In the two-body problem, eccentricity, semi-major axis, argument of perihelion, ascending node and inclination remain constant; the only orbital element that changes is the true anomaly as the bodies orbit their centre of gravity. In an n-body system for $n > 2$ like the solar system, the effects of other bodies perturb the motion of any given body, causing precession: each revolution the arument of perhelion has shifted slightly further around, relative to a fixed frame of reference (e.g. distant stars). [1]

Another quantity of interest is the mean motion n (measured in revolutions per day, not to be confused with the number n of bodies in the system), which is set by the semi-major axis by the relationship

$$n = \sqrt{\frac{GM}{a^3}},$$

[1]Incidentally, this effect on the orbit of Uranus is the basis of Neptune's discovery: a discrepancy between published tables of the planets' orbits and observations of Uranus prompted mathematical predictions of an extra planet, which was found on the 23rd of September, 1846, within one degree of arc of the location predicted by French mathematician Urbain Le Verrier.

where M is the mass of the central body (in solar masses $\mathrm{M}_\odot$) and G is Newton's gravitational constant (in $\mathrm{AU}^3\mathrm{M}_\odot^{-1}\mathrm{d}^{-2}$).

Although the orbital elements in the n-body problem are not constant, they can still be used to track features of the orbits by calculating them for each body as if it were instantaneously in the 2-body problem with the sun and it alone. This intantaneous 2-body orbit is tangential to the true orbit, hence why the elements calculated thusly are called the osculating elements, from Latin *osculare* ("to kiss").

The equations for the osculating elements are given in [**25**].

APPENDIX B

Initial conditions for the Sun, Jupiter and Saturn

Body	m ($\mathrm{M}_\odot$)	$\mathbf{q}$ (AU)	$\mathbf{v}$ (AU d^{-1})
Sun	1.00000597682	0 0 0	0 0 0
Jupiter	0.000954786104043	-3.5023653 -3.8169847 -1.5507963	0.00565429 -0.0041249 -0.00190589
Saturn	0.000285583733151	9.0755314 -3.0458353 -1.6483708	0.00168318 0.00483525 0.00192462

TABLE 1. Initial conditions for the Sun, Jupiter and Saturn corresponding to their actual positions and velocities at 0h00, 24th of September 1994. From Hairer, Lubich and Wanner [**18**].

APPENDIX C

Specifying the Asteroid's Initial Conditions

The initial conditions I use for the outer planets are given in [**18**], with the sun initially at the origin.

The asteroid's initial conditions can be chosen arbitrarily, but to keep the parameter space simple, we want to place an asteroid initially in line with Jupiter, with the same inclination as Jupiter, and be able to specify its eccentricity and orbital resonance with Jupiter, but with its orbit at either aphelion or perihelion at the initial moment.

First we let $\mathbf{r}_{ast} = p\mathbf{r}_{jup}$, $\mathbf{h}_{ast} = h\mathbf{h}_{jup}$, where $\mathbf{r}_{ast}$ and $\mathbf{r}_{jup}$ are respectively the vector positions of the asteroid and Jupiter with respect to the sun (initially at the origin), $\mathbf{h}_{ast}$ and $\mathbf{h}_{jup}$ are the angular momentum per unit mass of, respectively, the asteroid and Jupiter and p and h are scalars.

Fixing the $\mathbf{r}_{ast}$ parallel to $\mathbf{r}_{jup}$ and $\mathbf{h}_{ast}$ parallel to $\mathbf{h}_{jup}$ sets up the orbit of the asteroid such that it is initially in line with and has the same inclination as Jupiter's orbit, eliminating many variables from consideration.

Also, $\mathbf{h}_{jup} = \mathbf{r}_{jup} \times \mathbf{v}_{jup}$, where $\mathbf{v}_{jup}$ is Jupiter's vector velocity, and $\mathbf{h}_{ast} = \mathbf{r}_{ast} \times \mathbf{v}_{ast}$, where $\mathbf{v}_{ast}$ is same for the asteroid.

Again, to simplify the proceedings and eliminate more variables from consideration, we choose $\mathbf{v}_{ast}$ to have direction such that $\mathbf{r}_{ast}$, $\mathbf{v}_{ast}$ and $\mathbf{h}_{ast}$ are mutually orthogonal.

Given a desired mean motion resonance (n) and eccentricity (e), we can determine the asteroid's semi-major axis, given that Jupiter's mean motion is also known. In general, $a = (\frac{\mu}{(nn_{jup})^2})^{\frac{1}{3}} = \frac{h^2}{\mu(1-e^2)}$, where a is osculating semi-major axis, n_{jup} is Jupiter's (osculating) mean motion, n is the mean motion ratio we desire (with nn_{jup} being the desired mean motion of the asteroid), h is the magnitude of the angular momentum per unit mass vector and $\mu = G(m_1 + m_2)$, where G is Newton's gravitational constant and m_1

and m_2 are the masses of the bodies we are considering (m_1 is typically the Sun or central body, m_2 is the other body), so for our purposes we have

$$h = \frac{\sqrt{\mu a \left(1 - e^2\right)}}{|h_{jup}|}.$$

Now with the relation $\mathbf{h}_{ast} = \mathbf{r}_{ast} \times \mathbf{v}_{ast} = h\mathbf{h}_{jup}$, and $\mathbf{r}_{ast}$ and $\mathbf{v}_{ast}$ are of known directions but unknown magnitudes, we can take the modulus of each side to get

$$\begin{aligned}
|\mathbf{r}_{ast} \times \mathbf{v}_{ast}| &= h|\mathbf{h}_{jup}| \\
|\mathbf{r}_{ast}||\mathbf{v}_{ast}| &= h|\mathbf{h}_{jup}| \;\; (\text{as } \mathbf{r}_{ast} \perp \mathbf{v}_{ast}) \\
|\mathbf{v}_{ast}| &= \frac{h|\mathbf{h}_{jup}|}{|\mathbf{r}_{ast}|} \\
&= \frac{h}{p}\frac{|\mathbf{h}_{jup}|}{|\mathbf{r}_{jup}|}
\end{aligned}$$

A different equation relating the semi-major axis of a body to the distance (R) from and speed (V) relative to the body it is orbiting is

$$a = \left(\frac{2}{R} - \frac{V^2}{\mu}\right)^{-1}$$

Thus we have

$$\begin{aligned}
a &= \left(\frac{2}{p|\mathbf{r}_{jup}|} - \frac{|\mathbf{v}_{ast}|^2}{\mu}\right)^{-1} \\
&= \left(\frac{2}{p|\mathbf{r}_{jup}|} - \frac{1}{\mu}\left(\frac{h}{p}\frac{|\mathbf{h}_{jup}|}{|\mathbf{r}_{jup}|}\right)^2\right)^{-1} \\
&= \left(\frac{2\mu p|\mathbf{r}_{jup}| - (h|\mathbf{h}_{jup}|)^2}{\mu(p|\mathbf{r}_{jup}|)^2}\right)^{-1} \\
&= \frac{\mu(p|\mathbf{r}_{jup}|)^2}{2\mu p|\mathbf{r}_{jup}| - (h|\mathbf{h}_{jup}|)^2},
\end{aligned}$$

which leads to a quadratic equation in p with solutions

$$p_+ = \frac{a}{|\mathbf{r}_{jup}|} + \frac{1}{\mu|\mathbf{r}_{jup}|}\sqrt{(\mu a)^2 - \mu a(h|\mathbf{h}_{jup}|)^2)}$$

and

$$p_- = \frac{a}{|\mathbf{r}_{jup}|} - \frac{1}{\mu|\mathbf{r}_{jup}|}\sqrt{(\mu a)^2 - \mu a(h|\mathbf{h}_{jup}|)^2)}.$$

The former equation p_+ corresponds to placing the asteroid at aphelion, the latter at perihelion.

Now that p is known (using either solution above) we can choose a magnitude for the vector v_{ast}, from

$$|\mathbf{v}_{ast}| = \frac{h}{p}\frac{|\mathbf{h}_{jup}|}{|\mathbf{r}_{jup}|},$$

with a smaller value corresponding to the p_+ solution and larger corresponding to p_-, as expected, as a body orbiting further away from its partner is expected to travel more slowly than one travelling near.

Thus we have enough conditions to specify the initial location and velocity of the asteroid, at least for the small number of cases we will sample.

APPENDIX D

Proof that Leapfrog Conserves Angular Momentum

Angular momentum of a body i is defined as $\mathbf{h}_i = \mathbf{q}_i \times \mathbf{p}_i$. The angular momentum of a system of N bodies is then $\mathbf{h}_S = \sum_{i=1}^{N} \mathbf{h}_i$. In a finite mapping scheme, the angular momentum at time step n is denoted by a further subscript.

The leapfrog algorithm is as follows for each body i:

$$\begin{aligned}
\mathbf{q}_{i_{n+\frac{1}{2}}} &= \mathbf{q}_{i_n} + \frac{\tau}{2}\frac{\mathbf{p}_{i_n}}{m_i} \\
\mathbf{p}_{i_{n+1}} &= \mathbf{p}_{i_n} - \tau \sum_{\substack{j=1 \\ j\neq i}}^{N} \frac{G m_i m_j (\mathbf{q}_{i_{n+\frac{1}{2}}} - \mathbf{q}_{j_{n+\frac{1}{2}}})}{|\mathbf{q}_{i_{n+\frac{1}{2}}} - \mathbf{q}_{j_{n+\frac{1}{2}}}|^3} \\
\mathbf{q}_{i_{n+1}} &= \mathbf{q}_{i_{n+\frac{1}{2}}} + \frac{\tau}{2}\frac{\mathbf{p}_{i_{n+1}}}{m_i}.
\end{aligned}$$

For notational simplicity, let $A_i = \frac{\tau}{2m_i}$ and $B_{ij} = -\frac{\tau G m_i m_j}{|\mathbf{q}_{i_{n+\frac{1}{2}}} - \mathbf{q}_{j_{n+\frac{1}{2}}}|^3}$.

Now leapfrog is simply

$$\begin{aligned}
\mathbf{q}_{i_{n+\frac{1}{2}}} &= \mathbf{q}_{i_n} + A_i \mathbf{p}_{i_n} \\
\mathbf{p}_{i_{n+1}} &= \mathbf{p}_{i_n} + \sum_{\substack{j=1 \\ j\neq i}}^{N} B_{ij} (\mathbf{q}_{i_{n+\frac{1}{2}}} - \mathbf{q}_{j_{n+\frac{1}{2}}}) \\
\mathbf{q}_{i_{n+1}} &= \mathbf{q}_{i_{n+\frac{1}{2}}} + A_i \mathbf{p}_{i_{n+1}},
\end{aligned}$$

which can be expressed as

$$\mathbf{p}_{i_{n+1}} = \mathbf{p}_{i_n} + \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij}(\mathbf{q}_{i_n} + A_i \mathbf{p}_{i_n} - \mathbf{q}_{j_n} - A_j \mathbf{p}_{j_n})$$

$$\mathbf{p}_{i_{n+1}} = \mathbf{q}_{i_n} + 2A_i \mathbf{p}_{i_n} + \sum_{\substack{j=1 \\ j \neq i}}^{N} A_i B_{ij}(\mathbf{q}_{i_n} + A_i \mathbf{p}_{i_n} - \mathbf{q}_{j_n} - A_j \mathbf{p}_{j_n}).$$

The angular momentum of the system at step $N + 1$ is

$$
\begin{aligned}
\mathbf{h}_{Sn+1} &= \sum_{i=1}^{N} \mathbf{h}_{in+1} \\
&= \sum_{i=1}^{N} \mathbf{q}_{in+1} \times \mathbf{p}_{in+1} \\
&= \sum_{i=1}^{N} ((\mathbf{q}_{in} + 2A_i \mathbf{p}_{in} + \sum_{\substack{j=1 \\ j \neq i}}^{N} A_i B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}})) \\
&\quad \times (\mathbf{p}_{in} + \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}}))) \\
&= \sum_{i=1}^{N} (\mathbf{q}_{in} \times \mathbf{p}_{in} + 2A_i \mathbf{p}_{in} \times \mathbf{p}_{in} + \sum_{\substack{j=1 \\ j \neq i}}^{N} A_i B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}}) \times \mathbf{p}_{in} \\
&\quad + \mathbf{q}_{in} \times \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}}) + 2A_i \mathbf{p}_{in} \times \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}}) \\
&\quad + \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}}) \times \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}})) \\
&= \sum_{i=1}^{N} (\mathbf{h}_{in} + \mathbf{q}_{in+\frac{1}{2}} \times \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} (\mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{jn+\frac{1}{2}})) \\
&= \sum_{i=1}^{N} (\mathbf{h}_{in} + \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} (\mathbf{q}_{in+\frac{1}{2}} \times \mathbf{q}_{in+\frac{1}{2}} - \mathbf{q}_{in+\frac{1}{2}} \times \mathbf{q}_{jn+\frac{1}{2}})) \\
&= \sum_{i=1}^{N} (\mathbf{h}_{in} - \sum_{\substack{j=1 \\ j \neq i}}^{N} B_{ij} \; \mathbf{q}_{in+\frac{1}{2}} \times \mathbf{q}_{jn+\frac{1}{2}}).
\end{aligned}
$$

However, when both summations are expanded, pairs of terms will appear that look like

$$
B_{ij} \; \mathbf{q}_{in+\frac{1}{2}} \times \mathbf{q}_{jn+\frac{1}{2}} + B_{ji} \; \mathbf{q}_{jn+\frac{1}{2}} \times \mathbf{q}_{in+\frac{1}{2}},
$$

which cancel, leaving only

$$\mathbf{h}_{Sn+1} = \sum_{i=1}^{N} \mathbf{h}_{in} = \mathbf{h}_{Sn}$$

as required.

While exact conservation of angular momentum is not proved here for the fourth order algorithm, the argument proceeds along similar lines.

APPENDIX E

Figures Comparing Evolution of System With and Without Long Term Drift

The following figures compare the evolution of several aspects of orbits when the drift discussed in sections 3.1.2.1 and 3.3 is present and neutralised, justifying that although truncation inevitably becomes significant over particularly long time scales, the dynamics of the orbits do not appear significantly unreliable over 100 Myears if the drift is not neutralised. Without a deeper understanding of the structure of the phase space for each system, however, this cannot be more than a tentative statement. Further, these runs were performed with a time step $\tau = 43.31572$ days, so as per the observations in section 3.4.4 the accuracy of the dynamics themselves may not be trustworthy at all, even if the effect of the drift is negligible over this time span.

