N-Body problem

Marcel Boersma
Shabaz Sultan

October 25, 2014

1 Introduction

Commonly used technology in everyday life is supported by satellites, e.g. GPS in car navigation, Internet and pictures from Google Maps. And with increasingly more satellites orbiting the earth it is important that we know that a new satellite will not crash into another. Or, in more general terms we need to know how objects orbiting other objects work such that we can launch more satellites.

It is not a new problem, because we have been trying to analyze how planets orbit each other in our own solar system for quite while, there is still room for improvement. The analytical solution for orbiting planets only exist for 2-body system, while in reality we have to cope with N-body systems. Approximation methods can be used only calculating is intractable by hand. Hence, the use of computers is promising for solving N-body systems. However, using computers and approximation methods comes with different types of problems, i.e. how accurate is my approximation method as well as the physical limitations of computers in terms of floating point operations [DR11].

This research aims to provide insight with regards to the approximation methods used for N-body systems, i.e. how accurate is the simulation. In section 2 the background of the analytical models used are explained and section 3 introduces the algorithms as well a methods for benchmarking our approximation methods. The results are shown in section 4 and section 5 our conclusion is discussed.

2 Background

When calculating the trajectory of a body it is necessary to know which forces interact with our body. Newton's seconds law, i.e. the acceleration is equal to the net force on a object, is used to describe the force on our object. And Newton's third law, i.e. every action has an equal and opposite reaction, is used.

2.1 Newton's laws

Newton's second law describes that the force on an object is equal to the change of momentum in time. Because we only consider constant mass systems this is equal to the mass times the acceleration of an object. Thus, if the object is accelerating there is a force on our object present. Knowing that the only possible force acting on our object are originating from other bodies in the system we can modify the second law by equating the law of universal gravitation to it. Let the following equation be the force F_p on object p by object c with mass m_p, m_c respectively.

$$F_p = G\frac{m_p m_c}{r^2} \tag{1}$$

With r as the euclidean distance between object p, c. Combining Newton's second law with the law of universal gravitation gives the acceleration a_p of object p

$$\begin{aligned} a_p &= G\frac{m_p m_c}{m_p r^2} \\ &= G\frac{m_c}{r^2} \end{aligned} \tag{2}$$

. The third law states that if there is a force F_1 then there also exists a force F_2 such that $F_2 = -F_1$.

In the next section a simple 2-body system is introduced for thorough understanding the approximation methods used in later sections.

3 Methodology

To understand the consequences of our approximation methods we first need to understand the analytical solution of a 2-body system. Next, we assume a simple two body system with an analytical solution and we will apply Newton's law to obtain the trajectory of our objects. Remark that this is special simple case designed to provided better understanding of orbiting objects, this is required to understand the effects of discrete approximation methods in our simulations.

Assume we have a 2 body system called Pluto-Charon system. We assume that the whole system has a constant velocity, in our case we assume zero velocity, hence the next equation must be equal to zero

$$V = \frac{P}{M} \tag{3}$$

with P as the total momentum and M as the total mass of the system. We know that the total momentum is the sum of all forces acted upon and the mass is a constant. Therefore, our zero velocity implies that the total momentum must be equal to zero. Or more formally

$$P = m_p\overrightarrow{v_p} + m_c\overrightarrow{v_c} = 0. \tag{4}$$

Each body has several forces acted upon. Our objective is to determine the path of body p, c and we have a force vector F_x pulling a body X to the other planet and also a velocity vector v_x for each body. The product of those vectors gives us a directional vector of, for example, body X. The center of mass can be calculated relative to one of our bodies.

$$(m_p + m_c)\overrightarrow{r_m} = m_p\overrightarrow{r_p} + m_c\overrightarrow{r_c} \tag{5}$$

With r_m being the vector to the center of mass, then we can rewrite this to calculate r_m

$$\overrightarrow{r_m} = \frac{m_p \overrightarrow{r_p} + m_c \overrightarrow{r_c}}{(m_p + m_c)} \tag{6}$$

and we will calculate the $\overrightarrow{r_m}$ with as reference point our body C, then the $\overrightarrow{r_c} = 0$ because the distance between our body C and C is zero, and the $\overrightarrow{r_p} = R$ with R as the total distance between the two bodies. Then $\overrightarrow{r_m}$ becomes

$$\overrightarrow{r_m} = \frac{m_p}{M} \overrightarrow{R} \tag{7}$$

If we want to know the force of body p on body c we can use Newton's law of gravitation, see equation 2. We can fill in the masses of bodies p, c using the fact that $F_c = m_c a_c$ we obtain

$$\begin{aligned} a_c m_c &= G \frac{m_c m_p}{r^2} \\ a_c &= G \frac{m_p}{r^2} \end{aligned} \tag{8}$$

