Long-term Stability of Planetary Systems in Clusters

George Corney

Abstract. An n-body simulation was used to investigate the long-term stability of the Solar System in a range of environments with different star densities. The encounter distances required to induce instability is estimated to be at least 150 AU for globular clusters, 800 AU for open clusters and 50 AU for the Solar Neighbourhood. These figures indicate a mean time between collisions of at most 60 Myr for globular clusters, 800 Myr for open clusters and 1 Tyr for the solar neighbour. Consequently, we expect stars within the cores of clusters to experience many critical encounters during their lifetime and hence suggest that Solar System-like planetary systems cannot survive unperturbed over billion-year timescales in these environments.

1. Introduction and Context

As the stars in clusters are of similar age and composition, they offer an ideal environment for the study of exoplanets. Despite this, we have very little observational evidence of planets in clusters and the theoretical understanding of planet formations in high density conditions is poor. Although very few planets have been discovered in globular clusters, there is evidence to suggest they could even be as common in open clusters as they are in isolated environments (Brucalassi et al. 2014).

The exoplanets that have been discovered in clusters have unusual characteristics; PSR B1620-26 b, one of the only planets discovered in a globular cluster, orbits two stars, a pulsar and a neutron star, with a semi-major axis of 23 AU. It is also believed to be one of the oldest exoplanets discovered at 12.7 billion years old (Sigurdsson et al. 2003). The exoplanets discovered in open clusters tend to be hot jupiters, orbiting stars as or more massive than the Sun in eccentric but very tight orbits, less than about 0.05 AU (Brucalassi et al. 2014).

To distinguish these discoveries from products of observational biases requires a theoretical understanding complete enough to generate expected distributions of different types of planetary systems. This will offer constraints on what types of planetary systems we expect to see and not to see in clusters. The most interesting results will be the observations that conflict with these predictions - as has often been the case in exoplanet research.

2. Method

The average time between destabilising interactions can be approximated with a mean free path approach. A star with a planetary system travelling at velocity v, traces out cylinder of radius $R_c(v, m_2)$, where R_c is defined as the furthest distance at which a gravitational interaction with a star of mass m_2 will destabilise the planetary system. If the local stellar environment is static with number density n, then expected number of critical interactions $\bar{N}$ within a time t is:

$$\bar{N} = \pi v n t R_c(v, m_2)^2 \tag{1}$$

And so the average time between critical interactions $\bar{T}$, assuming typical stellar mass $\bar{m}$ and relative velocity $\bar{v}$ in the environment, is:

$$\bar{T} \approx \frac{1}{\pi \bar{v} n R_c(\bar{v}, \bar{m})^2} \tag{2}$$

To develop a rough picture of long-term stability, the form of $R_c(v, m_2)$ for the (present day) Solar System was estimated and used to find the average time between critical interaction in different stellar environments.

2.1. Estimating R_c for the Solar System

To imitate an encounter with a nearby star in an n-body simulation, a neighbouring is star placed at random position $10,000$ AU from the Sun ‡ and given a trajectory to pass within a specified distance, d, of the Solar System (neglecting gravity) (see appendix Appendix A for details). After the closest approach is reached, the system is evolved for 1 Myr and on completion the magnitude of perturbation is measured.

By repeatedly running the simulation whilst varying the approach distance d and initial velocity v of the neighbouring star, a map of stability can be made, from which, the critical interaction radius R_c of the Solar System can be found for different stellar environments. Each combination of d and v is run 10 times, each time randomising the starting coordinates of the neighbouring star in order to smooth out fluctuations.

2.2. Measuring Perturbation

A system is classified to have lost stability if either a planet is ejected (its eccentricity exceeds 1), or the semi-major axis of any one of the planets changes by a factor 2 since the simulation began. The semi-major axis is computed from orbital energy conservation as follows:

$$a = \left(\frac{2}{r} - \frac{v^2}{GM(r)} \right)^{-1} \tag{3}$$

‡ $10,000$ AU is chosen such that at that distance the interaction potential is much less than relative kinetic energy between the stars and, at a few km/s, the time required to collide is suitably small.

the eccentricity can be found from the semi-major axis with:

$$e = \sqrt{1 - \frac{|\vec{r} \times \vec{v}|^2}{aGM(r)}} \tag{4}$$

where $\vec{r}$ is the planet-sun displacement vector, $\vec{v}$ is the velocity of the planet, $M(r)$ is the total mass (including planets) within radius r and G is the gravitational constant. While this method accounts for the additional mass within the planet-sun radius, it neglects the distribution of that mass, as well as potential perturbations outside the planet-sun radius.

When a system isn't classified as unstable the average relative change in semi-major axis of each of the planets is used as a measure of the *magnitude* of perturbation. The relative change in semi-major axis is $|a_0 - a_{end}|/a_0$, where a_0 is the initial semi-major axis and a_{end} is the semi-major axis computed at the end of the simulation.

2.3. Integrating the N-body Problem Over Long Timescales

To build a stability map the long-term n-body simulation must be run many thousands of times and so it is critical that each $\sim 1\,\mathrm{Myr}$ execution takes as little wall-time as possible but still preserves enough accuracy to maintain stability (when there are no perturbations) and ensure realistic orbits. The integrator then, must be developed to balance these constraints.