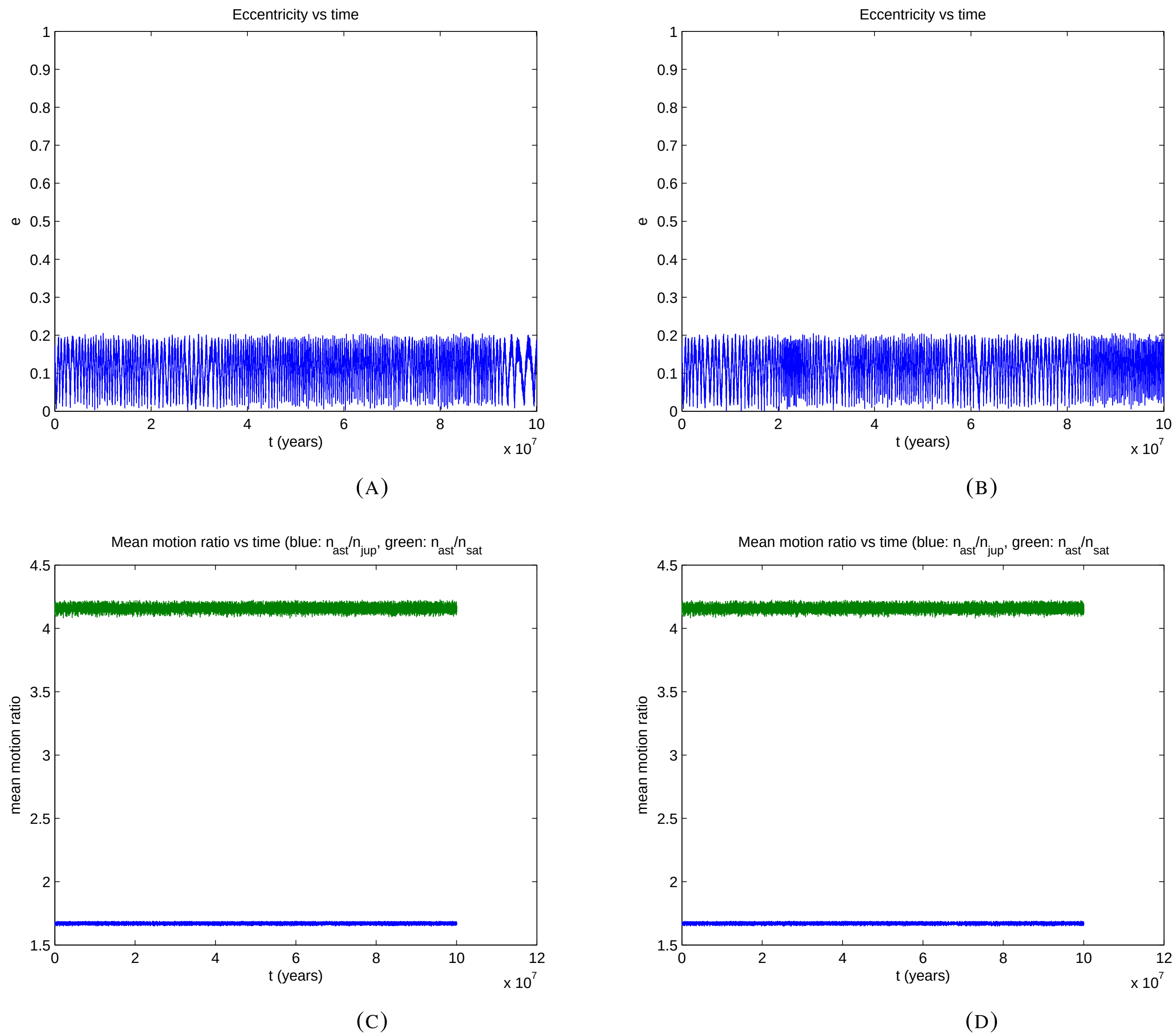
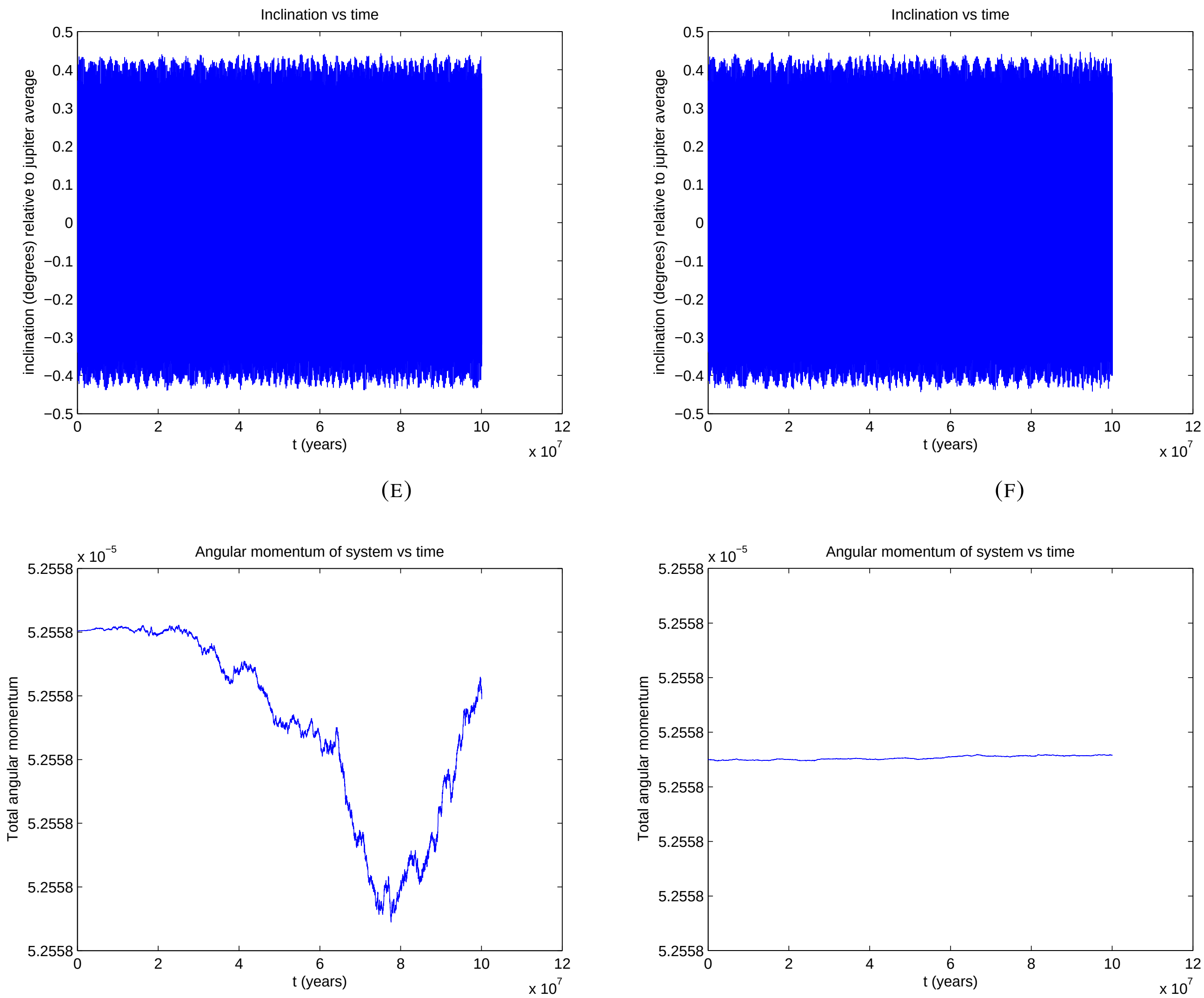
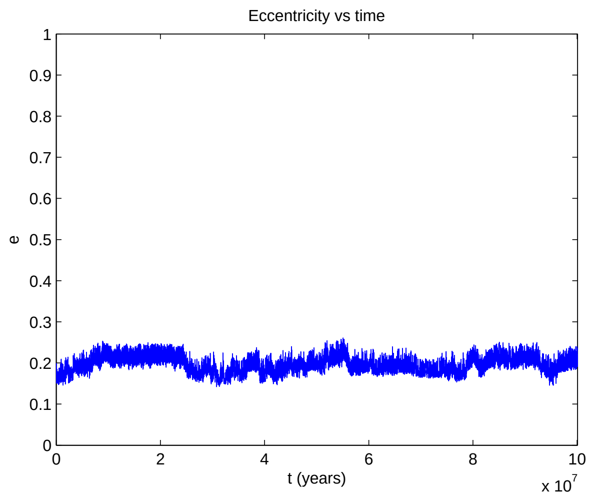


FIGURE 1. Comparison of: (A) and (B) eccentricity; (C) and (D) mean motion ratio of asteroid with Jupiter (blue/lower curve) and Saturn (green/upper curve); (E) and (F) inclination; (G) and (H) angular momentum. The former of each pair is for the system with drift included, while the initial momentum is neutralised in the latter. Both runs started with the asteroid at perihelion, directly opposite Jupiter's IC, initial eccentricity $e = 0.15$ and inital mean motion ratio with Jupiter being $\frac{n_{ast}}{n_{jup}} = 1.666666666666667$.

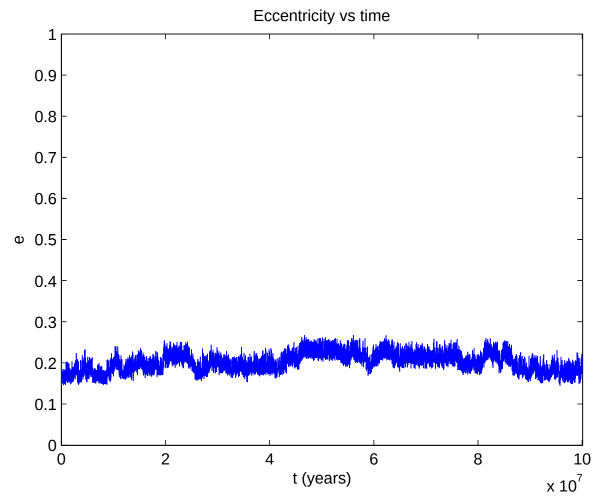


(E) (F) (G) (H)

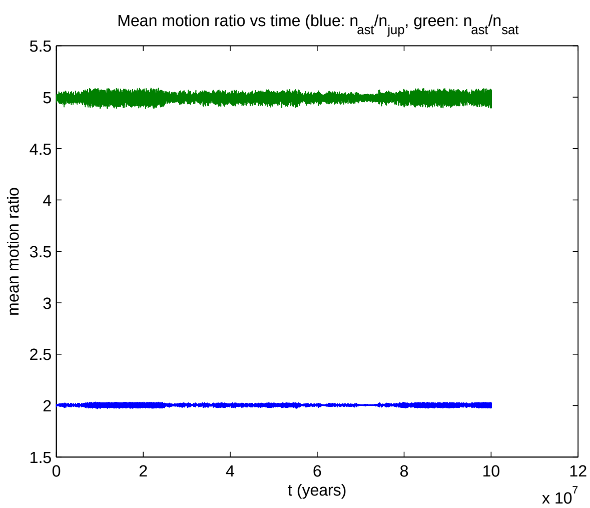
FIGURE 2. Continuation of previous figure.



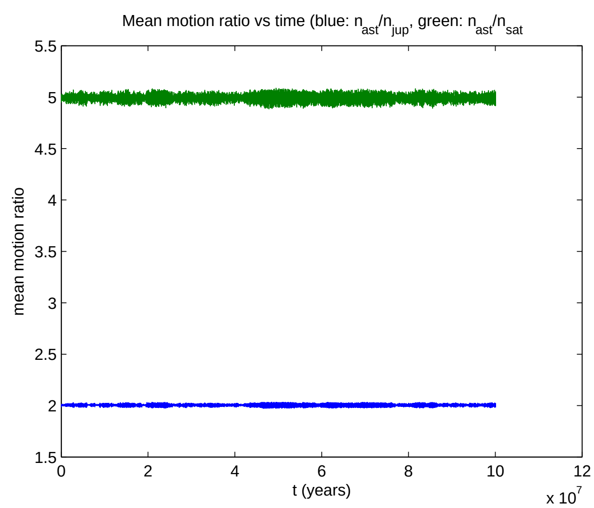
(A)



(B)



(C)



(D)

FIGURE 3. Comparison of: (A) and (B) eccentricity; (C) and (D) mean motion ratio of asteroid with Jupiter (blue/lower curve) and Saturn (green/upper curve); (E) and (F) inclination; (G) and (H) angular momentum. The former of each pair is for the system with drift included, while the initial momentum is neutralised in the latter. Both runs started with the asteroid at perihelion, directly opposite Jupiter's IC, initial eccentricity $e = 0.15$ and inital mean motion ratio with Jupiter being $\frac{n_{ast}}{n_{jup}} = 2.00$.

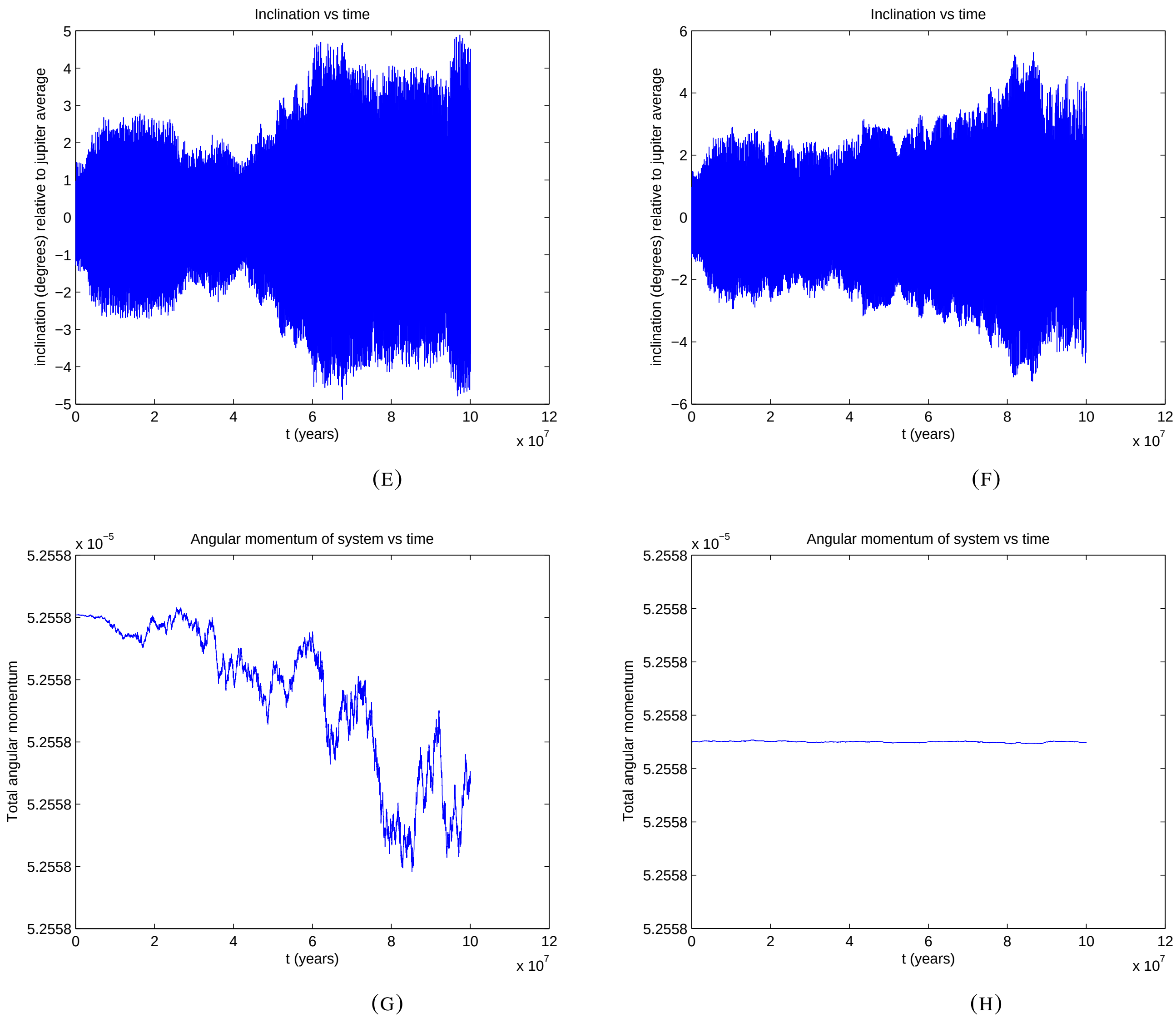
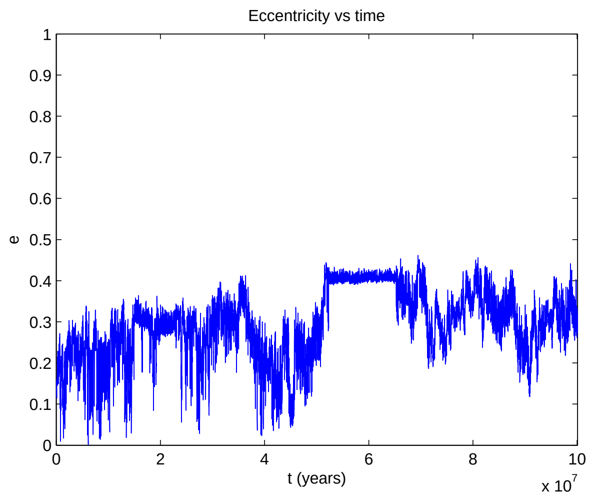
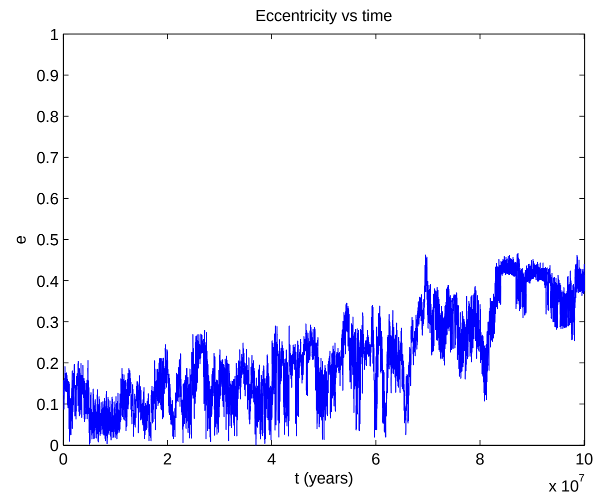


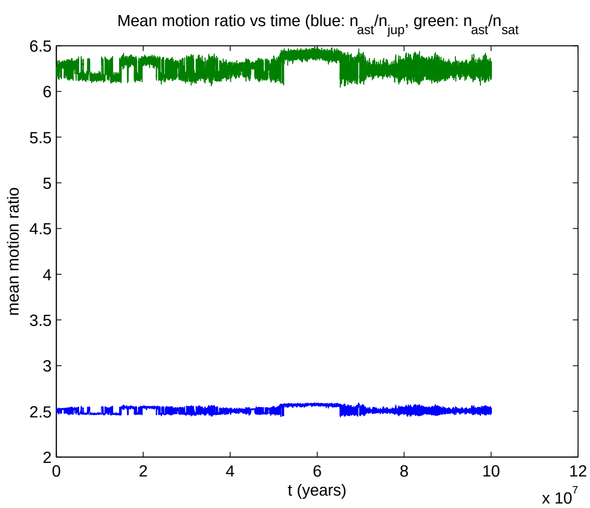
FIGURE 4. Continuation of previous figure.