Now that we have derived the acceleration in terms of the mass of p and the distance between the bodies r, we can use this calculate the acceleration of body c. The change of position i.e. distance of body c at time Δt is given by

$$\frac{d^2}{dt^2} r_c = G \frac{m_p}{r^2} \tag{9}$$

So the new position can be derived by taking the position at time $t = 0$ plus the distance traveled during δt. This can be obtained by taking the double integral of the acceleration. This can be solved analytically when only considering two bodies [Gle13]. If you consider $n \geq 3$ bodies it becomes unsolvable analytically.

3.1 Discrete Approximation

The N-body system for more than 2 bodies can not be solved analytically, hence we must use approximation methods to calculate the solution. We want to obtain the new positions of our bodies at time $t + 1$ by using Newton's second law. We can derive the next position by adding the distance traveled in time Δt to the old position at x_t, i.e. $x_{t+1} = x_t + v_t \Delta t$ with v_t as the speed at time t. Nonetheless, we do not know speed v_t but this can be obtained by $v_t = v_t + a_t \Delta t$ and a_t is known for our body. The next section describes an even more accurate approach for x_t.

3.1.1 Taylor expansion

If we want to approximate the values of, for example, formula $f(x)$ at point a we can use a Taylor expansion defined as

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x - a)^n \tag{10}$$

when we sum to infinity the new function will be the same as the original function $f(x)$. In our case we want to approximate the formulas $x(t), v(t), a(t)$ at points near t and we can use equation 10 to describe $x(t)$ as with

$$x(t) = \sum_{n=0}^{\infty} \frac{x^{(n)}(t)}{n!}(x-t)^n \tag{11}$$

however, we are dealing with limited precision and will only use a few terms of the summation such that $x(t)$ becomes

$$x(t) = \frac{x^{(0)}(t)}{0!}(x-t)^0 + \frac{x^{(1)}(t)}{1!}(x-t)^1 + \frac{x^{(2)}(t)}{2!}(x-t)^2 + \frac{x^{(3)}(t)}{3!}(x-t)^3 \tag{12}$$

notice that $x^{(1)}(t)$ is the derivative of $x(t)$ which is equal to the velocity $v(t)$ and $x^{(2)}$ is equal to the acceleration $a(t)$. But we can also formulate a Taylor expansion for $v(t)$ and $a(t)$

$$\begin{gathered} v(t) = \frac{v^{(0)}(t)}{0!}(x-t)^0 + \frac{v^{(1)}(t)}{1!}(x-t)^1 + \frac{v^{(2)}(t)}{2!}(x-t)^2 \\ v(t) = v(t) + a(t)(x-t) + \frac{1}{2}j(t)(x-t)^2 \end{gathered} \tag{13}$$

$$\begin{gathered} a(t) = \frac{a^{(0)}(t)}{0!}(x-t)^0 + \frac{a^{(1)}(t)}{1!}(x-t)^1 \\ a(t) = a(t) + j(t)(x-t) \end{gathered} \tag{14}$$

we can calculate $j(t)$ by taking the derivative of equation 2 with respect to a. Thus obtaining more accurate results for our $x(t), v(t)$ and $a(t)$.

3.1.2 Model verification

To verify the accuracy of the simulation we can apply some fundamental conservation laws from physics to our simulation. We know based on empirical data that these laws seem to be true for the natural world, so if our simulation is a faithful model of nature these laws should also be true for our simulation. If they do not, the amount our simulation fails to conserve certain values that are supposed to be conserved can be used as a metric for the error in our simulation.

Conservation of Momentum

Momentum is defined as the mass of an object multiplied with its velocity vector. To get the momentum of a system of N-bodies you can calculate the momentum of each body and sum them all together.

$$\vec{p}_{total} = \sum_{i=1}^{n} m_i \times \vec{v_i} \tag{15}$$

As long as two interacting bodies experience equal but opposite forces the conservation of momentum tends to be pretty well preserved, even if those forces are otherwise wildly inaccurate in a simulation. On the one hand this means that a fundamental law is enforced with any algorithm that ends up with the same force with a body to body interaction for both bodies, with only the direction reversed for one of them. This will be true for the N-body algorithms covered in this paper. On the other hand this means that the conservation of momentum is not a useful error metric to check the accuracy of an algorithm.