The acceleration on each body i due gravitational attraction to every other body j is:

$$\vec{a_i} = G \sum_{j,i \neq j} \frac{m_j \hat{x_{ij}}}{|\vec{x_{ij}}|^2} \qquad \text{where } \vec{x_{ij}} = \vec{x}_j - \vec{x}_i \tag{5}$$

and m_j is the mass of body j. As gravity is the only consideration, this gives the Hamiltonian to be integrated:

$$H = \sum_i \frac{1}{2} m v_i^2 + G \sum_i \sum_{j>i} \frac{m_j m_i}{|\vec{x_j} - \vec{x_i}|} \tag{6}$$

A 2nd order 'Leapfrog' integrator was chosen as the integrator and used to generate the stability maps (see figures 3 and 4 in the Results section), to understand why this method was selected over higher order methods, a brief review of the theoretical background and behaviour of integrators commonly used in n-body problems is given in the following section.

2.4. Review of Integrator Methods

The Taylor expansion provides a method to approximate a function at a nearby point in time using the time derivatives of the function and so serves as a foundation for the construction of many integrators. The Taylor expansion of arbitrary position function x is

$$x_1 = x_0 + v_0 \Delta t + \frac{a_0}{2!} \Delta t^2 + ... + \frac{x_0^{(n)}}{n!} \Delta t^n \tag{7}$$

where x_0 is position at time t and x_1 is the approximated position after time Δt.

The **Euler method** is formed by truncating the expansion after the first time derivative, giving explicit equations for position and velocity:

$$x_1 = x_0 + v_0 \Delta t \tag{8}$$

$$v_1 = v_0 + a_0 \Delta t \tag{9}$$

In practice it turns out that by advancing x with v_1 instead of v_0 in (8) (to become $x_1 = x_0 + v_1 \Delta t$) (See implementation Appendix B.1), yields considerably better energy conservation (Hairer et al. 2006, p.12). This can be explained from the proof that the method solves a reduced version of the Hamiltonian exactly (Gunduz 2010, p.17), making it a *symplectic* method. It is often called the **Symplectic Euler** method for this reason. The Euler method on the other hand is not symplectic - it solves Hamiltonians approximately and consequently exhibits comparatively poor conservation of total energy. The global error of Euler-based methods grows proportionally with Δt, making it a 1st order method.

Higher order integrators can be constructed by including more terms of the Taylor expansion, however, since each step must compute the Taylor terms for each pair of bodies and successive terms require more calculations (in the case of gravitational dynamics), this quickly grows in computational complexity (see expression for jerk (12)). This in itself wouldn't be prohibitive if the reduction in error was sufficient to offset additional CPU time when compared with cheaper methods with smaller timesteps.

It would be better instead to find a higher order method that behaves similarly to the Symplectic Euler discussed earlier. In common with the Symplectic Euler, **Hermite** integrators introduce interdependency between current and next timestep and achieve higher order accuracy without incurring the costs of computing higher order Taylor terms. The 4th order (global error $\propto \Delta t^4$) Hermite method is as follows:

$$x_1 = x_0 + \frac{1}{2}(v_0 + v_1)\Delta t + \frac{1}{12}(a_0 - a_1)\Delta t^2 \tag{10}$$

$$v_1 = v_0 + \frac{1}{2}(a_0 + a_1)\Delta t + \frac{1}{12}(\dot{a}_0 - \dot{a}_1)\Delta t^2 \tag{11}$$

Where $\dot{a}$ or *jerk* (from the time derivative of (5)) is:

$$\dddot{\vec{x}}_i = G \sum_{i \neq j} m_j \frac{\vec{x_{ij}}^2 \vec{v_{ij}} - 3\vec{x_{ij}}(\vec{x_{ij}} \cdot \vec{v_{ij}})}{|\vec{x_{ij}}|^5} \qquad \text{where } \vec{v_{ij}} = \vec{v}_j - \vec{v}_i \tag{12}$$

(See implementation Appendix B.2). To approximate position and velocity at time $t + \Delta t$ requires acceleration and jerk at $t + \Delta t$, but to compute acceleration and jerk at $t + \Delta t$ requires position and velocity at $t + \Delta t$, so the scheme is implicit. To resolve this, a lower order method is used to predict position and velocity at $t + \Delta t$ and calculate acceleration and jerk from the result. The predicted acceleration and jerk can then be used to find a better estimates of position and velocity from the Hermite equations. To

achieve greater accuracy this process can be repeated: the Hermite estimates can be fed back into the calculation of predicted acceleration and jerk which can in turn be used in the Hermite equations, however each iteration and recalculation of acceleration is expensive and weakens the advantage over other methods. The lower order predictor used in this paper is simply a Taylor expansion to the third time derivative (jerk).

Unlike the Symplectic Euler method, the Hermite method isn't symplectic but it is *time-symmetric* (Aarseth & Aarseth 2003, p.27). Consequently, if the timestep is inverted the simulation will play out in reverse, tracing back all the steps exactly. While this quality results in good conservation of physical quantities, symplectic integrators of the same order will offer better conservation.