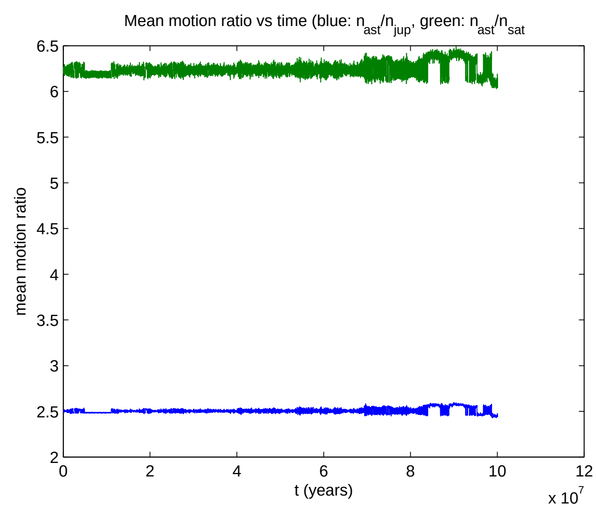
(A)



(B)

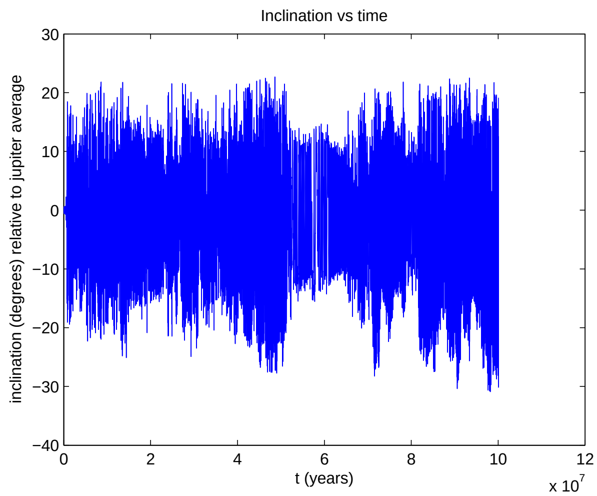


(C)

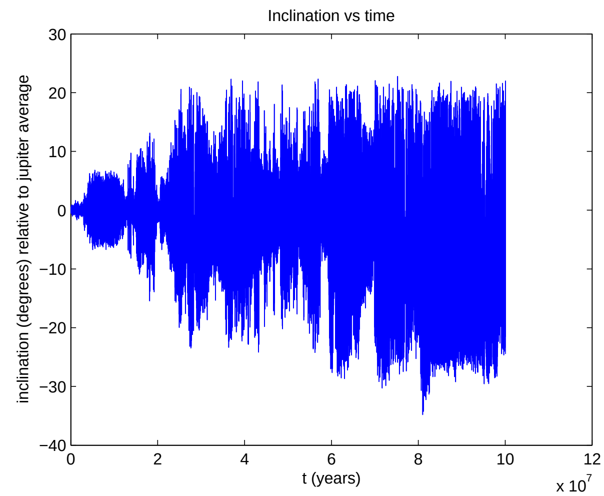


(D)

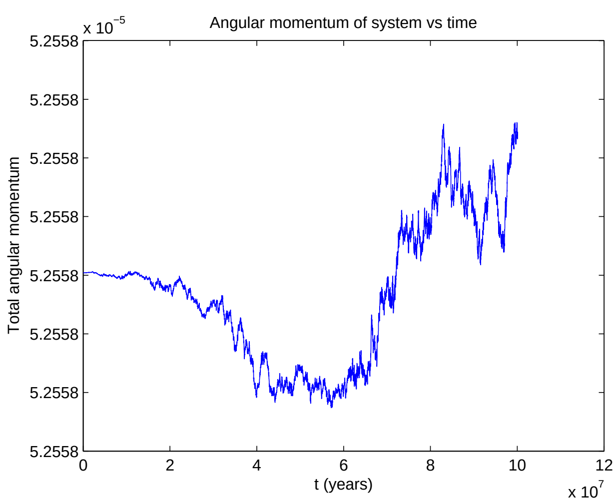
FIGURE 5. Comparison of: (A) and (B) eccentricity; (C) and (D) mean motion ratio of asteroid with Jupiter (blue/lower curve) and Saturn (green/upper curve); (E) and (F) inclination; (G) and (H) angular momentum. The former of each pair is for the system with drift included, while the initial momentum is neutralised in the latter. Both runs started with the asteroid at perihelion, directly opposite Jupiter's IC, initial eccentricity $e = 0.15$ and inital mean motion ratio with Jupiter being $\frac{n_{ast}}{n_{jup}} = 2.50$.



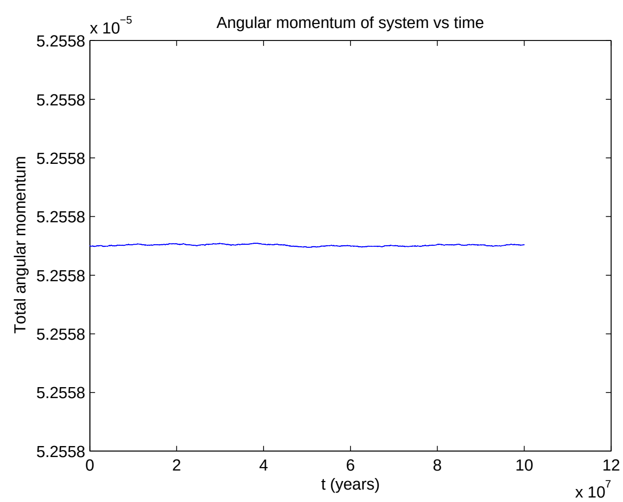
(E)



(F)



(G)



(H)

FIGURE 6. Continuation of previous figure.

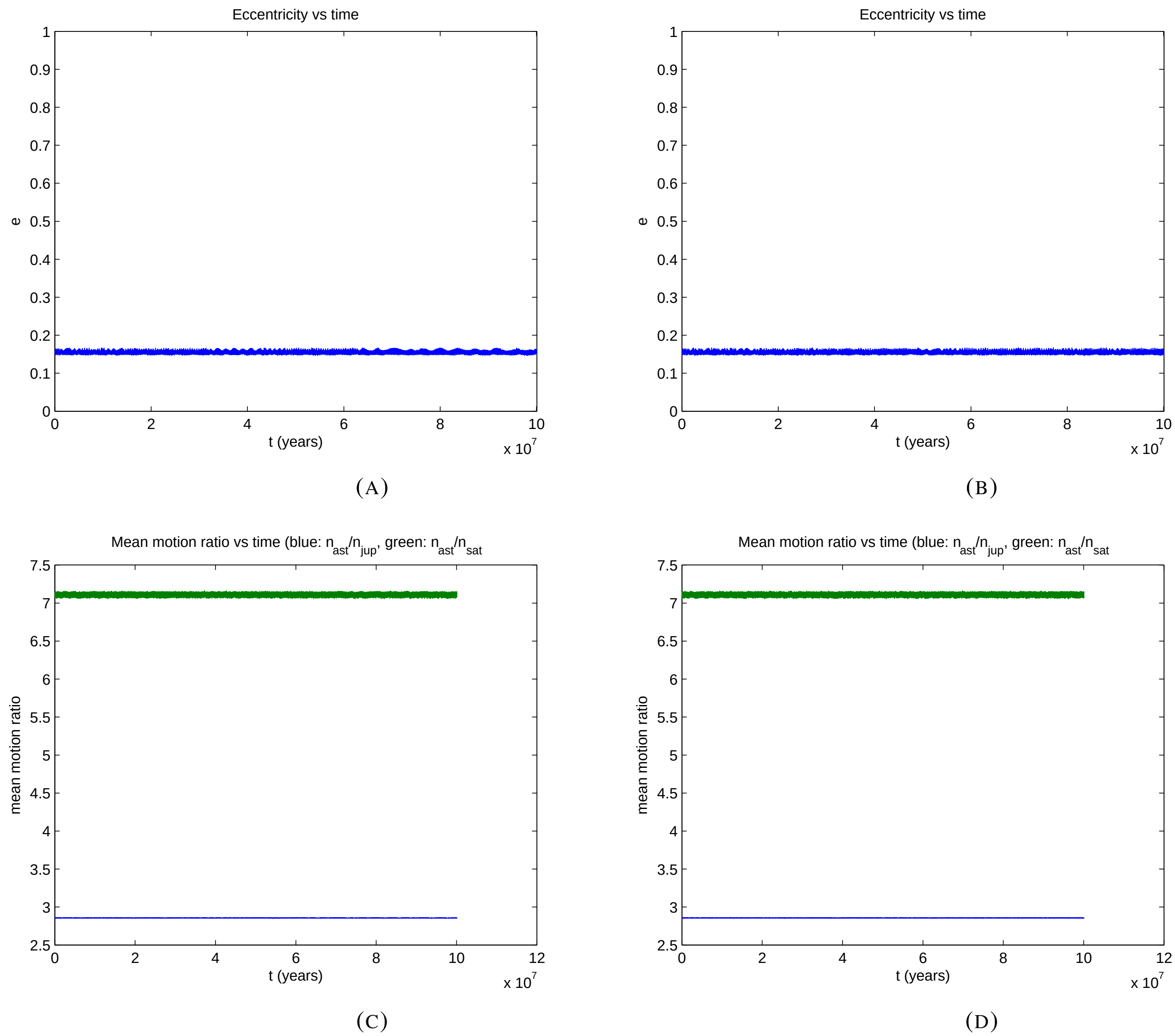
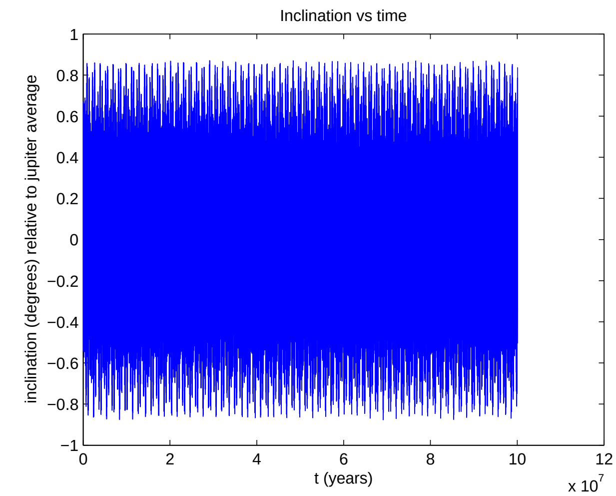
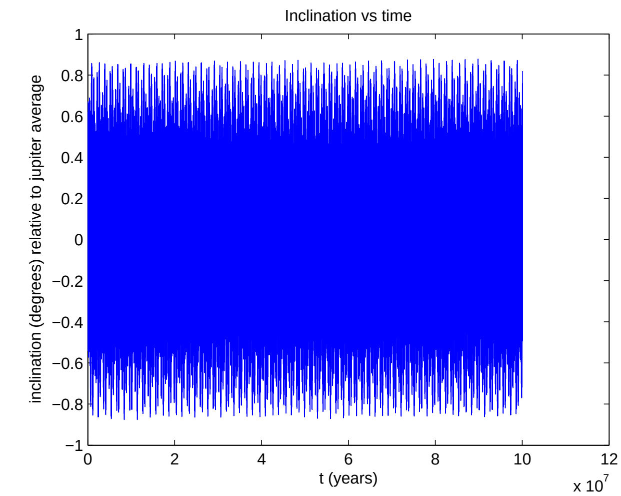


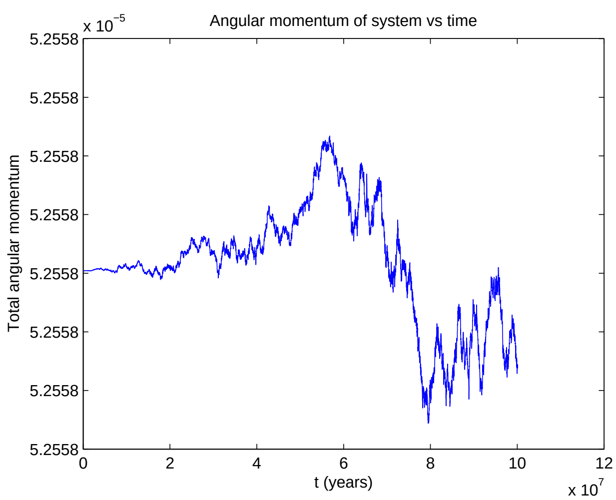
FIGURE 7. Comparison of: (A) and (B) eccentricity; (C) and (D) mean motion ratio of asteroid with Jupiter (blue/lower curve) and Saturn (green/upper curve); (E) and (F) inclination; (G) and (H) angular momentum. The former of each pair is for the system with drift included, while the initial momentum is neutralised in the latter. Both runs started with the asteroid at perihelion, directly opposite Jupiter's IC, initial eccentricity $e = 0.15$ and inital mean motion ratio with Jupiter being $\frac{n_{ast}}{n_{jup}} = 2.85720476458593$.



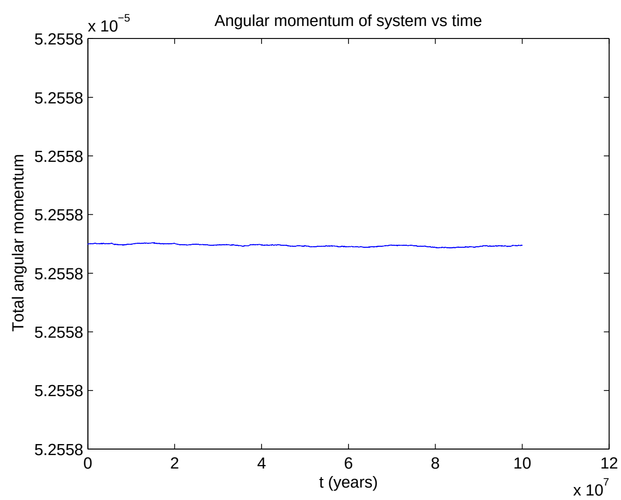
(E)



(F)



(G)



(H)

FIGURE 8. Continuation of previous figure.

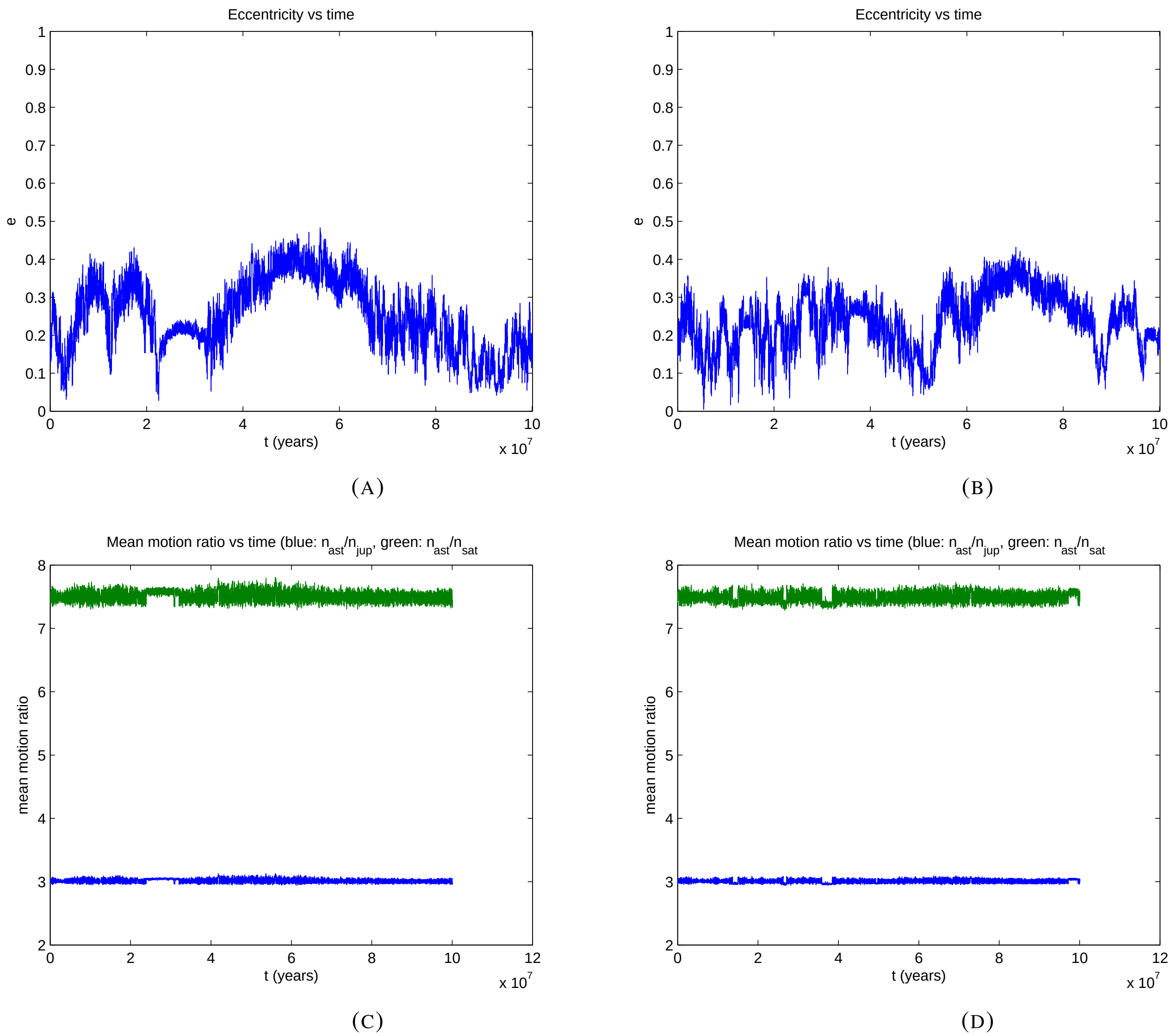
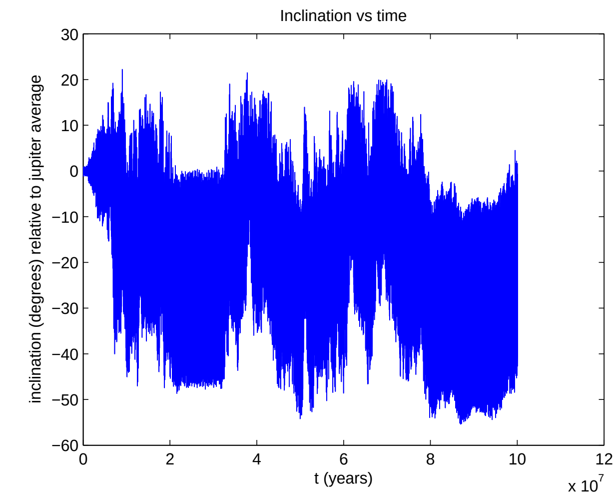
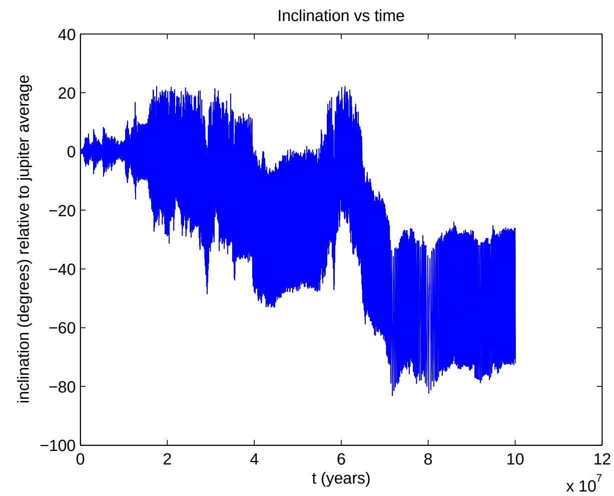


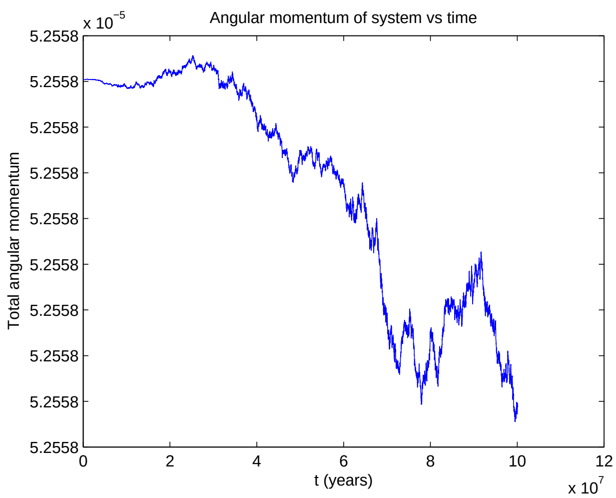
FIGURE 9. Comparison of: (A) and (B) eccentricity; (C) and (D) mean motion ratio of asteroid with Jupiter (blue/lower curve) and Saturn (green/upper curve); (E) and (F) inclination; (G) and (H) angular momentum. The former of each pair is for the system with drift included, while the initial momentum is neutralised in the latter. Both runs started with the asteroid at perihelion, directly opposite Jupiter's IC, initial eccentricity $e = 0.15$ and inital mean motion ratio with Jupiter being $\frac{n_{ast}}{n_{jup}} = 3.00$.