Conservation of Energy
Energy can not be created or destroyed, which in physics is expressed in the law of conservation of energy. Energy can be converted to a different type of energy, but the total amount of energy needs to be preserved. In a system of moving point masses that only gravitationally interact (i.e. an N-body system) the energy in the system is the sum of the kinetic energies and potential energies of each of the masses in the system. This sum should remain constant in a system that has no energy added or otherwise taken away from it.

The kinetic energy of a body is the energy it has due to its motion. The total kinetic energy of a system is the kinetic energies of all its bodies summed together.

$$K = \frac{1}{2}\sum_{i=1}^{n} m_i |\vec{v_i}|^2 \tag{16}$$

The potential energy of a body is the amount of energy it experiences due to its interaction with a force field, which would be with the gravitational field in a N-body simulation. The total potential energy of a system is the sum of the potential energy of all the bodies in the system.

$$W = -\frac{1}{2}\sum_{i=1, i\neq j}^{n} G\frac{m_i m_j}{|r_i - r_j|} \tag{17}$$

The total (mechanical) energy of a system is the potential and kinetic energy of the system added together, $E = W + K$. This total energy should remain constant. In a simulation this can however go wrong. For instance a body may experience a certain strong potential energy. In reality it would only experience this energy for a very short time. But in a simulation with discrete time-steps, the time-step may be longer than the time the body is supposed to experience the potential energy for. Thus it will create extra kinetic energy without that energy being based on potential energy that would be there in reality when it interact continuously with bodies rather than at discrete time-steps.
At the start of a simulation the initial energy in the system can be calculated. While running the simulation the energy can be calculated again. It should match the starting energy. The difference with the starting energy is the absolute energy error, and the

difference divided by the current energy is the relative energy error.

$$E_{error} = \frac{E_{current} - E_{initial}}{E_{current}} \tag{18}$$

This relative energy error metric can be used to see how the size of the error progresses during a simulation with a particular algorithm and how the accuracy of one algorithm compares with an other algorithm.

3.2 Algorithms

The N-body problem involves every body in a system interacting with every other body through gravity. This means that there are $\mathcal{O}(n^2)$ interactions that need to be calculated. Very broadly N-body simulators fall into two categories. The first category contains direct solvers, which simulate the gravitational interactions between all bodies directly, e.g. phiGRAPE [HGM+07].
Alternatively, there are solvers that use a tree to represent the set of bodies in a hierarchical manner. These algorithms have each of the bodies interact with the tree instead of the full set of N-bodies. This can lead to a computational complexity of $\mathcal{O}(n\log(n))$. The Barnes-Hut algorithm is a pioneering example of this class of N-body simulation algorithms [BH86].
We have opted to go for a direct simulator, because the implementation of such algorithms tends to be less complex. The important part in these simulations is the determination of the force each of the bodies experiences. In direct solvers, this is done the same for each body to body interaction and the total force on a particular body is a simple summation of the forces it experiences from every other body.
To determine the force between two bodies due to gravity a second order differential equation needs to be solved. Thus numerical integrators are the key component of direct solvers.

3.2.1 Integrator Algorithms

Euler's Method
The integrators need to solve a second order differential equation. Based on calculating the force at a current point in time acceleration for a body can be determined. Based on this acceleration the position can be updated. The most straightforward way to do this is to apply Euler's method. Based on an acceleration and a stepsize Δt it linearly approximates the speed at time $t + \Delta t$ from the speed at time t. Using the speed it then updates the position at time $t + \Delta t$ by linearly extrapolating from position at time t.

$$\begin{aligned} x_{t+1} &= x_t + v_t\Delta t \\ v_{t+1} &= v_t + a_t\Delta t \end{aligned} \tag{19}$$

This integrator is appealing for its simplicity, but because it is essentially a first order taylor expansion the expected error is proportional to Δt^2.

Jerk Integrator
It has been shown that through Newton's laws of gravity and mechanics you can calculate a body's acceleration based on the position of the other bodies. Acceleration is the second derivative of position, so it needs to be integrated twice to update the position of a body.