For comparison, a lower order (2nd) but *symplectic* scheme can be derived from the Hermite equations by truncating the quadratic terms and substituting v_1 in (10) for its predictor; Taylor expansion $v_0 + a_0\Delta t$ (Lambert 1973, p.201 eq(4)) giving:

$$x_1 = x_0 + v_0\Delta t + \frac{1}{2}a_0\Delta t^2 \tag{13}$$

$$v_1 = v_0 + \frac{1}{2}(a_0 + a_1)\Delta t \tag{14}$$

This 2nd order method can also be derived from solving a simplified Hamiltonian exactly (Dehnen & Read 2011, 2.3.1), explaining its symplecticity. It's typically implemented using alternative forms of (13) and (14) which involve advancing position and velocity $\frac{1}{2}\Delta t$ out of phase with one another (removing the need to store acceleration):

$$x_1 = x_0 + v_{\frac{1}{2}}\Delta t \tag{15}$$

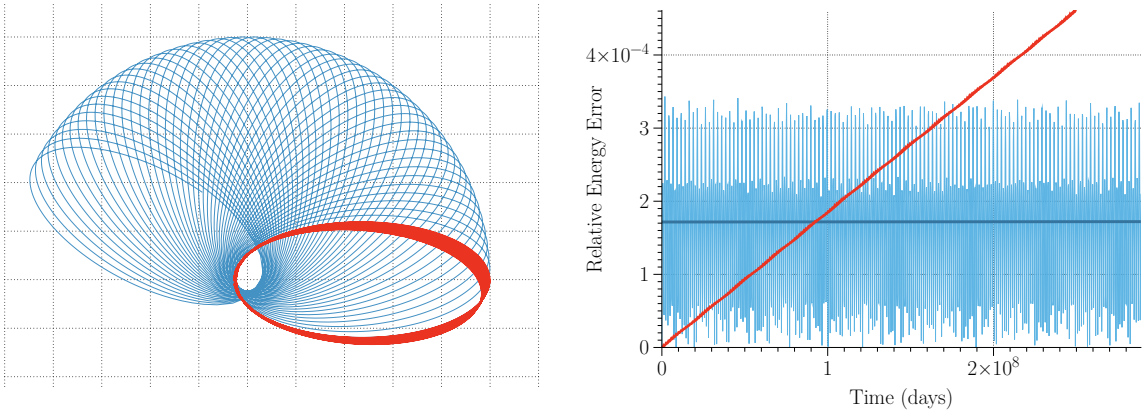
$$v_{1+\frac{1}{2}} = v_{\frac{1}{2}} + a_1\Delta t \tag{16}$$

(See implementation Appendix B.3). In this form the method is known as a **Leapfrog** integrator. Velocity advancements are termed 'kicks' and position, 'drifts'. To form a single self starting step they are combined as either kick-drift-kick or drift-kick-drift. The advantage of the latter is one less expensive force calculation, whereas we expect the prior to have smaller error in each step. In this paper the the kick-drift-kick variation is used.

Despite that the Leapfrog is in many ways a lower-order form of the Hermite integrator, it behaves quite differently to the Hermite. Leapfrog orbits tend to precess with a constant semi-major axis, whereas Hermite orbits show much less precession but tend to lose energy more rapidly. For illustration, figure 1a shows the path of a 0.9 eccentricity orbit evolved using the Leapfrog (blue) and Hermite (red). Figure 1b shows a comparison of the conservation of total energy in a gas giant only solar system with a timestep of 80 days. The Leapfrog's error § (blue), although noisy, when averaged remains approximately constant, this is demonstrative of it's *symplecticity* (solving an

§ The relative error used in this comparison and throughout the rest of the paper is calculated as $|E_0 - E(t)|/E_0$ where E_0 is the initial energy and $E(t)$ is the energy at time t.

approximate Hamiltonian exactly). The Hermite's error (red), on the other hand, starts out much smaller and grows steadily to overtake the Leapfrog's after about 300,000 years.



(a) Path of Earth-like planet about a Sun-like star with an eccentricity of 0.9. Evolved with a 1 day timestep over 55 years.

(b) Absolute relative error in the total energy of the Solar System consisting of only the gas giants, evolved with an 80 day timestep over 1 Myr.

Figure 1: Comparison between 2nd order Leapfrog (blue) and 4th order Hamiltonian (red)

2.5. Selection of Integrator

Arbitrarily small error can be achieved by simply decreasing the timestep of any method, however, when selecting an integrator it's important to make comparisons with a metric that reflects the conditions of the experiment. To build a stability map the system must be repeatedly evolved over Myr timescales many thousands of times and so the primary criteria for selecting an integrator is the CPU-time needed to maintain stability over these timescales. To compare the Leapfrog against the 4th order Hermite integrator, the Solar System with only the gas giants is evolved over 1 Myr with a range of timesteps, the relative error achieved for each timestep is plotted against the CPU-time in figure 2. ‖

As the figure shows, the Hermite integrator will be able to achieve energy errors smaller than $\approx 1.4 \cdot 10^{-5}$ in a shorter time than the Leapfrog over a 1 Myr timescale. The cross-over point is unique to the timescale - shorter timescales have a larger cross-over error and so the Hermite is more likely to be preferred, whereas for longer timescales the reverse is true. As timescale is increased, the cross-over would shift to the right on figure 2. The cross-overs for different timescales can be seen in table 1.