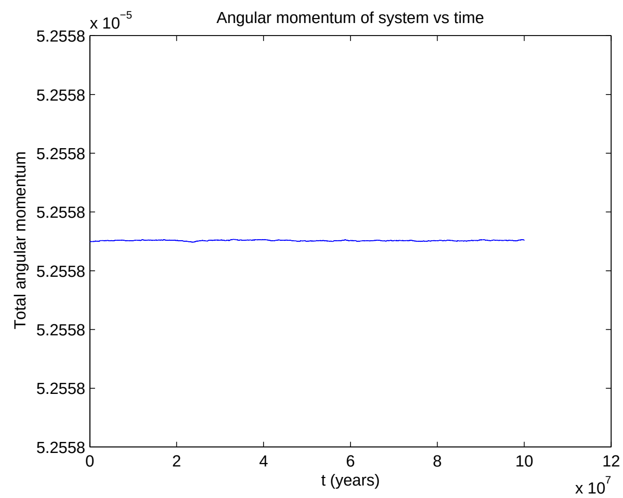
(E)



(F)

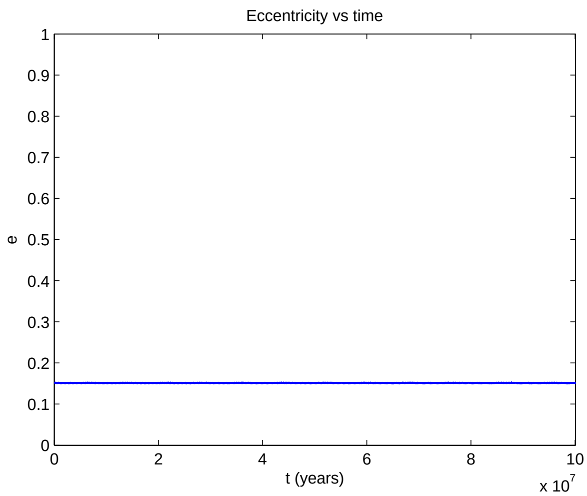


(G)

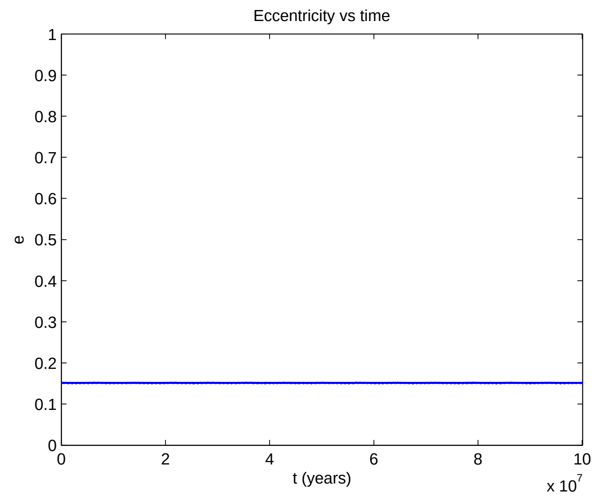


(H)

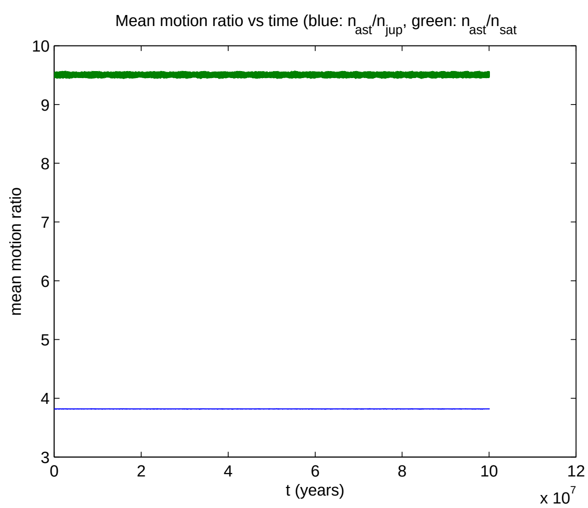
FIGURE 10. Continuation of previous figure.



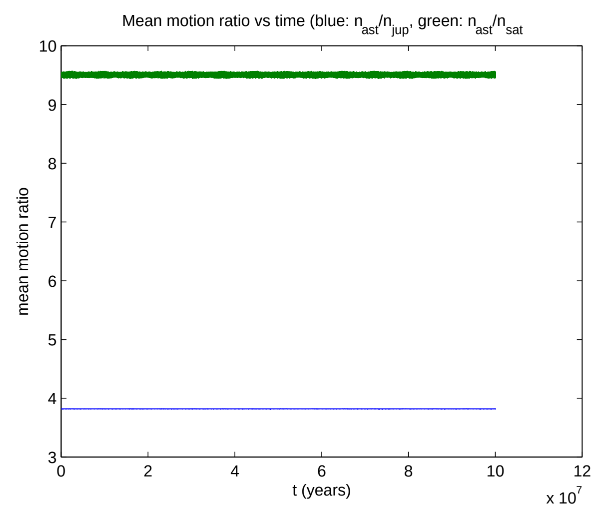
(A)



(B)

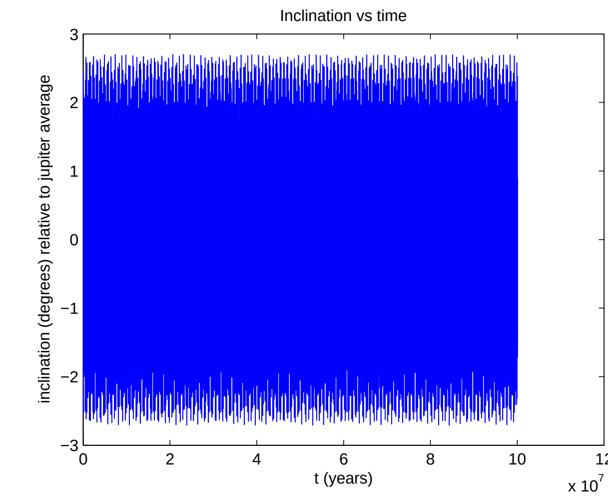


(C)

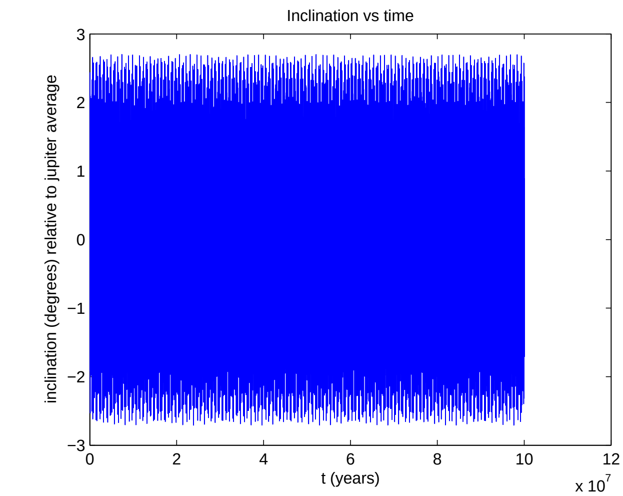


(D)

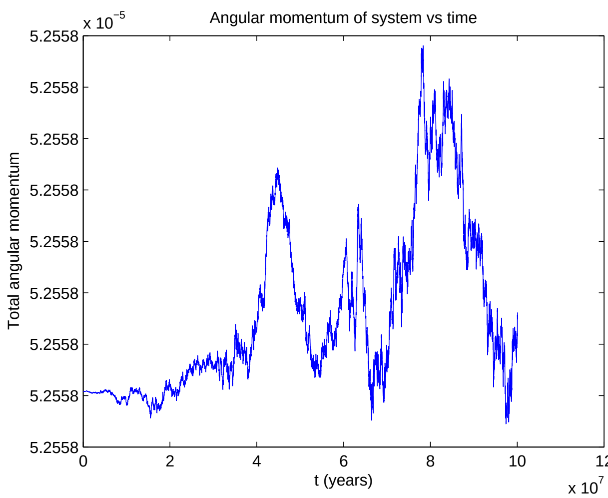
FIGURE 11. Comparison of: (A) and (B) eccentricity; (C) and (D) mean motion ratio of asteroid with Jupiter (blue/lower curve) and Saturn (green/upper curve); (E) and (F) inclination; (G) and (H) angular momentum. The former of each pair is for the system with drift included, while the initial momentum is neutralised in the latter. Both runs started with the asteroid at perihelion, directly opposite Jupiter's IC, initial eccentricity $e = 0.15$ and inital mean motion ratio with Jupiter being $\frac{n_{ast}}{n_{jup}} = 3.82164505322$.



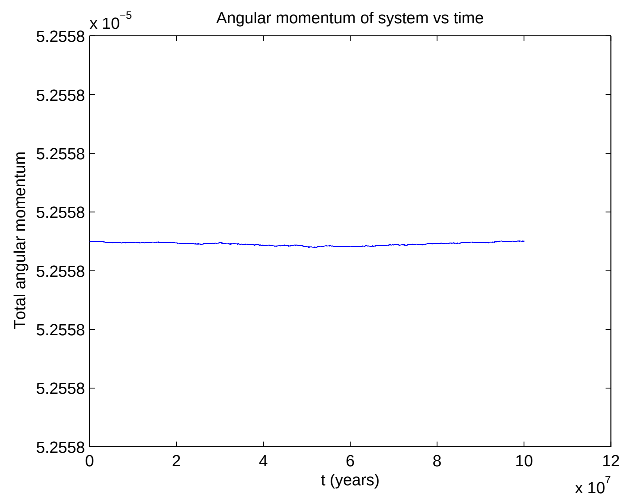
(E)



(F)



(G)



(H)

FIGURE 12. Continuation of previous figure.

Appendix F with the MATLAB and Fortran codes on pages 80 to 134 are available from the Applied Maths Honours coordinator as a separate volume on request.

APPENDIX F

Codes

F1. MATLAB Codes

File: *asteroid_integrate.m*

```
clear;
format long

numplan = 4;    % number of bodies
dim = 3;        % number of spatial dimensions
dt = 1;         % timestep size (days)
N = 365300;

storefrequency = 100;   % frequency with which orbital
                        % data are written to buffer
dumpfrequency = 10; % length of buffer (data dumped each
                    % storefrequency*dumpfrequency steps)

[posscale speedscale] = generatescales(0.15,2);

initial_data;

testerror = 0;

opendat2;

method = 2;% 2 leapfrog, 3 fourth order

a=0;
b=0;
if method == 3
    a = [1/(2*(2-2^(1/3)));(1-2^(1/3))/(2*(2-2^(1/3)));...
         (1-2^(1/3))/(2*(2-2^(1/3)));1/(2*(2-2^(1/3)))];
    b = [1/(2-2^(1/3));-(2^(1/3))/(2-2^(1/3));...
         1/(2-2^(1/3));0];
```

```
end

i_final=ceil(N/storefrequency)*storefrequency;

fprintf(fdetails,...
    '%i\n%i\n%16.16e\n%i\n%i\n',...
    '%16.16e\n%i\n%i\n%i\n%s\n',...
    numplan,dim,dt,i_final,storefrequency,G,...
    testerror,method,dumpfrequency,fpath);

pstore(1:numplan,1:dim,1:dumpfrequency)=0;
qstore(1:numplan,1:dim,1:dumpfrequency)=0;
for i = 1:numplan
    fprintf(fm,'%16.16e\n',m(i));
end
breakrun=false;
for i=0:i_final-1
    if mod(i,storefrequency)== 0    % store to buffer
        pstore(:,:, mod(i,storefrequency*dumpfrequency)/...
            storefrequency+1)=p;
        qstore(:,:, mod(i,storefrequency*dumpfrequency)/...
            storefrequency+1)=q;
        if asteccentricity(p,q,m,G,dim) > 0.8 && ¬breakrun
            breakrun = true;
            fprintf('Eccentricity of asteroid > 0.8\n');
        end
%         fprintf('%i: stored to buffer\n',i);
    end
    if mod(i,storefrequency*dumpfrequency)==...
            dumpfrequency*storefrequency-1
        fprintf(phasecoord(1),'%+16.16e\n',qstore);
        fprintf(phasecoord(2),'%+16.16e\n',pstore);
%         fprintf('%i::f dumped buffer to disk\n',i);
        % clear buffer
        pstore(1:numplan,1:dim,1:dumpfrequency)=0;
        qstore(1:numplan,1:dim,1:dumpfrequency)=0;
%         fprintf('%i: cleared buffer\n',i);
        if breakrun
            i_final = i;
            break
        end
    elseif i == i_final-1
        fprintf(phasecoord(1),'%+16.16e\n',qstore);
        fprintf(phasecoord(2),'%+16.16e\n',pstore);
%         fprintf('%i:f dumped buffer to disk\n',i);
```

```
        % clear buffer
        pstore(1:numplan,1:dim,1:dumpfrequency)=0;
        qstore(1:numplan,1:dim,1:dumpfrequency)=0;
%         fprintf('%i: cleared buffer\n',i);
%         break
    end

    [p q] = integrateorbit3(p,q,m,G,method,dt,a,b);
end
if testerror ==1
%     k1 = mod(i_final,dumpfrequency);
    p = -p;
    fprintf('testerror = true. Reversing flow.\n');
    for i = 0:i_final+storefrequency
        if mod(i,storefrequency)==0 && i≠0 % store to buffer
            pstore(:,:,...
            mod(i-storefrequency,storefrequency*dumpfrequency)/...
            storefrequency+1)=p;
            qstore(:,:,...
            mod(i-storefrequency,storefrequency*dumpfrequency)...
            /storefrequency+1)=q;
            fprintf('%i: stored to buffer\n',...
            i_final+storefrequency-i);
        end
        if mod(i,storefrequency*dumpfrequency)==0 && i≠0
            fprintf(phasecoordr(1),'%+16.16e\n',qstore);
            fprintf(phasecoordr(2),'%+16.16e\n',pstore);
            fprintf('%i::r dumped buffer to disk\n',...
            i_final+storefrequency-i);
            % clear buffer
            pstore(1:numplan,1:dim,1:dumpfrequency)=0;
            qstore(1:numplan,1:dim,1:dumpfrequency)=0;
            k1=0;
%             fprintf('%i:- cleared buffer\n',i);
        elseif i == i_final
            fprintf(phasecoordr(1),'%+16.16e\n',qstore);
            fprintf(phasecoordr(2),'%+16.16e\n',pstore);
            fprintf('%i:r dumped buffer to disk\n',...
            i_final+storefrequency-i);
            % clear buffer
            pstore(1:numplan,1:dim,1:dumpfrequency)=0;
            qstore(1:numplan,1:dim,1:dumpfrequency)=0;
            break
%             fprintf('%i: cleared buffer\n',i);
        end
```

```

        [p q] = integrateorbit3(p,q,m,G,method,dt,a,b);
    end
end
fprintf('Done\n');
fclose('all');
clear;
```

File: *initial_data.m*

```

G = 2.95912208286e-4;
m = [1.00000597682,...
     1e-15,...
     0.000954786104043,0.000285583733151];

vx = [0,...
     0.761576392933587*speedscale,...
     0.00565429,0.00168318...
     ...,0.00354178,0.00288930,0.00276725
     ];
vy = [0,...
     -0.588733316817015*speedscale,...
     -0.00412490,0.00483525...
     ...,0.00137102,0.00114527,-0.00170702
     ];
vz = [0,...
     -0.270914155030523*speedscale,...
     -0.00190589,0.00192462...
     ...,0.00055029,0.00039677,-0.00136504
     ];

qx = [0;...
     -3.5023653*posscale;...
     -3.5023653;9.0755314...
     ...;8.3101420;11.4707666;-15.5387357
     ];
qy = [0;...
     -3.8169847*posscale;...
     -3.8169847;-3.0458353...
     ...;-16.2901086;-25.7294829;-25.2225594
     ];
qz = [0;...
     -1.5507963*posscale;...
     -1.5507963;-1.6483708...
     ...;-7.2521278;-10.8169456;-3.1902382
     ];

% initial momenta
p(1:numplan,1:dim) = 0;
p(:,:) = [vx(:).*m(:),vy(:).*m(:),vz(:).*m(:)];

```

```
% initial positions organised into a matrix
q(1:numplan,1:dim) = 0;
q(:,:) = [qx(:),qy(:),qz(:)];

clear vx vy vz qx qy qz
```

File: *integrateorbit3.m*

```
function [p,q] = integrateorbit3(p,q,m,G,method,dt,a,b)
        if method == 0      % Euler's method
            dU = dpotential(q,G,m);
            v = vel(p,m);
            q = q + dt*v;
            p = p - dt*dU;
        elseif method == 1  % Symplectic Euler
            v = vel(p,m);
            q = q + dt*v;
            dU = dpotential(q,G,m);
            p = p - dt*dU;
        elseif method == 2  % leapfrog
            v = vel(p,m);
            q_ihalf = q + dt/2*v;
            dU = dpotential(q_ihalf,G,m);
            p = p - dt*dU;
            v = vel(p,m);
            q = q_ihalf + dt/2*v;
        elseif method == 3  % 4th order symplectic
            for i = 1:4
                if a(i) ≠ 0
                    v = vel(p,m);
                    q = q + a(i)*dt*v;
                end
                if b(i) ≠ 0
                    dU = dpotential(q,G,m);
                    p = p - b(i)*dt*dU;
                end
            end
        end
end
```