It is however also possible to directly calculate the third derivative of the position, the so-called jerk. The jerk is also the first derivative of the acceleration and can thus be obtained by differentiating the acceleration once with respect to time.

$$\vec{j_i} = \frac{d}{dt}\vec{a_i} = \frac{d}{dt}\left(G\sum_{k=1,k\neq i}^{n}\frac{m_k(\vec{r_k}-\vec{r_i})}{|\vec{r_k}-\vec{r_i}|^3}\right) \tag{20}$$

$$= G\sum_{k=1,k\neq i} m_k\left(\frac{\vec{v_k}-\vec{v_i}}{|\vec{r_k}-\vec{r_i}|^3} - 3\frac{(\vec{r_k}-\vec{r_i})\left((\vec{r_k}-\vec{r_i})\cdot(\vec{v_k}-\vec{v_i})\right)}{|\vec{r_k}-\vec{r_i}|^5}\right) \tag{21}$$

Using the calculated jerk, the updated acceleration, velocity and position can be approximated.

$$\begin{aligned} x_{t+1} &= x_t + v_t\Delta t + \frac{1}{2}a_t(\Delta t)^2 + \frac{1}{6}j_t(\Delta t)^3 \\ v_{t+1} &= v_t + a_t\Delta t + \frac{1}{2}j_t(\Delta t)^2 \\ a_{t+1} &= a_t + j_t\Delta t \end{aligned} \tag{22}$$

Leapfrog Integrator
The leapfrog integrator calculates the positions and velocities of the bodies in a simulated system at interleaved time-steps.

$$\begin{aligned} x_{t+1} &= x_t + v_{t+1/2}\Delta t \\ v_{t+1/2} &= v_{t-1/2} + a_t\Delta t \end{aligned} \tag{23}$$

According to the literature (e.g. [HMM95]) this can be rewritten in a form where all variables are evaluated at the same time.

$$\begin{aligned} x_{t+1} &= x_t + v_t\Delta t + \frac{1}{2}a_t(\Delta t)^2 \\ v_{t+1} &= v_t + \frac{a_t + a_{t+1}}{2}\Delta t \end{aligned} \tag{24}$$

One of the appealing characteristics of the leapfrog integrator is that it is time-symmetric. This means that if you let it run forward for a certain number of steps and then run it in reverse it should end up at the same initial starting state. This is supposed to improve its energy conservation properties.

Handling Close Encounters
In a gravitational simulation with discrete time-steps strange things can happen if two bodies get really close to each other. If two bodies are really close to each other they may experience a very strong force interaction. In reality this would happen for a very short time before the bodies either collide or they move away from each other again and experience a decreased amount of force.
The N-body simulation that has been studied uses a collisionless N-body model. This means that the first option for close encounters (collision) is disregarded. A simulation with discrete time-steps will assume that the force experienced at the start of a time-step by a body will be experienced by that body during the whole time-step. This is not necessarily true for two close bodies in collisionless system. The bodies would only experience the strong force for a much shorter time than the time-step actually last and thus the force interaction gets overestimated.
One option to improve accuracy of close encounters is to have the simulation run with smaller time-steps. This would however increase computational load and be inefficient if most simulation steps don't actually have encounters close enough to need a shorter time-step.
Instead a dynamic time-step system can be devised. By looking at each gravitational interaction in the system, the resulting speeds and accelerations between every pair of bodies due to force interaction can be derived. It can then be calculated how long it would take to cover the distance between the two bodies based on their force interaction. If the length of time is shorter than the default time-step length, the time-step for the simulation is lowered for the next simulation step.
This lowering of the time-step is done globally, for the entire simulation. The simulation is slowed down based on the closest interaction between two bodies in the system. There are algorithms that lower the time-step per body, but those have not been looked at.
The Euler and Jerk integrators can run in a optional dynamic time-step mode. The leapfrog because of its interleaving nature cannot legitimately be run in a dynamic time-step mode in the version implemented and has thus only been run in a fixed time-step mode.
Additionally a softening factor has been added to the calculations of forces between two object. The calculation for acceleration (and jerk) divides by a power of the distance between two bodies, which means that if the distance gets close to zero the force gets multiplied by a very big number. To soften this and make sure that (as far as the force calculations go) can't get infinitely close a small ϵ is added to the distance between two bodies in these calculations.

3.3 Implementation

The simulation was implemented in a C++11 application, with SDL2 and OpenGL for the visualisation. Calculations have been done with single precision floating point math, though the application is programmed in such a way that it is relatively easy to change this to double precision floating point math later.