‖ Hardware used in all experiments: processor: 2.8 GHz i7, RAM speed: 1067 MHz

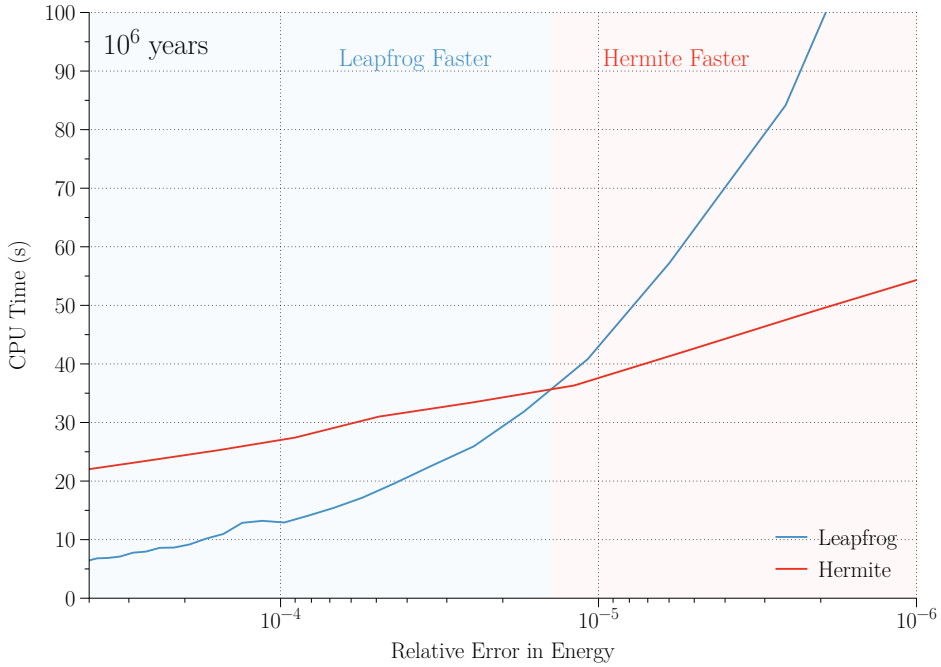


Figure 2: Relative error in the total energy against CPU-time required for Leapfrog (blue) and 4th order Hermite integrator (red) over 1 Myr. Relative error axis is reversed and logarithmic.

Timescale (years)	Cross-Over Error
10^4	$7.9 \cdot 10^{-5}$
10^5	$3.2 \cdot 10^{-5}$
10^6	$1.4 \cdot 10^{-5}$

Table 1: Cross-over relative error between favouring Leapfrog and Hermite integrators. The Hermite will outperform the Leapfrog when achieving errors smaller than the cross over error.

It was found that, for the minimum timescale used in this experiment, 1 Myr, the energy error required to maintain stability should be $\sim 10^{-4}$. With these conditions and with reference to the results in figure 2, it is the Leapfrog integrator that will offer the best performance - requiring nearly half the time the Hermite would need. These results also demonstrate that if the timescales were short and high accuracy is required, the Hermite integrator would offer the best performance.

2.6. Improving performance with adaptive timesteps

To achieve the same accuracy, bodies with high acceleration require smaller timesteps than bodies with low acceleration. It is therefore possible to improve performance by advancing bodies separately with a timestep that reflects this.

To keep the implementation simple, the drift-kick-drift variation of the Leapfrog

was used as the basis for the adaptive timestep method (see implementation Appendix B.4). Bodies are assigned a timestep and grouped by powers of two, all the bodies sharing a timestep are advanced together. A scheduling approach is used where time is counted in units of the smallest timestep, a timestep group is advanced when its timestep is an integer factor of the current time - timesteps satisfying this condition are termed as '*scheduled*' in the following explanation.

At the beginning of a step all the bodies with scheduled timesteps, Δt_i, are drifted forward $\frac{1}{2}\Delta t_i$, after drifting, timestep selection occurs. Selecting the timestep after drifting is done to prevent the Leapfrog's symplecticity being lost (Quinn et al. 1997). If the new timestep is not also a scheduled timestep then is is rejected - this is required to maintain synchronisation. The timestep groups are then arranged in ascending order and bodies with scheduled timesteps 'kick' all the other bodies with timesteps greater than or equal to the kicking bodies' timestep. To minimise the need for predicting the positions of bodies with longer timesteps, the groups are aligned (in time) so that an equal number of kicks by shorter timestep bodies happen either side of the drifts of longer timestep bodies.

A planet's timestep is selected based on its expected acceleration given its proximity to the host star with the following equation (Zemp et al. 2008):

$$\Delta t_{new} = \eta\sqrt{\frac{r^3}{GM_\star}} \tag{17}$$

where r is the planet-star distance, $M_\star$ is the mass of the star and η scales the timesteps (increasing η improves performance but reduces accuracy).

We found that when simulating the Solar System, the same order of energy error can be achieved in about half the time when using adaptive timestep method (compared to the conventional Leapfrog). Although this was clearly an improvement, in its current form the adaptive method is only suitable when the Solar System is isolated. This is because encounters between bodies are not reflected in the timestep. It is for this reason that the conventional Leapfrog was used to determine the critical interaction radius. Ideally, the timestep would be selected upon acceleration but if symplecticity is to be maintained, this would require calculating acceleration at the time of the selection and thus visiting each pair of bodies in an expensive force calculation. This approach was not investigated.

3. Results

Using a timestep of 20 days, excluding mercury and interacting with a star with mass fixed at $0.5\,\mathrm{M}_\odot$, the time between destabilising interactions for the solar neighbourhood was found to be on the order of a trillion years - many times the present age of the universe. In the cores of open clusters destabilising interactions are found to occur on $\sim 600\,\mathrm{Myr}$ timescales and this figure is smaller still for globular clusters where it is $\sim 40\,\mathrm{Myr}$. These results are summarised in table 2.

Environment	n (pc^{-3})	$\bar{v}$ (kms^{-1})	$R_c(\bar{v}, 0.5\,\mathrm{M}_\odot)$ (AU)	$\bar{T}$ (Myr)
Globular Cluster Core	1000	10	150 - 300	15 - 60
Open Cluster Core	52	0.5	800 - 1200	400 - 800
Solar Neighbourhood	0.1	30	50 - 100	10^6

Table 2: Parameters $\bar{v}$ (average relative velocity) and n (stellar number density) with results R_c (critical interaction radius) and $\bar{T}$ (expected time between critical interactions) for different stellar environments. (See equation (2))

The lower bound on the critical interaction radius R_c was taken as the first approach distance at which all runs were stable. The upper bound was taken to be the approach distance at which the average relative semi-major axis change was < 0.06, for comparison, when isolated this quantity is ~ 0.0005. If the system remains stable after the closest approach it is evolved for a further 1 Myr to examine its long term stability.