File: *asteccentricity.m*

```
function e = asteccentricity(p,q,m,G,dim)
    r        = q(2,:)-q(1,:);            % relative position
    nr       = sqrt(sum(abs(r).^2));     % magnitude of r
    v(1:dim) = 0;
    v(:)     = p(2,:)/m(2)-p(1,:)/m(1); % relative velocity
    nv       = sqrt(sum(abs(v).^2));     % magnitude of v
    h        = cross(r,v);               % normal vector
    nh       = sqrt(sum(abs(h).^2));     % magnitude of normal

    mu = G*(m(1)+m(2));                  % reduced mass
    a=1./(2./nr - nv.^2/mu);
    e=sqrt(1-nh.^2./(mu*a));             % eccentricity
return
end
```

File: *vel.m*

```
function v = vel(p,m)
    v = p;
    for i = 1:size(p,2)
        v(:,i) = v(:,i)./m(:);
    end
end
```

File: *dpotential.m*

```
function DU = dpotential(q,G,m)
    numplan = size(q,1);
    dim = size(q,2);
    pow = 3/2;
    dU(1:numplan,1:dim,1:numplan) = 0;
    diff(1:numplan,1:dim,1:numplan) = 0;
    denom(1:numplan,1:numplan) = 0;
    for l = 1:numplan
        for j = 1:numplan
            if l == j
                diff(l,:,j) = 0;
                denom(l,j) = 0;
                dU(l,:,j) = 0;
            else
                diff(l,:,j) = -(q(l,:)-q(j,:));
                denom(l,j) = sum(diff(l,:,j).^2,2);
                dU(l,:,j) =...
                 -G*m(l)*m(j)*diff(l,:,j)/denom(l,j)^pow;
            end
        end
    end
    DU = sum(dU,3);
    return
end
```

File: *generatescales.m*

```
function [p s] = generatescales(e, meanmotratio)
% given a desired average eccentricity and average mean motion ratio with
% jupiter (given that the asteroid starts within jupiter's orbit), this
% determines an appropriate pair of scale factors for the asteroid's
% initial conditions (for simplicity having the asteroid start directly on
% the line between jupiter and the sun).

G = 2.95912208286e-4;   % gravitational constant

m = 1.00000597682;      % mass of sun

mu = G*(m+1e-15);       % 1e-15 is the mass of the asteroid

% jupiter's initial state
vjupi = [0.00565429, -0.00412490, -0.00190589];
qjupi = [-3.5023653,-3.8169847,-1.5507963];
hjupi = cross(qjupi,vjupi);

rji = sqrt(sum(qjupi.^2));
vji = sqrt(sum(vjupi.^2));
hji = sqrt(sum(hjupi.^2));

rj = rji;
vj = vji;
hj = hji;

vjup = vj*vjupi/vji;
qjup = rj*qjupi/rji;
hjup = cross(qjup,vjup);

va1 = cross(hjup,qjup);
vmag = sqrt(sum(va1.^2));
vunit = va1/vmag;

averagemeanmotjup = 1.450072902967737e-03;

averagemeanmotast = averagemeanmotjup*meanmotratio;

% semi-major axis of asteroid
a = nthroot(mu/averagemeanmotast^2,3);

hsquared = mu*a*(1-e^2)/hj^2;
```

```
h = sqrt(hsquared);

% pplus = a/rj + sqrt((mu*a)^2 - mu*a*hsquared*hj^2)/(mu*rj);
% splus = (h/pplus)*(hj/rj);
% p = pplus;
% s = splus;

pminus = a/rj - sqrt((mu*a)^2 - mu*a*hsquared*hj^2)/(mu*rj);
sminus = (h/pminus)*(hj/rj);
p = pminus;
s = sminus;

end
```

File: *asteroid_resume_run.m*

```
clear;

resumedat2

% read essential details from file
numplan = fscanf(fdetails,'%i',1);
dim = fscanf(fdetails,'%i',1);
dt = fscanf(fdetails,'%f',1);
N = fscanf(fdetails,'%f',1);
storefrequency = fscanf(fdetails,'%f',1);
G = fscanf(fdetails,'%f',1);
testerror=fscanf(fdetails,'%i',1);
method=fscanf(fdetails,'%i',1);
dumpfrequency=fscanf(fdetails,'%i',1);

fclose(fdetails);

% read masses from file
m(1:numplan) = 0;
for j = 1:numplan
    m(j) = fscanf(fm,'%f',1);
end

fclose(fm);

% get to the last p and q properly recorded to file
q(1:numplan,1:dim) = 0;
p(1:numplan,1:dim) = 0;
qtemp(1:numplan,1:dim) = 0;
ptemp(1:numplan,1:dim) = 0;
k = 0;
eof=false;
while ¬feof(phasecoord(1))
    qtemp = fscanf(phasecoord(1),'%f',[numplan,dim]);
    ptemp = fscanf(phasecoord(2),'%f',[numplan,dim]);

    if size(ptemp,1)≠numplan || size(ptemp,2)≠dim
        fprintf('Last output was incompletely written in p - ');
        position = ftell(phasecoord(1))-numplan*dim*25;
        break
    elseif size(qtemp,1)≠numplan || size(qtemp,2)≠dim
        fprintf('Last output was incompletely written in q - ');
```

```
        position = ftell(phasecoord(2))-numplan*dim*25;
        break
    elseif (sum(sum(ones(numplan,dim)-(qtemp(:,:)==0)))==0 &&...
            sum(sum(ones(numplan,dim)-(ptemp(:,:)==0)))==0)
        fprintf('Reached end of output - ');
        position = ftell(phasecoord(1));
        break
    elseif isempty(qtemp) || isempty(ptemp)
        eof=true;
        fprintf('Reached eof - ');
        position = ftell(phasecoord(1));
        break
    else
        q = qtemp;
        p = ptemp;
    end
    k=k+1;
end
% k = number of recorded timesteps
% numplan*dim*k = number of lines recorded in p,q
% numplan*dim*k*25 = number of characters recorded in p,q
if eof == true
    fseek(phasecoord(1),0,'eof');
    fseek(phasecoord(2),0,'eof');
    fprintf('placing marker at eof\n');
else
    fprintf('placing marker at end of last completed output %i\n',...
            numplan*dim*k*25);
    fseek(phasecoord(1),numplan*dim*k*25,'bof');
    fseek(phasecoord(2),numplan*dim*k*25,'bof');
end

a=0;
b=0;
if method == 3
    a = [1/(2*(2-2^(1/3)));(1-2^(1/3))/(2*(2-2^(1/3)));...
          (1-2^(1/3))/(2*(2-2^(1/3)));1/(2*(2-2^(1/3)))];
    b = [1/(2-2^(1/3));-(2^(1/3))/(2-2^(1/3));...
          1/(2-2^(1/3));0];
end

pstore(1:numplan,1:dim,1:dumpfrequency)=0;
qstore(1:numplan,1:dim,1:dumpfrequency)=0;

k1=mod(k,dumpfrequency);
```

```
i_final = N;

if k*storefrequency < i_final

    % this block progresses us to the next timestep that
    % would be recorded and avoids the calculation going
    % out of phase with an uninterrupted simulation from
    % the original ICs.
    for i = 1:storefrequency
        [p q] = integrateorbit3(p,q,m,G,method,dt,a,b);
    end

    fprintf('Resuming\n');
    breakrun=false;
    for i=k*storefrequency:i_final-1
        if mod(i,storefrequency)== 0     % store current data to buffer
            pstore(:,:,mod(i,storefrequency*dumpfrequency)/...
                    storefrequency+1-k1)=p;
            qstore(:,:,mod(i,storefrequency*dumpfrequency)/...
                    storefrequency+1-k1)=q;
            if asteccentricity(p,q,m,G,dim) > 0.8
                breakrun = true;
                fprintf('Eccentricity of asteroid > 0.8');
            end
        end
        if mod(i,storefrequency*dumpfrequency)==...
                dumpfrequency*storefrequency-1
            for h = 1:dumpfrequency-k1
                for j = 1:dim
                    for l = 1:numplan
                        fprintf(phasecoord(1),'%+16.16e\n',...
                                qstore(l,j,h));
                        fprintf(phasecoord(2),'%+16.16e\n',...
                                pstore(l,j,h));
                    end
                end
            end
            % clear buffer
            pstore(1:numplan,1:dim,1:dumpfrequency)=0;
            qstore(1:numplan,1:dim,1:dumpfrequency)=0;
            if breakrun
                i_final = i;
                break
            end
            k1=0;
```

```
        elseif i == i_final-1
            for h = 1:dumpfrequency-k1
                for j = 1:dim
                    for l = 1:numplan
                        fprintf(phasecoord(1),'%+16.16e\n',...
                                qstore(l,j,h));
                        fprintf(phasecoord(2),'%+16.16e\n',...
                                pstore(l,j,h));
                    end
                end
            end
            %fprintf('%i:f dumped buffer to disk\n',i);
            % clear buffer
            pstore(1:numplan,1:dim,1:dumpfrequency)=0;
            qstore(1:numplan,1:dim,1:dumpfrequency)=0;
%             break
        end

        [p q] = integrateorbit3(p,q,m,G,method,dt,a,b);
    end
    if testerror == 1
        p = -p;
        fprintf('testerror = true. Reversing flow.\n');
        for i = 0:i_final+storefrequency
            if mod(i,storefrequency)==0 && i≠0
                pstore(:,:,...
                mod(i-storefrequency,storefrequency*dumpfrequency)/...
                storefrequency+1)=p;
                qstore(:,:,...
                mod(i-storefrequency,storefrequency*dumpfrequency)/...
                storefrequency+1)=q;
            end
            if mod(i,storefrequency*dumpfrequency)==0 && i≠0
                fprintf(phasecoordr(1),'%+16.16e\n',qstore);
                fprintf(phasecoordr(2),'%+16.16e\n',pstore);
                % clear buffer
                pstore(1:numplan,1:dim,1:dumpfrequency)=0;
                qstore(1:numplan,1:dim,1:dumpfrequency)=0;
                k1=0;
            elseif i == i_final
                fprintf(phasecoordr(1),'%+16.16e\n',qstore);
                fprintf(phasecoordr(2),'%+16.16e\n',pstore);
                fprintf('%i:r dumped buffer to disk\n',...
                        i_final+storefrequency-i);
                % clear buffer
```

```
                pstore(1:numplan,1:dim,1:dumpfrequency)=0;
                qstore(1:numplan,1:dim,1:dumpfrequency)=0;
                break
%                 fprintf('%i: cleared buffer\n',i);
            end

            [p q] = integrateorbit3(p,q,m,G,method,dt,a,b);
        end
    end
else
    fprintf('No need to resume forward run\n');
    pforwardfinal = p;
    qforwardfinal = q;
    if testerror == 1
        fprintf('testerror = 1. Testing to possibly resume reverse run.\n');
        % get to the last p and q properly recorded to file
        q(1:numplan,1:dim) = 0;
        p(1:numplan,1:dim) = 0;
        qtemp(1:numplan,1:dim) = 0;
        ptemp(1:numplan,1:dim) = 0;
        k = 0;
        eof=false;
        while ¬feof(phasecoord(1))
            qtemp = fscanf(phasecoordr(1),'%f',[numplan,dim]);
            ptemp = fscanf(phasecoordr(2),'%f',[numplan,dim]);

            if size(ptemp,1)≠numplan || size(ptemp,2)≠dim
                fprintf('Last output was incompletely written in p - ');
                position = ftell(phasecoordr(1))-numplan*dim*25;
                break
            elseif size(qtemp,1)≠numplan || size(qtemp,2)≠dim
                fprintf('Last output was incompletely written in q - ');
                position = ftell(phasecoordr(2))-numplan*dim*25;
                break
            elseif (sum(sum(ones(numplan,dim)-...
                    (qtemp(numplan,dim)==0)))==0 &&...
                    sum(sum(ones(numplan,dim)-...
                    (ptemp(numplan,dim)==0)))==0
                fprintf('Reached end of output - ');
                position = ftell(phasecoordr(1));
                break
            elseif isempty(qtemp) || isempty(ptemp)
                eof=true;
                fprintf('Reached eof - ');
                position = ftell(phasecoordr(1));
```

```
                break
            else
                q = qtemp;
                p = ptemp;
            end
            k=k+1;
        end
        % k = number of recorded timesteps
        % numplan*dim*k = number of lines recorded in p,q
        % numplan*dim*k*25 = number of characters recorded in p,q
        if eof == true
            fseek(phasecoordr(1),0,'eof');
            fseek(phasecoordr(2),0,'eof');
            fprintf('placing marker at eof\n');
        else
            fprintf(...
            'placing marker at end of last completed output %i\n',...
            numplan*dim*k*25);
            fseek(phasecoordr(1),numplan*dim*k*25,'bof');
            fseek(phasecoordr(2),numplan*dim*k*25,'bof');
        end

        a=0;
        b=0;
        if method == 3
            a = [1/(2*(2-2^(1/3)));(1-2^(1/3))/(2*(2-2^(1/3)));...
                 (1-2^(1/3))/(2*(2-2^(1/3)));1/(2*(2-2^(1/3)))];
            b = [1/(2-2^(1/3));-(2^(1/3))/(2-2^(1/3));...
                 1/(2-2^(1/3));0];
        end

        pstore(1:numplan,1:dim,1:dumpfrequency)=0;
        qstore(1:numplan,1:dim,1:dumpfrequency)=0;

        k1=mod(k,dumpfrequency);
        i_final = N;

        if k*storefrequency < i_final
            fprintf('Resuming reverse run\n');
            breakrun=false;
            firstwriteiteration=true;

            if k == 0
                for i = 1:storefrequency
                    [pforwardfinal qforwardfinal] =...
```

```
        integrateorbit3(pforwardfinal,qforwardfinal,...
            m,G,method,dt,a,b);
    end
    p=-pforwardfinal;
    q=qforwardfinal;
else
    % this block progresses us to the next
    % timestep that would be recorded and
    % avoids the calculation going out of
    % phase with an uninterrupted simulation
    % from the original ICs.
    for i = 1:storefrequency
        [p q] =...
         integrateorbit3(p,q,m,G,method,dt,a,b);
    end
end
for i = k*storefrequency:i_final+storefrequency
    if (mod(i,storefrequency)==0 && i≠0)
        pstore(:,:,mod(i-storefrequency,...
        storefrequency*dumpfrequency)/...
        storefrequency+1)=p;
        qstore(:,:,mod(i-storefrequency,...
        storefrequency*dumpfrequency)/...
        storefrequency+1)=q;
    end
    if firstwriteiteration && mod(i,storefrequency*...
        dumpfrequency)==0
        for h = mod(i-storefrequency,...
            storefrequency*dumpfrequency)/...
            storefrequency:dumpfrequency
            for j = 1:dim
                for l = 1:numplan
                    fprintf(phasecoordr(1),'%+16.16e\n',...
                            qstore(l,j,h));
                    fprintf(phasecoordr(2),'%+16.16e\n',...
                            pstore(l,j,h));
                end
            end
        end
        fprintf('%i::r dumped buffer to disk\n',...
                i_final+storefrequency-i);
        % clear buffer
        pstore(1:numplan,1:dim,1:dumpfrequency)=0;
        qstore(1:numplan,1:dim,1:dumpfrequency)=0;
        k1=0;
```

```
                    firstwriteiteration=false;
                elseif (mod(i,storefrequency*dumpfrequency)==0 && i≠0)
                    for h = 1:dumpfrequency
                        for j = 1:dim
                            for l = 1:numplan
                                fprintf(phasecoordr(1),'%+16.16e\n',...
                                        qstore(l,j,h));
                                fprintf(phasecoordr(2),'%+16.16e\n',...
                                        pstore(l,j,h));
                            end
                        end
                    end
                    fprintf('%i::r dumped buffer to disk\n',...
                            i_final+storefrequency-i);
                    % clear buffer
                    pstore(1:numplan,1:dim,1:dumpfrequency)=0;
                    qstore(1:numplan,1:dim,1:dumpfrequency)=0;
                    k1=0;
                elseif i == i_final
                    for h = 1:dumpfrequency-k1
                        for j = 1:dim
                            for l = 1:numplan
                                fprintf(phasecoordr(1),'%+16.16e\n',...
                                        qstore(l,j,h));
                                fprintf(phasecoordr(2),'%+16.16e\n',...
                                        pstore(l,j,h));
                            end
                        end
                    end
                    fprintf('%i:r dumped buffer to disk\n',...
                            i_final+storefrequency-i);
                    % clear buffer
                    pstore(1:numplan,1:dim,1:dumpfrequency)=0;
                    qstore(1:numplan,1:dim,1:dumpfrequency)=0;
                    break
                end

                [p q] = integrateorbit3(p,q,m,G,method,dt,a,b);
            end
        else
            fprintf('No need to resume reverse run\n');
        end
    end
end
fprintf('Done\n');
```

```
fclose('all');
% clear;
```