The implementation uses sequential (i.e. non-parallel) code to implement the studied N-body simulation algorithms. Data was written to textfiles and analyzed using R and Gnuplot.

4 Experimental Results

The results of each model implementation are compared using the metrics designed in section 3.1.2. Using these methods allows us to compare the performance of each model. For all the experiments with energy comparisons we used a fixed total *model evaluation time*, i.e. we simulate 100 model minutes. Depending on the time-steps used in each method, this might take more or less iterations to calculate, meaning that the total simulation time increases. Section 4.1 compares the relative error of the Euler method using different time-steps. Section 4.2 compares the improvement of adding an extra term in the Taylor expansion. And, section 4.3 shows the results using the Leapfrog integrator and all the models are compared in the last section 4.5.

4.1 Euler

To verify our model, we have defined the energy metrics. In this experiment the relative energy metric as defined in equation 18 is used. In total we have compared four different time-step scales to evaluate the performance effect of an increasing/decreasing time-step scale. The time-step scale used are $0.001, 0.01, 0.1, 1$ and figure 1 shows a plot of the relative energy error for the *model simulation time*. As can be seen in figure 1 there is a spike around time 18 for time-step scale 1. This error has no apparent reason and in future time-steps it will decrease and stabilize around -1. Furthermore, it is interesting to see that with smaller time-step scales the total relative energy error increases faster. Meaning, that if we run the model for an infinite time the total error will explode and produce no meaning full results. The expectation may have been that smaller time-step scales should increase the accuracy in terms of relative energy error. Therefore, we can conclude that the increase in accuracy per time-step does not outweighs the increase in iterations to complete the calculation.

4.2 Jerk Integrator

Using the same energy metrics defined in equation 18 we can do the same simulation for the third order Taylor approximation. We expect that adding an extra term in the Taylor expansion would increase our accuracy. And in figure 2 we can see that indeed slower increase in relative energy error than using bigger time-step scales. However, all the time-step scale converge to -1 after about 20% of the total model simulation time. Also, we noticed that the 0.001 time-step scale converges faster to the -1 relative energy error than the 0.0001 time-step scale. Here, we can observe that decreasing the time-step by 10 does not increase the accuracy enough to compensate for the increase in number

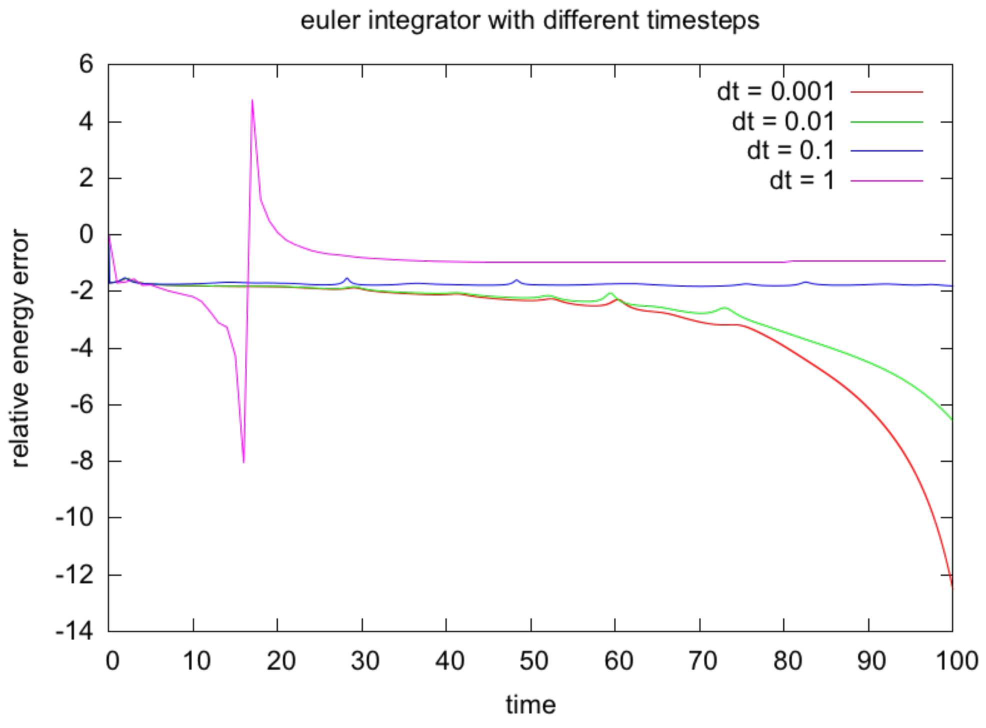

Figure 1: Relative energy error for the Euler integrator using different time-step scales for a fixed total model simulation time.