Figures 3 and 4 illustrate the stability and relative change in semi-major axis for the gas giant only Solar System over a range of initial velocities from 0 to 5 kms^{-1} and approach distances from 0 to 1000 AU . It was found that to build similar maps for a solar system consisting of all the planets excluding mercury would take wall-time on the order of days because the timestep has to be many times smaller ¶. While including the terrestrial planets made the system more sensitive to instability, the impact on R_c was small and not significant enough to alter conclusions or the appearance of the maps.

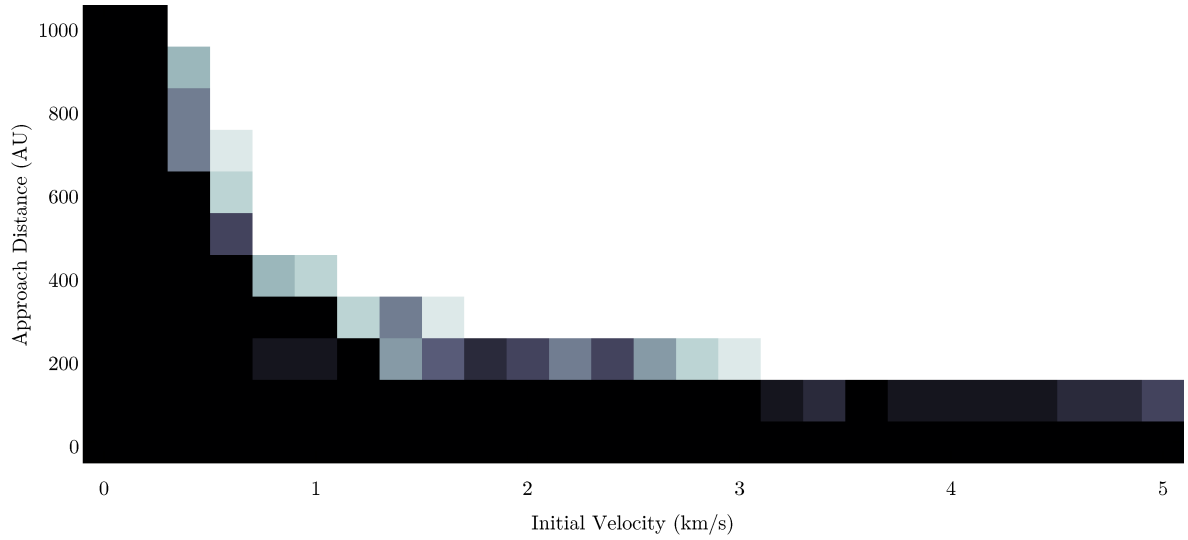


Figure 3: Stability map for the Solar System with only the gas giants considered. Each trial of closest approach d and initial velocity v was run 10 times - the black squares represent trials in which all runs were unstable, whereas in the white squares, all were stable. The gradient between them illustrates the fraction of unstable runs.

¶ The gas giant only maps took 6 hours to complete with a 96 day timestep

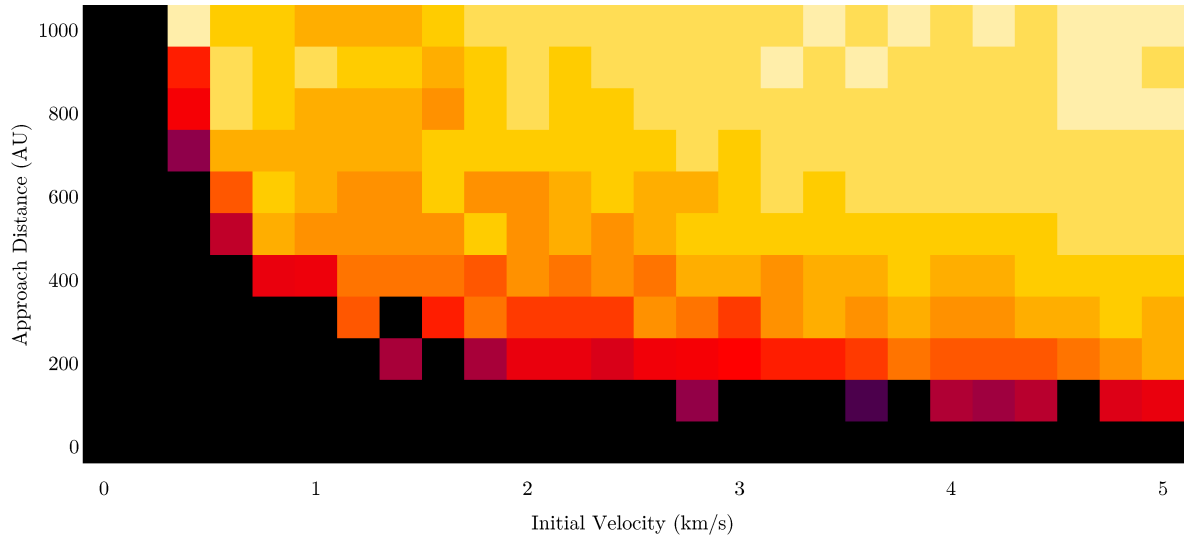


Figure 4: Perturbation map for the Solar System with only the gas giants considered. The relative change in semi-major axis (see section 2.2) is represented by the gradient of black to pale yellow through red. Black squares characterise trials in which the system received the largest perturbations of the experiment and the pale yellow squares characterise the smallest.