File: *asteroid_compare_runs.m*

```
clear;

readdat2
readdatcomp
[numplan(1) dim(1) dt(1) N(1) storefrequency(1)...
 G(1) err(1) method(1) dumpfrequency(1) Nmax(1)...
  m(1,:) t(1,:)] = getdetails(fm,fdetails);
[numplan(2) dim(2) dt(2) N(2) storefrequency(2)...
 G(2) err(2) method(2) dumpfrequency(2) Nmax(2)...
  m(2,:) t(2,:)] = getdetails(fm2,fdetails2);

if dim(1) ≠ dim(2)
    error('spatial dimensions unequal');
end
dim = dim(1);

if t(1,Nmax(1)) ≠ t(2,Nmax(2))
    error('runs are for different amounts of time');
else
    equalsteps = false;
    if Nmax(1) == Nmax(2)
        dt = dt(1);
        N = N(1);
        storefrequency = storefrequency(1);
        equalsteps = 2;
    end
end

if G(1) ≠ G(2)
    error('G does not match between runs');
end
G = G(1);
if err(1) ≠ err(2)
    fprintf('one run reverses, other does not');
    err = 1;
else
    err = err(1);
end


[p q h hsys T U a e inclination trueanom argperi...
    ascnode meanmot com vcom] = extract(phasecoord,...
```

```
    numplan(1), dim, dt(1), G, Nmax(1), m(1,:));
if equalsteps
    [p2 q2 h2 hsys2 T2 U2 a2 e2 inclination2...
     trueanom2 argperi2 ascnode2 meanmot2 com2 vcom2] =...
     extract(phasecoord2, numplan(2), dim, dt(1), G, Nmax,...
     m(2,:));
else
    [p2 q2 h2 hsys2 T2 U2 a2 e2 inclination2...
    trueanom2 argperi2 ascnode2 meanmot2 com2 vcom2] =...
    extract(phasecoord2, numplan(2), dim, dt(2), G, Nmax(2),...
    m(2,:));
end

if err == 2
    [pr qr hr hsysr Tr Ur ar er inclinationr trueanomr...
     argperir ascnoder meanmotr  comr vcomr] =...
     extract(phasecoordr, numplan(1), dim, dt(1), G,...
             Nmax(1), m(1,:));
    if equalsteps
        [pr2 qr2 hr2 hsysr2 Tr2 Ur2 ar2 er2 inclinationr2...
         trueanomr2 argperir2 ascnoder2 meanmotr2 comr2...
         vcomr2] = extract(phasecoordr2, numplan(2),...
                           dim, dt, G, Nmax, m(2,:));
    else
        [pr2 qr2 hr2 hsysr2 Tr2 Ur2 ar2 er2 inclinationr2...
         trueanomr2 argperir2 ascnoder2 meanmotr2...
         comr2 vcomr2] = extract(phasecoordr2, numplan(2),...
                         dim, dt(2), G, Nmax(2), m(2,:));
    end
end

figure(1)
plot(t,e(2,:)-e2(2,:),t,e(3,:)-e2(3,:))
if err == 2
    hold on
    plot(t,er(2,Nmax:-1:1)-er2(2,Nmax:-1:1),'r',t,...
         er(3,Nmax:-1:1)-er2(3,Nmax:-1:1),'m')
end
xlabel('t (years)');
ylabel('e1 - e2');
figure(2)
plot(t,a(2,:)-a2(2,:),t,a(3,:)-a2(3,:))
if err == 2
    hold on
    plot(t,ar(2,Nmax:-1:1)-ar2(2,Nmax:-1:1),'r',...
```

```
        t,ar(3,Nmax:-1:1)-ar2(3,Nmax:-1:1),'m')
end
xlabel('t (years)');
ylabel('a1 - a2');
figure(3)
plot(t,meanmot(2,:)./meanmot(3,:)-...
       meanmot2(2,:)./meanmot2(3,:))
if err == 2
    hold on
    plot(t,meanmot(2,Nmax:-1:1)./meanmotr(3,Nmax:-1:1)-...
         meanmotr2(2,Nmax:-1:1)./meanmotr2(3,Nmax:-1:1),'r')
end
xlabel('t (years)');
ylabel('mean motion: asteroid1/jupiter1 - asteroid2/jupiter2');
figure(4)
plot(t,T+U-T2-U2)
if err == 2
    hold on
    plot(t,Tr(Nmax:-1:1)+Ur(Nmax:-1:1)-...
           (Tr2(Nmax:-1:1)+Ur2(Nmax:-1:1)),'r')
end
xlabel('t (years)');
ylabel('Hamiltonian1 - Hamiltonian2');
figure(5)
plot(t,sqrt(sum((hsys).^2,1))-sqrt(sum((hsys2).^2,1)))
if err == 2
    hold on
    plot(t,sqrt(sum((hsysr(:,Nmax:-1:1)).^2,1))...
           -sqrt(sum((hsysr2(:,Nmax:-1:1)).^2,1)),'r')
end
xlabel('t (years)');
ylabel('Total angular momentum1 - total angular momentum2');
figure(6)
qast(:,:) = q(2,:,:);
qast = qast - com;
qast2(:,:) = q2(2,:,:);
qast2 = qast2 - com2;
plot3(qast(1,:),qast(2,:),qast(3,:),'.',...
      qast2(1,:),qast2(2,:),qast2(3,:),'.')
axis equal
grid on
%axis square
if err == 2
    qastr(:,:) = qr(2,:,:);
    qastr = qastr - comr;
```

```
    hold on
    plot3(qast(1,1),qast(2,1),qast(3,1),'ro',...
          qast2(1,1),qast2(2,1),qast2(3,1),'go')
    plot3(qastr(1,Nmax),qastr(2,Nmax),qastr(3,Nmax),'r*',...
          qastr(1,Nmax),qastr(2,Nmax),qastr(3,Nmax),'g*')
end
if err == 2
    fprintf('q_ast_start - q_ast_finish = %16.16f\n',...
            sqrt(sum(q(2,:,1).^2-qr(2,:,Nmax).^2,2)));
    fprintf('q_jup_start - q_jup_finish = %16.16f\n',...
            sqrt(sum(q(3,:,1).^2-qr(3,:,Nmax).^2,2)));
    fprintf('q_ast_start2 - q_ast_finish2 = %16.16f\n',...
            sqrt(sum(q2(2,:,1).^2-qr2(2,:,Nmax).^2,2)));
    fprintf('q_jup_start2 - q_jup_finish2 = %16.16f\n',...
            sqrt(sum(q2(3,:,1).^2-qr2(3,:,Nmax).^2,2)));
end
fclose('all');
% clear;
```

File: *asteroid_plot.m*

```
clear;

readdat2
[numplan dim dt N storefrequency G err method...
    dumpfrequency Nmax m t] = getdetails(fm,fdetails);
[p q h hsys T U a e inclination trueanom...
    argperi ascnode meanmot com vcom endi] =...
    extract(phasecoord, numplan, dim, dt, G, Nmax, m);

if endi ≠ Nmax
    fprintf('run terminated at step %i, not step %i\n',...
            endi*storefrequency,Nmax*storefrequency)
    if err == 2
        err = 0;
    end
else
    if err == 2
        [pr(:,:,Nmax:-1:1) qr(:,:,Nmax:-1:1)...
         hr(:,:,Nmax:-1:1) hsysr(:,Nmax:-1:1)...
         Tr(Nmax:-1:1) Ur(Nmax:-1:1) ar(:,Nmax:-1:1)...
         er(:,Nmax:-1:1) inclinationr(:,Nmax:-1:1)...
         trueanomr(:,Nmax:-1:1) argperir(:,Nmax:-1:1)...
         ascnoder(:,Nmax:-1:1) meanmotr(:,Nmax:-1:1)...
         comr(:,Nmax:-1:1) vcomr(:,Nmax:-1:1) endi] =...
         extract(phasecoordr, numplan, dim, dt, G, Nmax, m);
    end

    if endi≠ Nmax
        fprintf(...
        'reverse run terminated at step %i, not step %i\n',...
        endi*storefrequency,Nmax*storefrequency)
    end
end

figure(1)
plot(t,e(2,:),t,e(3,:))
if err == 2
    hold on
    plot(t,er(2,:),'r',t,er(3,:),'m')
end
axis([0 max(t) 0 1])
xlabel('t (years)');
```

```
ylabel('e');
figure(2)
plot(t,a(2,:),t,a(3,:))
if err == 2
    hold on
    plot(t,ar(2,:),'r',t,ar(3,:),'m')
end
xlabel('t (years)');
ylabel('a (AU)');
axis([0 max(t) 0 6])
figure(3)
plot(t,meanmot(2,:)./meanmot(3,:),...
     t,meanmot(2,:)./meanmot(4,:))
if err == 2
    hold on
    plot(t,meanmotr(2,:)./meanmotr(3,:),'r',...
         t,meanmotr(2,:)./meanmotr(4,:),'m')
end
xlabel('t (years)');
ylabel('mean motion: asteroid/jupiter');
figure(4)
plot(t,-(inclination(2,:)-...
     mean(inclination(3,:),2))*180/pi)
if err == 2
    hold on
    plot(t,(inclinationr(2,:)-...
         mean(inclinationr(3,:),2))*180/pi,'r')
end
xlabel('t (years)');
ylabel('inclination (degrees) relative to jupiter mean');
figure(5)
plot(t,T+U)
if err == 2
    hold on
    plot(t,Tr+Ur,'r')
end
xlabel('t (years)');
ylabel('Hamiltonian');
figure(6)
plot(t,sqrt(sum((hsys).^2,1)))
if err == 2
    hold on
    plot(t,sqrt(sum((hsysr).^2,1)),'r')
end
xlabel('t (years)');
```

```
ylabel('Total angular momentum');
figure(7)
qast(:,:) = q(2,:,:);
qast = qast - com;
qjup(:,:) = q(3,:,:);
qjup = qjup - com;
qsun(:,:) = q(1,:,:);
qsun = qsun - com;
plot3(qast(1,:),qast(2,:),qast(3,:),'.',...
      qjup(1,:),qjup(2,:),qjup(3,:),'.',...
      qsun(1,:),qsun(2,:),qsun(3,:),'.')
axis equal
grid on
%axis square
if err == 2
    qastr(:,:) = qr(2,:,:);
    qastr = qastr - comr;
    hold on
    plot3(qast(1,1),qast(2,1),qast(3,1),'ro')
    plot3(qastr(1,1),qastr(2,1),qastr(3,1),'r*')
end
if err == 2
    fprintf('q_ast_start - q_ast_finish = %16.16f\n',...
            abs(sqrt(sum(q(2,:,1).^2-qr(2,:,1).^2,2))));
    fprintf('q_jup_start - q_jup_finish = %16.16f\n',...
            abs(sqrt(sum(q(3,:,1).^2-qr(3,:,1).^2,2))));
end
fclose('all');
% clear;
```

File: *getdetails.m*

```
function [numplan dim dt N storefrequency G...
          err method dumpfrequency Nmax m t] =...
          getdetails(fm,fdetails)

numplan = fscanf(fdetails,'%i',1);
dim = fscanf(fdetails,'%i',1);
dt = fscanf(fdetails,'%f',1);
N = fscanf(fdetails,'%f',1);
storefrequency = fscanf(fdetails,'%f',1);
G = fscanf(fdetails,'%f',1);
err = 1+fscanf(fdetails,'%i',1);
method = fscanf(fdetails,'%i',1);
dumpfrequency = fscanf(fdetails,'%i',1);
Nmax = floor(N/storefrequency);
t = (0:dt*storefrequency:...
     (Nmax-1)*dt*storefrequency)/365.25;
m(1:numplan) = 0;

for j = 1:numplan
    m(j) = fscanf(fm,'%f',1);
end
```

File: *extract.m*

```
function [p q h hsys T U a e inclination trueanom...
          argperi ascnode meanmot com vcom endi] =...
          extract(phasecoord, numplan, dim, dt, G, Nmax, m)


q(1:numplan,1:dim,1:Nmax) = 0;
p(1:numplan,1:dim,1:Nmax) = 0;
orbitalels(1:numplan,1:6,1:Nmax)=0;
e(1:numplan,1:Nmax) = 0;
a(1:numplan,1:Nmax) = 0;
inclination(1:numplan,1:Nmax) = 0;
ascnode(1:numplan,1:Nmax) = 0;
argperi(1:numplan,1:Nmax) = 0;
trueanom(1:numplan,1:Nmax) = 0;
T(1:Nmax)=0;
U(1:Nmax)=0;

```

```
qempty = false;
pempty = false;

for i = 1:Nmax
    if pempty || qempty
        if i == 2
            break
        end
        for l = i-1:Nmax
            q(:,:,l) = q(:,:,i-2);
            p(:,:,l) = p(:,:,i-2);
            T(l) = T(i-2);
            U(l) = U(i-2);
            orbitalels(:,:,l) = orbitalels(:,:,i-2);
        end
        break
    end
    for k = 1:dim
        for j = 1:numplan
            if ¬qempty
                qjkitemp = fscanf(phasecoord(1),'%f',1);
            end
            if ¬isempty(qjkitemp)
                q(j,k,i) = qjkitemp;
            else
                qempty = true;
            end
            if ¬pempty
                pjkitemp = fscanf(phasecoord(2),'%f',1);
            end
            if ¬isempty(pjkitemp)
                p(j,k,i) = pjkitemp;
            else
                pempty = true;
            end
        end
    end

    T(i) = sum((sum(p(:,:,i).^2,2))./m(:)/2);
    U(i) = potential(G,m,q(:,:,i));
    orbitalels(:,:,i)=orbitalels3(p(:,:,i),q(:,:,i),m,G,numplan,dim);
end

if i == Nmax
    endi = Nmax;
```

```
else
    endi = i-2;
end

e(:,:) = orbitalels(:,1,:);
a(:,:) = orbitalels(:,2,:);
inclination(:,:) = orbitalels(:,3,:);
trueanom(:,:) = orbitalels(:,4,:);
argperi(:,:) = orbitalels(:,5,:);
ascnode(:,:) = orbitalels(:,6,:);

meanmot(1:numplan,1:Nmax) = 0;
for i = 1:numplan
    meanmot(i,:) = sqrt(G.*(m(1)+m(i))./a(i,:).^3);
end

% centre of mass
if numplan≥4
    com(1:dim,1:Nmax) = (q(1,:,:)*m(1)+...
        q(2,:,:)*m(2)+q(3,:,:)*m(3)+...
        q(4,:,:)*m(4))/sum(m);
else
    com(1:dim,1:Nmax) = (q(1,:,:)*m(1)+...
        q(2,:,:)*m(2)+q(3,:,:)*m(3))/sum(m);
end

% velocity of ventre of mass
vcom(1:dim,1:Nmax) = 0;
for i = 1:Nmax-1
    vcom(1:3,i) = (com(:,i+1)-com(:,i))/dt;
end
vcom(:,Nmax) = vcom(:,Nmax-1);

% positions and velocities relative to com
qrelcom1(1:dim,1:Nmax) = 0;
qrelcom2(1:dim,1:Nmax) = 0;
qrelcom3(1:dim,1:Nmax) = 0;
if numplan≥4
    qrelcom4(1:dim,1:Nmax) = 0;
end
vrelcom1(1:dim,1:Nmax) = 0;
vrelcom2(1:dim,1:Nmax) = 0;
vrelcom3(1:dim,1:Nmax) = 0;
if numplan≥4
    vrelcom4(1:dim,1:Nmax) = 0;
```

```
end
for j = 1:dim
    for k = 1:Nmax
        qrelcom1(j,k) = q(1,j,k) - com(j,k);
        qrelcom2(j,k) = q(2,j,k) - com(j,k);
        qrelcom3(j,k) = q(3,j,k) - com(j,k);
        if numplan≥4
            qrelcom4(j,k) = q(4,j,k) - com(j,k);
        end
        vrelcom1(j,k) = p(1,j,k)./m(1) - vcom(j,k);
        vrelcom2(j,k) = p(2,j,k)./m(2) - vcom(j,k);
        vrelcom3(j,k) = p(3,j,k)./m(3) - vcom(j,k);
        if numplan≥4
            vrelcom4(j,k) = p(4,j,k)./m(4) - vcom(j,k);
        end
    end
end

% angular momentum of each body
h(1,:,:) = m(1)*cross(qrelcom1,vrelcom1);
h(2,:,:) = m(2)*cross(qrelcom2,vrelcom2);
h(3,:,:) = m(3)*cross(qrelcom3,vrelcom3);
if numplan≥4
    h(4,:,:) = m(4)*cross(qrelcom4,vrelcom4);
end

hsys = squeeze(sum(h,1));
end
```

File: *orbitalels3.m*

```
function orbitalels = orbitalels3(p,q,m,G,numplan,dim)
orbitalels(1:numplan,1:6)=0;
for i = 1:numplan
    r(1:dim)=q(i,:)-q(1,:);      % relative position
    nr=sqrt(sum(abs(r).^2));     % magnitude of r
    v(1:dim)=0;
    v(:)=p(i,:)/m(i)-p(1,:)/m(1); % relative velocity
    nv=sqrt(sum(abs(v).^2));     % magnitude of v
    h(1:dim)= cross(r,v);        % normal vector of orbit
    nh=sqrt(sum(abs(h).^2));     % magnitude of normal

    %%%%%%%%%% FROM MURRAY & DERMOTT %%%%%%%%%
    mu = G*(m(1)+m(i));          % reduced mass
    a=1./(2./nr - nv.^2/mu);     % semi-major axis
    e=sqrt(1-nh.^2./(mu*a));     % eccentricity
    inclination=acos(h(3)./nh);
    ascnode=asin(h(1)./(nh.*sin(inclination)));
    if h(3)<0
        ascnode=-ascnode;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%% FROM WIKIPEDIA %%%%%%%%%%%%
    evec=cross(v,h)/mu-r/nr;
    nevec=sqrt(sum(evec.^2));
    nvec=[cos(ascnode);sin(ascnode);0*ascnode];
    nnvec=sqrt(sum(nvec.^2,1));
    argperi=acos(dot(nvec,evec)./(nevec.*nnvec));
    if evec(3)<0
        argperi=2*pi-argperi;
    end
    trueanom=acos(dot(evec,r)./(nevec.*nr));
    if dot(r(:),v(:))<0
        trueanom=2*pi-trueanom;
    end

    orbitalels(i,1)=e;
    orbitalels(i,2)=a;
    orbitalels(i,3)=inclination;
    orbitalels(i,4)=trueanom;
    orbitalels(i,5)=argperi;
    orbitalels(i,6)=ascnode;
end
```

```
return
end
```