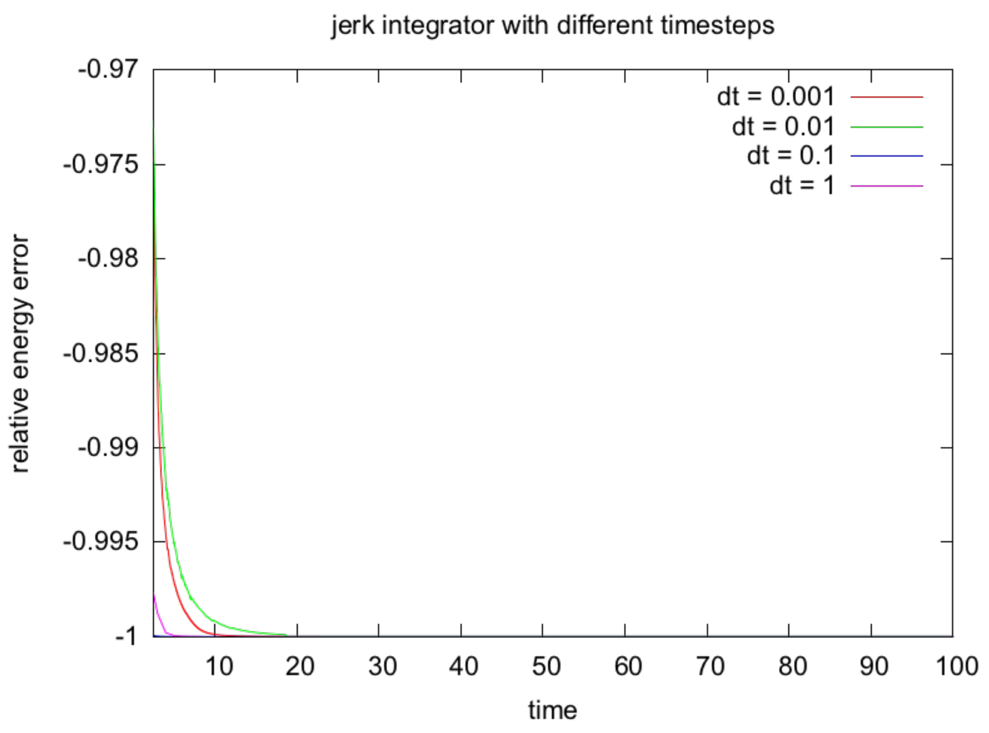

Figure 2: Relative energy error for the Taylor integrator using different time-steps for a fixed total model simulation time.

of iterations. The could be due to machine-precision, but requires further investigation to be sure.

4.3 Leapfrog

Using the Leapfrog integrator, which is a slight modified Euler integrator, we should expect similar results. And as can be seen in figure 3. the relative energy error shows similar results as for the Euler approximation in figure 1. However, the magnitude of the relative energy error is higher with the Leapfrog integrator at around time-step 18. This is probably due to the approximation of the in between time-steps which the Leapfrog integrator uses. Nevertheless, the results are relatively the same as the Euler approximation and does, at least in our simulation, lead to a decrease in relative energy error.

4.4 Dynamic time-scales

Each of the models is also run with the dynamic time-scale, which calculates the global minimum time-step scale such that any two bodies can not ghost through one another. However, our experiments showed that after just a few iterations the dynamic time-scale