Figure 5 shows a typical destabilising encounter in the center of mass frame of Sun. In this example, the approach distance and initial velocity were set to 100 AU and 2 kms^{-1} respectively, falling well into the unstable regions on the stability maps. The thick green path is the neighbouring star (travelling from the top to the bottom right), although it was initially given a trajectory to approach at 100 AU, under the gravitational influence of the Sun the closest approach is actually 15 AU and its passing velocity is 9 kms^{-1}. It's approach takes it very close to Saturn's orbit, capturing it in the encounter. Energy seems to have been taken from Uranus's orbit, which drops down to cross Jupiter's. Neptune is also significantly perturbed as the star exits the Solar System, sending it into a a much wider orbit. The terrestrial planets are not shown in this figure, but they receive only a weak change to their orbits - not enough to be visible at the pictured distance.

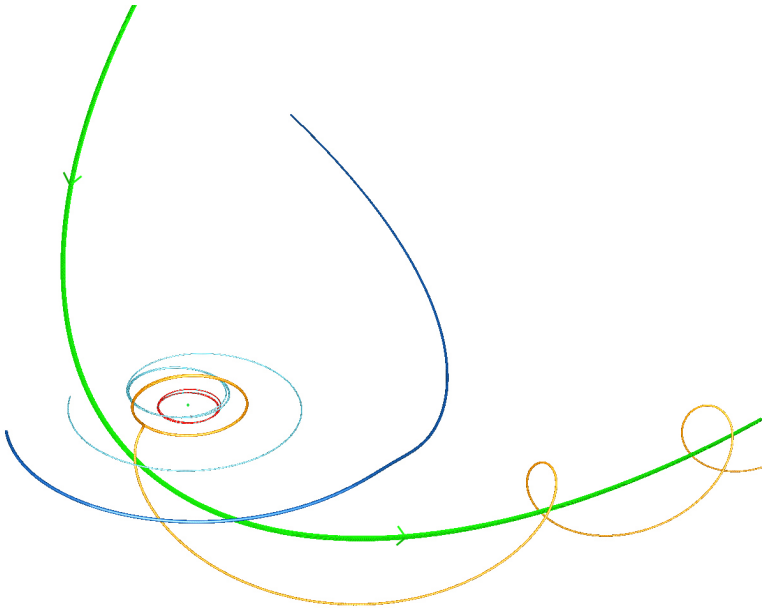


Figure 5: A typical close encounter. The thick green path is the interacting star, travelling from the top down and across to the bottom right. The orange is Saturn, red is Jupiter, cyan is Uranus and dark blue is Neptune. Unperturbed Approach distance d was set to 100 AU and initial velocity v was set to $2\,\mathrm{kms}^{-1}$.

4. Discussion and Conclusions

The results suggest that while stars with planetary systems in the solar neighbourhood will almost never experience a destabilising encounter, we expect stars in clusters to experience multiple during their lifetime. This indicates that Solar System-like planetary systems cannot survive in cluster environments, but it does not necessary rule out the existence of other types of planetary systems; the simulations suggest that the terrestrial planets are much less sensitive to encounters than the outer planets. As the time between critical interactions, $\bar{T}$, is proportional to $1/{R_c}^2$, (see (2)) then $\bar{T}$ is strongly dependant on R_c. In the case of the open cluster core, if $R_c \approx 200$ AU, we find that $\bar{T}$ is now on the order of 13 Gyr - enabling the long-term survival of these systems. It may be the case then, that tightly orbiting systems with planets in orbits similar to Venus and Mercury are common in open cluster environments. However, there is no evidence to suggest these systems are habitable, and so they may be poor environments to search for life.

Very little observational evidence of planets in open clusters exists, however, the planets that have been discovered tend to have masses on the order of Jupiter's, in orbits with high eccentricities ($\gtrsim 0.3$), very small semi-major axes, < 0.05 AU (Brucalassi et al. 2014)(Meibom et al. 2013) and about stars with masses similar to, or of greater than mass of the Sun. This evidence, however, is subject to the sweeping observational biases of radial velocity detection methods and so offers no conclusions except that

these systems can exist for long timescales in open clusters.

As orbital resonance is a crucial element in the stability and instability of planetary systems (Barnes & Quinn 2004), it is a concern that the Leapfrog integrator exhibits such significant precession. This may act to weaken resonances and make the systems more stable than they should be. If this is the case, then the results may present a Solar System that is more resistant to encounters.

4.1. Further Work

It's clear that the impact of precession needs to be investigated and improved. The adaptive timestep Leapfrog integrator offers much less precession, however, in its current form the timestep choice is based on distance to the Sun. Consequently, strong interactions with other bodies are not reflected in the planet's timestep. This can be tackled by developing a timestep criterion that uses acceleration instead.

Another possible route to improve the performance of the Leapfrog integrator would be to swap over to a Hermite integrator in favourable situations such as close encounters - as the results in section 2.5 indicate, the Hermite outperforms the Leapfrog when high accuracy is required over short timescales.

Since only the Solar System's stability is explored in the presented results, many questions are left unanswered. In a future study, for example, planetary systems could be parametrised into characteristics such as the spread of the planets, their skews towards wide or tight orbits and the mass of the host star. The stability of variations of these parameters could be mapped and the *types* of planetary systems expected to exist in different stellar environments could be better constrained.