File: *opendat2.m*

```
fpath = [input('enter name of directory to contain data files: ','s') '/'];

mkdir(fpath);

fq = fopen([fpath 'q.dat'], 'w');
fp = fopen([fpath 'p.dat'], 'w');

phasecoord = [fq fp];

fm = fopen([fpath 'm.dat'],'w');

if testerror == 1
    fqr = fopen([fpath 'qr.dat'], 'w');
    fpr = fopen([fpath 'pr.dat'], 'w');

    phasecoordr = [fqr fpr];
end

fdetails = fopen([fpath 'integrationdetails.dat'],'w');
```

File: *readdat2.m*

```
fpath = [input('enter name of directory containing data files: ','s') '/'];

fq = fopen([fpath 'q.dat'], 'r');
fp = fopen([fpath 'p.dat'], 'r');

phasecoord = [fq fp];

fm = fopen([fpath 'm.dat'],'r');

fqr = fopen([fpath 'qr.dat'], 'r');
fpr = fopen([fpath 'pr.dat'], 'r');

phasecoordr = [fqr fpr];

fdetails = fopen([fpath 'integrationdetails.dat'],'r');
```

File: *readdatcomp.m*

```
fpath2 =[input...
  ('enter name of directory containing comparison data files: ','s') '/'];

fq2 = fopen([fpath2 'q.dat'], 'r');
fp2 = fopen([fpath2 'p.dat'], 'r');

phasecoord2 = [fq2 fp2];

fm2 = fopen([fpath2 'm.dat'],'r');

fqr2 = fopen([fpath2 'qr.dat'], 'r');
fpr2 = fopen([fpath2 'pr.dat'], 'r');

phasecoordr2 = [fqr2 fpr2];

fdetails2 = fopen([fpath2 'integrationdetails.dat'],'r');
```

File: *resumedat2.m*

```
fpath = [input('enter name of directory containing data files: ','s') '/'];

fq = fopen([fpath 'q.dat'], 'r+');
fp = fopen([fpath 'p.dat'], 'r+');

phasecoord = [fq fp];

fm = fopen([fpath 'm.dat'],'r');

fqr = fopen([fpath 'qr.dat'], 'r+');
fpr = fopen([fpath 'pr.dat'], 'r+');

phasecoordr = [fqr fpr];

fdetails = fopen([fpath 'integrationdetails.dat'],'r');
```

F2. Fortran Integrator

File: *asteroid.f90*

```
module globals
    integer*8 :: numplan, dimensions, storefrequency, dumpfrequency
end module globals

program asteroid
    use globals

    integer*8 :: testerror, order, coeffs, method
    integer*8 :: N, i
        ! numplan: number of planets
        ! dimensions: number of spatial dimensions
        ! N: total number of timesteps
        ! storefrequency: determines which timesteps are stored
        ! dumpfrequency: size of buffer
        ! testerror: whether to integrate backwards in time
        ! order: order of accuracy
        ! coeffs: number of integration coefficients
    double precision :: G, dt, eccentricity, meanmotionratio, e_ast
        ! G: newton's gravitional constant
        ! dt: step size
        ! eccentricity: desired initial eccentricity of asteroid
        ! meanmotionratio: desired initial mean motion ratio of
        !            asteroid and jupiter
    double precision, dimension(2) :: scales
        ! position and speed scales for asteroid's ICs
    double precision, allocatable :: a(:), b(:)
        ! integration coefficients
    double precision, dimension(4) :: m     ! masses
    double precision, dimension(4) :: vx, vy, vz, qx, qy, qz
        ! initial condition arrays: x, y, z per planet
    double precision, dimension(4,3) :: lp, lq
        ! last written p and q for resuming
    integer*8 :: li, k1
        ! li: last fully written integration step for resuming
        ! k: line number in file
        ! k1: calculating how much to write to file when resuming
    double precision, allocatable :: p(:,:), q(:,:)
        ! actual momentum and position at timestep n
    double precision, allocatable :: pstore(:,:,:), qstore(:,:,:)
        ! buffer matrix of p and q values
    integer*8, dimension(2) :: phasecoord=[1,2], phasecoordr=[3,4]
    integer*8 :: fm=7, fdetails=8
        ! handy reference for logical unit numbers
    character :: fpath*64, paramfname*64, overwrite*1
        ! directory name for the data files from the integration
```

```
        ! parameters filesname
        ! overwrite permission if a run has been completed in
        ! fpath location
    logical :: breakrun=.false., fexist=.false., done,&
               &started, doneforward, fwriteiter
        ! breakrun: set to true if the integration ends before
        ! N iterations
        ! fexist: used in inquiries into existence of files
        ! done: whether or not a run has been completed in
        ! directory given by fpath already
        ! started: whether or not a run has started but not
        ! finished
        ! fwriteiter: when resuming reverse run, if it is
        ! writing for the first time
    integer*8 :: ios    ! iostat result
    integer*8 :: pos, posr  ! file positions for resuming

    namelist /parameters/ eccentricity, meanmotionratio, numplan,&
                          &dimensions, dt, N, storefrequency,&
                          &dumpfrequency, testerror, method,&
                          &fpath, G, m;

    namelist /initconds/ vx, vy, vz, qx, qy, qz;
    namelist /laststate/ li, lp, lq;

    !namelist /highordercoeffs/ w

    !write(*,'(A)',advance='no') 'Enter name of parameters file: '
    !read*,paramfname
    paramfname = 'params.dat';

    inquire(file=trim(paramfname),exist=fexist);
    if (.not.fexist) then
        stop 'Parameters file does not exist.'
    endif

    open(100,file=trim(paramfname));

! loop around the main program until the end of params.dat is reached
prog: do
    read(100,nml=parameters, iostat=ios);
    open(101,file='ics.dat');
    read(101,nml=initconds);
    close(101);

```

```
    allocate(p(numplan,dimensions),q(numplan,dimensions));
    allocate(pstore(numplan,dimensions,dumpfrequency),&
             &qstore(numplan,dimensions,dumpfrequency));

    inquire(file=trim(fpath)//'/fin', exist=done);
    if (done) then
        overwrite = 'n';
        if (overwrite/='y') then
            deallocate(p,q);
            deallocate(pstore,qstore);
                if (ios == -1) then
                    print*,'reached end of params.dat';
                    exit
                endif
            cycle prog
        endif
    endif

    inquire(file=trim(fpath)//'/p.dat',exist=started);
    if (.not.started) then
        ! do everything normally.

        call generatescales(eccentricity, meanmotionratio, scales);

        if (method == 1) then
            order = 4;
            coeffs = 4;
        elseif (method == 2) then
            order = 8;
            coeffs = 16;
        else
            order = 2;
            coeffs = 2;
        endif
        allocate(a(coeffs),b(coeffs));

        call setcoeffs(method,coeffs,order,a,b)!,w)

        ! set initial conditions into correct arrays
        p(1:numplan,1:dimensions) = 0;
        do i = 1,numplan
           p(i,1) = vx(i)*m(i);
           p(i,2) = vy(i)*m(i);
           p(i,3) = vz(i)*m(i);
            if (i == 2) then
```

```
                p(i,:) = p(i,:)*scales(2);
            endif
        enddo

        q(1:numplan,1:dimensions) = 0;
        do i = 1,numplan
            q(i,1) = qx(i);
            q(i,2) = qy(i);
            q(i,3) = qz(i);
            if (i == 2) then
                q(i,:) = q(i,:)*scales(1);
            endif
        enddo

        N = ceiling(real(N)/real(storefrequency))*storefrequency;

        call opendatafiles(fpath, phasecoord, phasecoordr, fm, fdetails);
        open(102,file=trim(fpath)//'/laststate.dat');
        open(104,file=trim(fpath)//'/resumelineno.dat');
        close(104);

        li = 0;
        lp = p;
        lq = q;
        write(102,nml=laststate);
        rewind(102);

        ! write integration details and masses for the
        ! run to appropriate files
        do i = 1,size(m)
            write(fm,300),m(i);
        enddo
        write(fdetails,301), numplan, dimensions, dt,&
              &N, storefrequency,G, testerror,&
              &method, dumpfrequency, fpath;
        close(fm);
        close(fdetails);
300 format(e24.17);
301 format(i8,/,i1,/,e24.17,/,i16,/,i16,/,e24.17,/,i1,/,i1,/,i16,/,a,/);

        ! here we begin to integrate, storing to buffer and
        ! writing buffer to disk as needed
        breakrun = .false.;
        i = 0;
        do i=0,N-1
```

```
        if (mod(i,storefrequency)== 0) then
            call store(q,p,qstore,pstore,&
                &mod(i,storefrequency*dumpfrequency)/storefrequency+1);
            call asteccentricity(p,q,m,G,e_ast);
            !print*,e_ast
            if (e_ast > 0.8 .and. .not.breakrun) then
                breakrun = .true.;
                print*,'Eccentricity of asteroid ',e_ast,&
                    &' > 0.8 at timestep',i,'\n';
            endif
!             fprintf('%i: stored to buffer\n',i);
        endif
        if (mod(i,storefrequency*dumpfrequency)==&
            &dumpfrequency*storefrequency-1) then
            call dump(qstore,phasecoord(1));
            call dump(pstore,phasecoord(2));

            li = i;
            lp = p;
            lq = q;
            write(102,nml=laststate);
            rewind(102);

            !print*,'Dumped buffer at timestep',i
            if (breakrun) then
                N = i;
                exit;
            endif
        elseif (i == N-1) then
            call dump(qstore,phasecoord(1));
            call dump(pstore,phasecoord(2));

            li = i;
            lp = p;
            lq = q;
            write(102,nml=laststate);
            rewind(102);

            !print*,'Dumped buffer at timestep',i
        endif

        call integrate(p,q,m,G,dt,a,b,coeffs);
    enddo

    open(111,file=trim(fpath)//'/finforward');
```

```
        close(111);

        ! then we integrate backwards if necessary
        if (testerror == 1) then
            p = -p;
            do i = 0,N+storefrequency
                if (mod(i,storefrequency)==0 .and. i/=0) then
                    call store(q,p,qstore,pstore,&
                          &mod(i-storefrequency,&
                          &storefrequency*dumpfrequency)/storefrequency+1);
                endif
                if (mod(i,storefrequency*dumpfrequency)==0 .and. i/=0) then
                    call dump(qstore,phasecoordr(1));
                    call dump(pstore,phasecoordr(2));

                    li = i;
                    lp = p;
                    lq = q;
                    write(102,nml=laststate);
                    rewind(102);

                elseif (i == N) then
                    call dump(qstore,phasecoordr(1));
                    call dump(pstore,phasecoordr(2));

                    li = i;
                    lp = p;
                    lq = q;
                    write(102,nml=laststate);
                    rewind(102);

                endif

                call integrate(p,q,m,G,dt,a,b,coeffs);
            enddo
        endif

        ! finally, loop back and get some new ICs
        close(phasecoord(1));
        close(phasecoord(2));
        close(phasecoordr(1));
        close(phasecoordr(2));

        open(111, file=trim(fpath)//'fin')
        close(111)
```

```

        if (ios == -1) then
            print*,'reached end of params.dat';
            exit
        else
            deallocate(p,q);
            deallocate(pstore,qstore);
            deallocate(a,b);
            cycle prog
        endif

    else
        ! Let's find out where the last complete record is, then
        ! get to a position where we can just integrate normally.
        ! If testerror == 1, and p.dat/q.dat are full, go through
        ! pr.dat and qr.dat to find the last completed record and
        ! continue integrating from there.

        ! WARNING
        ! This section still does not work correctly.

        if (method == 1) then
            order = 4;
            coeffs = 4;
        elseif (method == 2) then
            order = 8;
            coeffs = 16;
        else
            order = 2;
            coeffs = 2;
        endif
        allocate(a(coeffs),b(coeffs));

        call setcoeffs(method,coeffs,order,a,b)!,w)

        N = floor(real(N)/real(storefrequency))*storefrequency;

        inquire(file=trim(fpath)//'/finforward', exist=doneforward);
        if (.not.doneforward) then

            call opendatafilesresume(fpath, phasecoord, phasecoordr);
            open(102,file=trim(fpath)//'/laststate.dat');
            open(104,file=trim(fpath)//'/resumelineno.dat',position='append');
            write(104,105)pos;
105 format ('f: ',i10);
```

```
            close(104);

            read(102,nml=laststate,err=103);

            pos = (li)*numplan*dimensions;
            k1 = mod(li,dumpfrequency);

            !print*,pos,li,lp,lq
            p = lp;
            q = lq;

            do i = 1,storefrequency
                call integrate(p,q,m,G,dt,a,b,coeffs);
            enddo

            breakrun=.false.;
            do i=(li)*storefrequency,N-1
                if (mod(i,storefrequency)== 0) then
                    pstore(:,:,mod(i,storefrequency*&
                        &dumpfrequency)/storefrequency+1-k1)=p;
                    qstore(:,:,mod(i,storefrequency*&
                        &dumpfrequency)/storefrequency+1-k1)=q;
                    call asteccentricity(p,q,m,G,e_ast);
                    if (e_ast > 0.8) then
                        breakrun = .true.;
                        print*,'Eccentricity of asteroid',&
                            e_ast,'> 0.8 at timestep',i;
                    endif
                    !fprintf('%i: stored to buffer\n',i);
                endif
                if (mod(i,storefrequency*dumpfrequency)==&
                    &dumpfrequency*storefrequency-1) then  ! write buffer to file
                    call dump(qstore(:,:,1:dumpfrequency-k1),phasecoord(1));
                    call dump(pstore(:,:,1:dumpfrequency-k1),phasecoord(2));

                    li = i;
                    lp = p;
                    lq = q;
                    write(102,nml=laststate);
                    rewind(102);
                    if (breakrun) then
                        N = i;
                        exit;
                    endif
                    k1=0;
```

```
            elseif (i == N-1) then
                call dump(qstore(:,:,1:dumpfrequency-k1),phasecoord(1));
                call dump(pstore(:,:,1:dumpfrequency-k1),phasecoord(2));

                li = i;
                lp = p;
                lq = q;
                write(102,nml=laststate);
                rewind(102);
            endif

            call integrate(p,q,m,G,dt,a,b,coeffs);
        enddo

        open(111,file=trim(fpath)//'/finforward');
        close(111);

        if (testerror == 1) then
            p = -p;
            print*,'testerror = true. Reversing flow.';
            do i = 0,N+storefrequency
                if (mod(i,storefrequency)==0&
                    &.and. i.ne.0) then     ! store current data to buffer
                    pstore(:,:,mod(i-storefrequency,&
                        &storefrequency*dumpfrequency)/storefrequency+1)=p;
                    qstore(:,:,mod(i-storefrequency,&
                        &storefrequency*dumpfrequency)/storefrequency+1)=q;
                    !fprintf('%i: stored to buffer\n',N+storefrequency-i);
                endif
                if (mod(i,storefrequency*dumpfrequency)==0 .and. i.ne.0) then
                    call dump(qstore(:,:,1:dumpfrequency-k1),phasecoord(1));
                    call dump(pstore(:,:,1:dumpfrequency-k1),phasecoord(2));

                    li = i;
                    lp = p;
                    lq = q;
                    write(102,nml=laststate);
                    rewind(102);
                    k1=0;
                    ! fprintf('%i:- cleared buffer\n',i);
                elseif (i == N) then
                    call dump(qstore(:,:,1:dumpfrequency-k1),phasecoord(1));
                    call dump(pstore(:,:,1:dumpfrequency-k1),phasecoord(2));

                    li = i;
```

```
                        lp = p;
                        lq = q;
                        write(102,nml=laststate);
                        rewind(102);

                        exit;
                        ! fprintf('%i: cleared buffer\n',i);
                    endif

                    call integrate(p,q,m,G,dt,a,b,coeffs);
                enddo
            endif

            ! finally, loop back and get some new ICs from parameters and initcon
            close(phasecoord(1));
            close(phasecoord(2));
            close(phasecoordr(1));
            close(phasecoordr(2));

            open(111, file=trim(fpath)//'/fin')
            close(111)

            if (ios == -1) then
                print*,'reached end of params.dat';
                exit
            else
                deallocate(p,q);
                deallocate(pstore,qstore);
                deallocate(a,b);
                cycle prog
            endif
        elseif (testerror == 1) then

            call opendatafilesresume(fpath, phasecoord, phasecoordr);
            open(102,file=trim(fpath)//'/laststate.dat');
            open(104,file=trim(fpath)//'/resumelineno.dat',position='append');
            write(104,106) posr;
106 format ('r: ',i10)
            close(104);

            read(102,nml=laststate,err=103);

            posr = li*numplan*dimensions;

            print*,posr,li,lp,lq
```

```
p = lp;
q = lq;

if (li*storefrequency < N) then
    print*,'Resuming reverse run';
    breakrun=.false.;
    fwriteiter=.true.;
    k1=mod(li,dumpfrequency);

    if (li == 0) then
        do i = 1,storefrequency
            call integrate(p,q,m,G,dt,a,b,coeffs);
        enddo
        p=-p;
    else
        ! this block progresses us to the next timestep
        ! that would be recorded and avoids the calculation
        ! going out of phase with an uninterrupted simulation
        ! from the original ICs.
        do i = 1,storefrequency
            call integrate(p,q,m,G,dt,a,b,coeffs);
        enddo
    endif
    do i = li*storefrequency,N+storefrequency
        if (mod(i,storefrequency)==0 .and. i.ne.0) then
            pstore(:,:,mod(i-storefrequency,&
                &storefrequency*dumpfrequency)/storefrequency+1)=p;
            qstore(:,:,mod(i-storefrequency,&
                &storefrequency*dumpfrequency)/storefrequency+1)=q;
            !fprintf('%i: stored to buffer\n',N+storefrequency-i);
        endif
        if (fwriteiter .and. mod(i,storefrequency*dumpfrequency)==0)
            call dump(qstore(:,:,1:dumpfrequency-k1),phasecoord(1));
            call dump(pstore(:,:,1:dumpfrequency-k1),phasecoord(2));

            li = i;
            lp = p;
            lq = q;
            write(102,nml=laststate);
            rewind(102);
            k1=0;
            fwriteiter = .false.;
!             fprintf('%i:- cleared buffer\n',i);
        elseif (mod(i,storefrequency*dumpfrequency)==0 .and. i.ne.0)
            call dump(qstore(:,:,1:dumpfrequency-k1),phasecoord(1));
```

```
                        call dump(pstore(:,:,1:dumpfrequency-k1),phasecoord(2));

                        li = i;
                        lp = p;
                        lq = q;
                        write(102,nml=laststate);
                        rewind(102);
                        k1=0;
            !             fprintf('%i:- cleared buffer\n',i);
                    elseif (i == N) then
                        call dump(qstore(:,:,1:dumpfrequency-k1),phasecoord(1));
                        call dump(pstore(:,:,1:dumpfrequency-k1),phasecoord(2));

                        li = i;
                        lp = p;
                        lq = q;
                        write(102,nml=laststate);
                        rewind(102);
                        exit;
            !             fprintf('%i: cleared buffer\n',i);
                    endif

                    call integrate(p,q,m,G,dt,a,b,coeffs);
                enddo
            else
                print*,'No need to resume reverse run';
            endif
        endif
    endif
103 print*,'Error reading last state from disk. Manual recovery required.'
    close(102);
enddo prog
    close(100);
end program asteroid

module useful
contains
function ones(n,m)
! outputs a 2D array of ones, with dimensions n x m.
    integer*8, dimension(n,m) :: ones

    ones(:,:) = 1;
end function ones

function zeros(n,m)
```

```
! outputs a 2D array of zeros, with dimensions n x m.
    integer*8, dimension(n,m) :: zeros

    zeros(:,:) = 0;
end function zeros

function cross(a,b)
! returns the cross product of length-3 arrays a and b
    double precision, dimension(3) :: a, b, cross

    cross(1) = a(2)*b(3)-a(3)*b(2);
    cross(2) = a(3)*b(1)-a(1)*b(3);
    cross(3) = a(1)*b(2)-a(2)*b(1);
end function cross
end module useful

subroutine setcoeffs(method,coeffs,order,a,b)!,w)
use globals
    integer*8 :: method, order, coeffs
    !double precision :: w(0:7)
    double precision :: a(coeffs), b(coeffs)

    !namelist /highordercoeffs/ w

    if(method == 1) then
        ! fourth order coefficients
        a = [1d0/(2d0*(2d0-2d0**(1d0/3d0))),&
            &(1d0-2d0**(1d0/3d0))/(2d0*(2d0-2d0**(1d0/3d0))),&
            &(1d0-2d0**(1d0/3d0))/(2d0*(2d0-2d0**(1d0/3d0))),&
            &1d0/(2d0*(2d0-2d0**(1d0/3d0)))];
        b = [1d0/(2d0-2d0**(1d0/3d0)),&
            &-(2d0**(1d0/3d0))/(2d0-2d0**(1d0/3d0)),&
            &1d0/(2d0-2d0**(1d0/3d0)),0d0];
        print*,'using 4-th order forest & ruth';
    elseif(method == 2) then
        ! eighth order coefficients
        !open(101,file='coeffs.dat');
        !read(101,nml=highordercoeffs);
        !read(101,nml=highordercoeffs);
        !read(101,nml=highordercoeffs);
        !read(101,nml=highordercoeffs);
        !read(101,nml=highordercoeffs);
        !close(101);

        !w(0) = 1-2*sum(w);
```

```

        !a(1)  = w(7)/2d0;
        !a(16) = w(7)/2d0;
        !b(1)  = w(7);
        !b(15) = w(7);
        !b(16) = 0d0;

        !do i = 2,order-1
        !    a(i)           = (w(order+1-i)+w(order+2-i))/2d0;
        !    a(2*order-i)   = (w(order+1-i)+w(order+2-i))/2d0;
        !    b(i)           = w(order-i);
        !    b(2*order-i-1) = w(order-i);
        !enddo
        !print*,'using 8-th order yoshida';
        stop 'Sorry, 8-th order routine not implemented';
    else
        ! leapfrog coefficients
        a = [0.5d0, 0.5d0];
        b = [1d0, 0d0];
        print*,'using leapfrog';
    endif
end subroutine setcoeffs

subroutine opendatafiles(fpath, phasecoord, phasecoordr, fm, fdetails)
    ! opens data files in the correct directories
    ! for reading/writing/appending
    logical :: direxist
    integer*8 :: phasecoord(2), phasecoordr(2), fm, fdetails!, fparams
    character :: fpath*64

    ! test if the specified path to write to exists: if not, create it.
    inquire(file=trim(fpath), exist=direxist);
    if (.not.direxist) then
        print*,'directory ',trim(fpath),' does not exist. creating it.'
        call system('mkdir ' // trim(fpath));
    endif

    open(phasecoord(1), file=trim(fpath)//'/q.dat');
    open(phasecoord(2), file=trim(fpath)//'/p.dat');

    open(phasecoordr(1), file=trim(fpath)//'/qr.dat');
    open(phasecoordr(2), file=trim(fpath)//'/pr.dat');

    open(fm, file=trim(fpath)//'/m.dat');