For a more complete theoretical understanding, simulations of encounters during the formation phase should be carried out using techniques like smoothed-particle hydrodynamics. Together with more comprehensive studies of velocities and approach distances in clusters, it will be possible to generate expected distributions of the different types of planetary systems in different stellar environments. Accurate distributions will allow observational biases to be better quantified when studying planetary systems in these environments.

5. Acknowledgement

NASA JPL's HORIZONS data was used for the initial conditions of the Solar System. Thanks to my supervisior Dr Simon Goodwin for his support and advice on the project.

Appendix A. Randomly Placing A Neighbouring Star

A random point is chosen on a sphere with radius d (the approach distance) centerd on the Sun. To ensure the points are picked with an even distribution, polar coordinates are generated using equations: $\theta = cos^{-1}(2u - 1)$, $\phi = 2\pi v$ where u and v are random

numbers ranging from 0 to 1. The initial position of the neighbouring star is then found by extending tangentially to the sphere until at a distance $10,000\,\mathrm{AU}$ from the Sun.

Appendix B. Integrators in Haxe

The complete source code can be obtained at github.com/haxiomic/Planetary-Stability-N-Body-Algorithms

Appendix B.1. SymplecticEuler

```
class SymplecticEuler extends Simulator{
    function step(){
        var dSq        : Float;
        var fc         : Float;
        for (i in 0...bodyCount){
            //Pairwise interaction
            for (j in i+1...bodyCount) {
                position.difference(i, j, r);
                dSq  = r.lengthSquared();
                fc   = G / dSq;
                r *= 1/Math.sqrt(dSq);//normalize

                velocity.addProductVec3(i, r, fc*mass[j]*dt);
                velocity.addProductVec3(j, r, -fc*mass[i]*dt);
            }

            position.addFn(i, inline function(k) return
                velocity.get(i,k)*dt
            );
        }

        time+=dt;
    }
}
```

Appendix B.2. Hermite4thOrder

```
class Hermite4thOrder extends Simulator{
    function step(){
        for(i in 0...bodyCount*3){
            oldPosition[i] = position[i];
            oldVelocity[i] = velocity[i];
            oldAcceleration[i] = acceleration[i];
            oldJerk[i] = jerk[i];
        }

        predict();
        for(i in 0...2){
            evaluate();
            correct();
        }

        time+=dt;
    }

    inline function predict(){
        for (i in 0...bodyCount){
            //x1 = x0 + v*dt + (1/2)a*dt^2 + (1/6)j*dt^3
            position.addFn(i, inline function(k) return
                velocity.get(i, k)*dt +
                acceleration.get(i, k)*dt*dt/2 +
                jerk.get(i, k)*dt*dt*dt/6
            );
            //v1 = v0 + a*dt + (1/2)j*dt^2
```

```
            velocity.addFn(i, inline function(k) return
                acceleration.get(i, k)*dt +
                jerk.get(i, k)*dt*dt/2
            );
        }
    }

    inline function correct(){
        for (i in 0...bodyCount){
            velocity.setFn(i, inline function(k) return
                oldVelocity.get(i, k) +
                (oldAcceleration.get(i, k) + acceleration.get(i, k))*dt/2 +
                (oldJerk.get(i, k) - jerk.get(i, k))*dt*dt/12
            );

            position.setFn(i, inline function(k) return
                oldPosition.get(i, k) +
                (oldVelocity.get(i, k) + velocity.get(i, k))*dt/2 +
                (oldAcceleration.get(i, k) - acceleration.get(i, k))*dt*dt/12
            );
        }
    }

    inline function evaluate(){
        var d          : Float;
        var dSq        : Float;
        var dvDotR_dSq : Float;
        var fc         : Float;
        var fcj        : Float;
        //Reset accelerations and jerks
        for (i in 0...bodyCount){
            acceleration.zero(i);
            jerk.zero(i);
        }

        //Pairwise interaction a & a dot
        for (i in 0...bodyCount) {
            for(j in i+1...bodyCount){
                position.difference(i, j, r);
                velocity.difference(i, j, dv);

                dSq  = r.lengthSquared();
                d    = Math.sqrt(dSq);

                dvDotR_dSq = dv.dot(r)/dSq;

                //force factor
                fc  = G / dSq;
                fcj = fc / d;

                jerk.addFn(i, inline function(k) return
                    fcj*mass[j]*((dv[k] - 3*dvDotR_dSq*r[k]))
                );
                jerk.addFn(j, inline function(k) return
                    -fcj*mass[i]*((dv[k] - 3*dvDotR_dSq*r[k]))
                );

                r *= 1/d;//normalize

                acceleration.addProductVec3(i, r, fc*mass[j]);
                acceleration.addProductVec3(j, r, -fc*mass[i]);
            }
        }
    }var dv:Vec3 = new Vec3();//object pool
}
```

Appendix B.3. Leapfrog

```
class Leapfrog extends Simulator{
    function stepKDK(){
        var dSq        : Float;
        var fc         : Float;
        for (i in 0...bodyCount){
            //Pairwise kick
            for (j in i+1...bodyCount) {
                position.difference(i, j, r);
                dSq  = r.lengthSquared();
                fc   = G / dSq;
                r *= 1/Math.sqrt(dSq);//normalize

                velocity.addProductVec3(i, r, fc*mass[j]*dt*.5);
                velocity.addProductVec3(j, r, -fc*mass[i]*dt*.5);
            }

            //Each-body drift
            position.addFn(i, inline function(k) return
                velocity.get(i,k)*dt
            );
        }

        for (i in 0...bodyCount){
            //Pairwise kick
            for (j in i+1...bodyCount) {
                position.difference(i, j, r);
                dSq  = r.lengthSquared();
                fc   = G / dSq;
                r *= 1/Math.sqrt(dSq);//normalize

                velocity.addProductVec3(i, r, fc*mass[j]*dt*.5);
                velocity.addProductVec3(j, r, -fc*mass[i]*dt*.5);
            }
        }

        time+=dt;
    }
}
```