```

```
    open(fdetails, file=trim(fpath)//'/integrationdetails.dat');

    !open(fparams, file=trim(fpath)//'/parameters.dat');

    return
end subroutine opendatafiles

subroutine opendatafilesresume(fpath, phasecoord, phasecoordr)
    ! opens data files in the correct directories
    ! for reading/writing/appending
    logical :: direxist
    integer*8 :: phasecoord(2), phasecoordr(2)
    character :: fpath*64

    ! test if the specified path to write to exists: if not, create it.
    inquire(file=trim(fpath), exist=direxist);
    if (.not.direxist) then
        print*,'directory ',trim(fpath),' does not exist. creating it.'
        call system('mkdir ' // trim(fpath));
    endif

    open(phasecoord(1), file=trim(fpath)//'/q.dat', position='append');
    open(phasecoord(2), file=trim(fpath)//'/p.dat', position='append');

    open(phasecoordr(1), file=trim(fpath)//'/qr.dat', position='append');
    open(phasecoordr(2), file=trim(fpath)//'/pr.dat', position='append');

    return
end subroutine opendatafilesresume

subroutine dump(variable,fileid)
    ! cycle i through length of var
    ! for var(i), write using logical unit fileid
    ! then clear var
    use globals
    integer*8 :: fileid
    double precision :: variable(numplan,dimensions,dumpfrequency)

    write(fileid,200) variable;
200 format (e24.17)
    variable = 0;

    return
end subroutine dump
```

```
subroutine store(q,p,qstore,pstore,i)
    ! place current value of q and p into a slot in the buffer
    use globals
    integer*8 :: i
    double precision, dimension(numplan,dimensions) :: q, p
    double precision, dimension(numplan,dimensions,dumpfrequency)&
                                  & :: qstore, pstore
    do j = 1,size(p,1)
        do k = 1,size(p,2)
            pstore(j,k,i)=p(j,k);
            qstore(j,k,i)=q(j,k);
        enddo
    enddo

    return
end subroutine store

module vel
contains
function velocity(p,m)
    use globals
    double precision, dimension(numplan,dimensions) :: p, velocity
    double precision, dimension(numplan) :: m
    do i = 1,numplan
        velocity(i,:) = p(i,:)/m(i);
    enddo

    return
end function velocity

function force(q,G,m)
use globals
    double precision, dimension(numplan,dimensions) :: q, force
    double precision, dimension(numplan) :: m
    double precision, dimension(numplan,dimensions,numplan) :: dU, diff
    double precision, dimension(numplan,numplan) :: denom
    double precision :: pow = 1.5d0, G

    dU(1:numplan,1:dimensions,1:numplan) = 0;
    diff(1:numplan,1:dimensions,1:numplan) = 0;
    denom(1:numplan,1:numplan) = 0;
    do i = 1,numplan
        do j = 1,numplan
            if (i == j) then
                diff(i,:,j) = 0;
```

```
                denom(i,j) = 0;
                dU(i,:,j) = 0;
            else
                diff(i,:,j) = -(q(i,:)-q(j,:));
                do k = 1,dimensions
                    denom(i,j) = denom(i,j)+(diff(i,k,j)**2);
                enddo
                dU(i,:,j) = -G*m(i)*m(j)*diff(i,:,j)/(denom(i,j)**pow);
            endif
        enddo
    enddo
    force = sum(dU,3);

    return
end function force
end module vel

subroutine integrate(p,q,m,G,dt,a,b,coeffs)
    ! perform the integration calculation from step n -> n+1.
    ! the length and contents of a and b determine which
    ! symplectic integrator is used.
    use globals
    use vel
    integer*8 :: coeffs
    double precision, dimension(numplan,dimensions) :: q, p
    double precision, dimension(numplan) :: m
    double precision, dimension(coeffs) :: a, b
    double precision :: G, dt
    do i = 1,coeffs
        if (a(i) /= 0) then
            q = q + a(i)*dt*velocity(p,m);
        endif
        if (b(i) /= 0) then
            p = p - b(i)*dt*force(q,G,m);
        endif
    enddo

    return
end subroutine integrate

subroutine asteccentricity(p,q,m,G,eccentricity)
    use globals
    use useful
    double precision, dimension(numplan) :: m
    double precision, dimension(dimensions) :: r, v, h
```

```
    double precision, dimension(numplan,dimensions) :: q, p
    double precision :: G, nr, nv, nh, mu, eccentricity

    r        = q(2,:)-q(1,:);                ! relative position
    nr       = sqrt(sum(r**2));              ! magnitude of r
    v        = p(2,:)/m(2)-p(1,:)/m(1);      ! relative velocity
    nv       = sqrt(sum(v**2));              ! magnitude of v
    h        = cross(r,v);                   ! normal vector of orbit
                                             ! i.e. angular momentum per unit mass

    nh       = sqrt(sum(h**2));              ! magnitude of normal

    mu = G*(m(1)+m(2));                      ! reduced mass
    eccentricity=sqrt(1-nh**2*(2*mu-nr*nv**2)/(nr*mu**2));

end subroutine asteccentricity

subroutine generatescales(eccentricity, meanmotratio, scales)
! given a desired average eccentricity and average mean motion ratio with
! jupiter (given that the asteroid starts within jupiter's orbit), this
! determines an appropriate pair of scale factors for the asteroid's
! initial conditions (for simplicity having the asteroid start directly on
! the line between jupiter and the sun).
    use useful

    double precision :: G=2.95912208286e-4, m=1.00000597682, mu, rj, vj, hj,&
                        &meanmotjupi, meanmotasti, a, ajup, eccentricity,&
                        &meanmotratio, h, hsquared
!     double precision :: pplus, splus
    double precision :: pminus, sminus
    double precision, dimension(2) :: scales
    double precision, dimension(3) :: vjupi, qjupi, hjupi

    !print*,G,m;

    mu = G*(m+1d-15);       ! 1e-15 is the mass of the asteroid

    ! Jupiter's initial state vectors
    vjupi = [0.00565429, -0.00412490, -0.00190589];
    qjupi = [-3.5023653,-3.8169847,-1.5507963];
    hjupi = cross(qjupi,vjupi)

    rj = sqrt(sum(qjupi**2));
    vj = sqrt(sum(vjupi**2));
    hj = sqrt(sum(hjupi**2));
```

```

    ajup = 1/(2/rj - vj**2/(G*(m+0.000954786104043)));
    meanmotjupi = sqrt(G*(m+0.000954786104043)/ajup**3);
    meanmotasti = meanmotratio*meanmotjupi;
    a = (mu/meanmotasti**2)**(1/3d0);

    hsquared = mu*a*(1-eccentricity**2)/hj**2;
    h = sqrt(hsquared);

    ! pplus = a/rj + sqrt((mu*a)**2 - mu*a*hsquared*hj**2)/(mu*rj);
    ! splus = (h/pplus)*(hj/rj);
    ! scales(1) = pplus;
    ! scales(2) = splus;

    pminus = a/rj - sqrt((mu*a)**2 - mu*a*hsquared*hj**2)/(mu*rj);
    sminus = (h/pminus)*(hj/rj);
    scales(1) = pminus;
    scales(2) = sminus;
end subroutine generatescales
```

File: *ics.dat*

```
&INITCONDS VX = 0., -0.761576392933587, 0.00565429, 0.00168318,
VY = 0., 0.588733316817015, -0.0041249, 0.00483525, VZ = 0.,
0.270914155030523, -0.00190589, 0.00192462, QX = 0., 3.5023653,
-3.5023653, 9.0755314, QY = 0., 3.8169847, -3.8169847, -3.0458353,
QZ = 0., 1.5507963, -1.5507963, -1.6483708,
/
```

File: *params.dat* This is just one example parameters file:

```
&PARAMETERS ECCENTRICITY = 0.15, MEANMOTIONRATIO = 2.846542,
 NUMPLAN = 4, DIMENSIONS = 3, DT = 43.31572, N = 8500000,
 STOREFREQUENCY = 1000, DUMPFREQUENCY = 10000, TESTERROR = 0,
 METHOD = 1, FPATH = 'Hherror05', G = 0.000295912208286,
 M = 1.00000597682, 1.E-15, 0.000954786104043, 0.000285583733151
 /
 &PARAMETERS ECCENTRICITY = 0.15, MEANMOTIONRATIO = 2.846542,
 NUMPLAN = 4, DIMENSIONS = 3,
 DT = .4331572, N = 850000000, STOREFREQUENCY = 100000,
 DUMPFREQUENCY = 10000, TESTERROR = 0,
 METHOD = 1, FPATH = 'Hherror06', G = 0.000295912208286,
 M = 1.00000597682, 1.E-15, 0.000954786104043, 0.000285583733151
 /
&PARAMETERS ECCENTRICITY = 0.15, MEANMOTIONRATIO = 2.846542,
NUMPLAN = 4, DIMENSIONS = 3, DT = 43.31572, N = 8500000,
STOREFREQUENCY = 1000, DUMPFREQUENCY = 10000, TESTERROR = 0,
 METHOD = 0, FPATH = 'Hherror07', G = 0.000295912208286,
 M = 1.00000597682, 1.E-15, 0.000954786104043, 0.000285583733151
 /
 &PARAMETERS ECCENTRICITY = 0.15, MEANMOTIONRATIO = 2.846542,
 NUMPLAN = 4, DIMENSIONS = 3, DT = .4331572, N = 850000000,
 STOREFREQUENCY = 100000, DUMPFREQUENCY = 10000, TESTERROR = 0,
 METHOD = 0, FPATH = 'Hherror08', G = 0.000295912208286,
 M = 1.00000597682, 1.E-15, 0.000954786104043, 0.000285583733151
 /
```

References

[1] P J Channell and C Scovel. Symplectic integration of hamiltonian systems. *Nonlinearity*, 3:231–259, 1990.

[2] J D Hadjimetriou. Asteroid motion near the 3:1 resonance. *Celestial Mechanics and Dynamical Astronomy*, 56:563–599, 1993.

[3] A Morbidelli and J Henrard. Secular resonances in the asteroid belt: theoretical perturbation approach and the problem of their location. *Celestial Mechanics and Dynamical Astronomy*, 51:131–677, 1991.

[4] J Henrard. Motion near the 3/1 resonance of the planar elliptic restricted three body problem. *Celestial Mechanics and Dynamical Astronomy*, 47:99–121, 1990.

[5] H Varvoglis K Tsiganis and J D Hadjimetriou. Stable chaos versus kirkwood gaps in the asteroid belt: A comparative study of mean motion resonances. *Icarus*, 159:284–299, 2002.

[6] H Varvoglis K Tsiganis and J D Hadjimetriou. Stable chaos in high-order jovian resonances. *Icarus*, 155:454–474, 2002.

[7] A Lemaitre and J Henrard. The 3/2 resonance. *Celestial Mechanics*, 43:91–98, 1988.

[8] F Namouni. Secular interactions of coorbiting objects. *Icarus*, 137:293–314, 1999.

[9] J Wisdom. The kirkwood gaps: A mapping technique for asteroidal motion near the 3/1 commensurability. *Astron. J*, 87:577–593, 1982.

[10] J Wisdom. Chaotic behaviour and the origin of the 3/1 kirkwood gap. *Icarus*, 56:51–74, 1983.

[11] J Wisdom. A perturbative treatment of motion near the 3/1 commensurability. *Icarus*, 63:272–289, 1983.

[12] C.D. Murray and S.F.Dermott. *Solar System Dynamics*, chapter 8, page 321. Cambirdge University Press, first edition, 1999.

[13] N Murray and M Holman. The role of chaotic resonances in the solar system. *Nature*, 410:773–779, April 2001.

[14] E Forest and R D Ruth. Fourth-order symplectic integration. *Physica D*, August 1989.

[15] J Candy and W Rozmus. A symplectic integration algorithm for separable hamiltonian functions. *Journal of Computational Physics*,

92:230–256, 1991.

[16] H Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150:262–268, November 1990.

[17] R I McLachlan and P Atela. The accuracy of symplectic integrators. *Nonlinearity*, 5:541–562, 1992.

[18] C Lubich E Hairer and G Wanner. *Geometrical Numerical Integration: Structure-preserving Algorithms for Ordinary Differential Equations*, chapter 1, page 11. Springer, 2002.

[19] C.D. Murray and S.F.Dermott. *Solar System Dynamics*, chapter 9, page 456. Cambirdge University Press, first edition, 1999.

[20] F Benitez and T Gallardo. The relativistic factor in the orbital dynamics of point masses. *Celestial Mechanics & Dynamical Astronomy*, 101:289–307, Jul 2008.

[21] W Martin J D Anderson, P B Esposito and D O Muhlemsn. Experimental test of general relativity using time-delay data from mariner 6 and mariner 7. *Astrophys. J*, 200:221–233, 1975.

[22] P Saha. Simulating the 3:1 kirkwood gap. *Icarus*, 100:434–439, 1992.

[23] M Moons. Review of the dynamics in the kirkwood gaps. *Celestial Mechanics and Dynamical Astronomy*, 65:175–207, 1997.

[24] C.D. Murray and S.F.Dermott. *Solar System Dynamics*, chapter 9, page 460. Cambirdge University Press, first edition, 1999.

[25] C.D. Murray and S.F.Dermott. *Solar System Dynamics*, chapter 2, page 53. Cambirdge University Press, first edition, 1999.