Appendix B.4. LeapfrogAdaptive

```
class LeapfrogAdaptive extends Simulator{
    function step(){
        //Step until synchronised
        do{
            subStep();
        }while(closedCount!=bodyCount);
    }

    var s:Int = 0;//current step
    var prevSmallestSS = 1;
    var closedCount:Int = 0;
    inline function subStep(){
        var smallestSS = maxSS;
        var dt;
        var reorder:Bool = false;

        //Open
        for (i in 0...bodyCount){
            var ssOld = stepSize[i];
            if(s % ssOld != 0) continue;//continue if it's not time to step body

            var dtOld = dtFromSS(ssOld);

            //Drift forward
            position.addFn(i, inline function(k) return
                velocity.get(i,k)*dtOld*.5
```

```
            );

            //Select
            var ssNew;
            if(i == mostMassiveIndex){
                ssNew = 16;
                stepSize[i] = ssNew;
            }else{
                ssNew = pickSS(i);
                if(s % ssNew == 0 && ssNew != ssOld){//both bodies are 'closed' - step size can only change at points of sync
                    stepSize[i] = ssNew;

                    //correct position
                    dt = dtFromSS(ssNew);
                    position.addFn(i, inline function(k) return
                        velocity.get(i,k)*(dt-dtOld)*.5
                    );
                }
            }

            if(ssNew != ssOld)reorder = true;

            if(stepSize[i] < smallestSS)
                smallestSS = stepSize[i];
        }

        //Order
        if(reorder)orderedIndicies.sort(ssAscending);

        //Pairwise kick, from smallest to largest step size
        var dSq        : Float;
        var fc         : Float;
        for (k in 0...bodyCount) {
            var i = orderedIndicies[k];
            if(s % stepSize[i] != 0) continue;//continue if it's not time to step body

            dt = dtFromSS(stepSize[i]);

            for (l in k+1...bodyCount) {
                var j = orderedIndicies[l];

                position.difference(i, j, r);
                dSq  = r.lengthSquared();
                fc   = G / dSq;
                r *= 1/Math.sqrt(dSq);//normalize

                velocity.addProductVec3(i, r,  fc*mass[j]*dt);
                velocity.addProductVec3(j, r, -fc*mass[i]*dt);
            }
        }

        closedCount = 0;
        //Close
        for (i in 0...bodyCount){
            if(Std.int(s+smallestSS) % stepSize[i] != 0) continue;//continue if it's not time to step body

            //Each-body drift
            dt = dtFromSS(stepSize[i]);
            position.addFn(i, inline function(k) return
                velocity.get(i,k)*dt*.5
            );

            closedCount++;
        }

        time += dtFromSS(smallestSS);
        s += smallestSS;
        s = s%maxSS;//wrap around
        prevSmallestSS  = smallestSS;
    }
```

```
    inline function pickSS(i:Int){
        var dSq     : Float;
        var dCu     : Float;
        var idealDt : Float;
        var idealSS : Float;

        position.difference(i, mostMassiveIndex, r);
        dSq = r.lengthSquared();
        dCu = dSq*Math.sqrt(dSq);
        idealDt = accuracyParameter*Math.sqrt(dCu/(G*mostMassiveMass));

        idealSS = idealDt/dtBase;

        return idealSS < maxSS ? floorBase2(idealSS) : maxSS;
    }

    inline function dtFromSS(ss:Int):Float return ss*dtBase;

    inline function floorBase2(x:Float):Int{
        var br:Int = 0;
        var y:Int = Std.int(x);
        while((y >>= 1) > 0) ++br;
        return 1 << br;
    }

    inline function ssAscending(i:Int, j:Int):Int
        return stepSize[i] - stepSize[j];//a-b => smallest to largest
}
```

References

Aarseth, S. J. & Aarseth, S. J. 2003, Gravitational N-Body Simulations, Tools and Algorithms (Cambridge University Press)

Barnes, R. & Quinn, T. 2004, The Astrophysical Journal, 611, 494

Brucalassi, A., Pasquini, L., Saglia, R., et al. 2014

Dehnen, W. & Read, J. I. 2011, The European Physical Journal Plus, 126, 1

Gunduz, H. 2010, Higher Order Symplectic Methods For Separable Hamiltonian Equations

Hairer, E., Lubich, C., & Wanner, G. 2006, Geometric Numerical Integration, Structure-Preserving Algorithms for Ordinary Differential Equations (Springer)

Lambert, J. D. 1973, Computational methods in ordinary differential equations (John Wiley & Sons Inc)

Meibom, S., Torres, G., Fressin, F., et al. 2013, Nature, 499, 55

Quinn, T., Katz, N., Stadel, J., & Lake, G. 1997, arXiv.org, 10043

Sigurdsson, S., Richer, H. B., Hansen, B. M., Stairs, I. H., & Thorsett, S. E. 2003, Science, 301, 193

Zemp, M., Moore, B., Stadel, J., Carollo, C. M., & Madau, P. 2008, Monthly Notices of the Royal Astronomical Society, 386, 1543