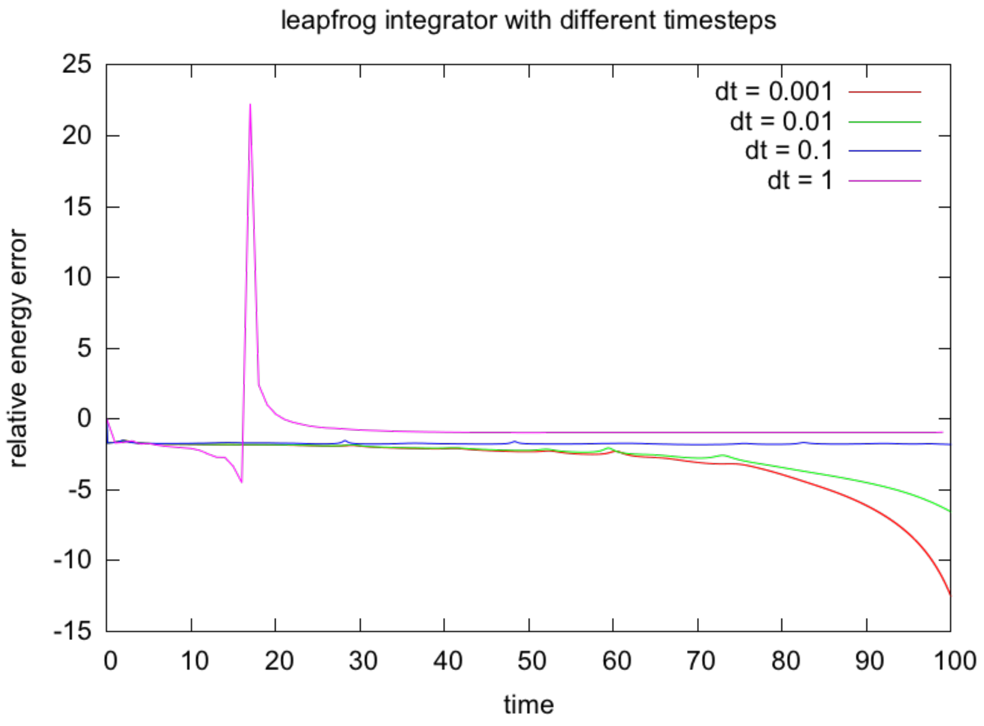

Figure 3: Relative energy error for the Leapfrog integrator using different time-steps for a fixed total model simulation time.

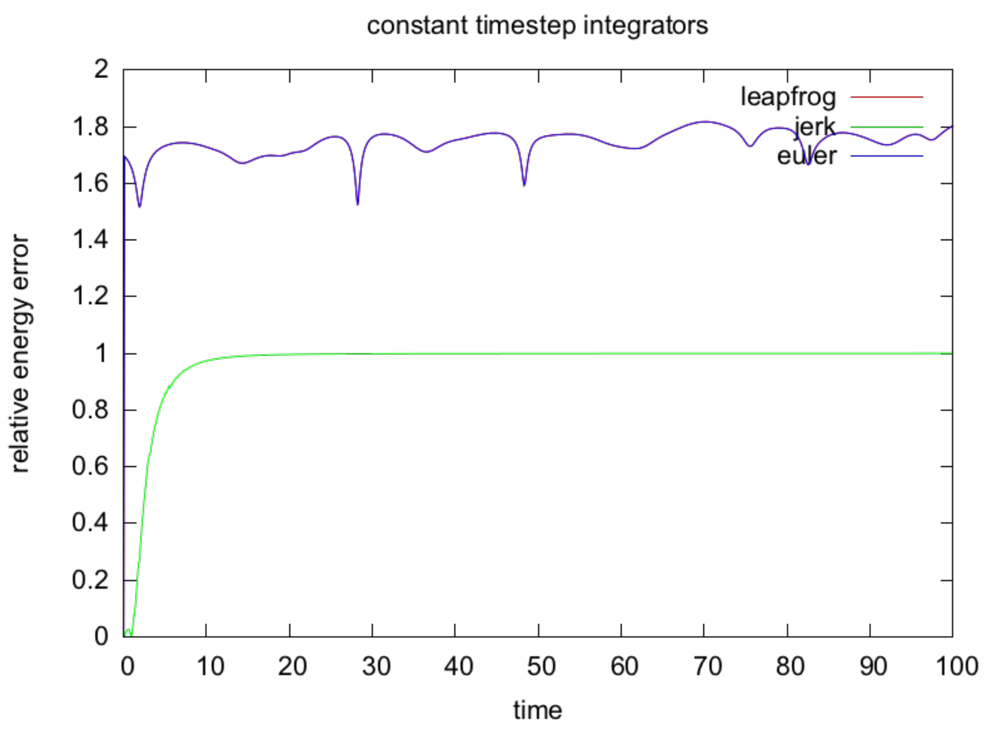

Figure 4: Relative energy error for the Euler,Taylor and Leapfrog integrator using fixed time-steps for a fixed total model simulation time.

converges to the minimum allowed time-step scale by our system. Hence, it is not useful to use the dynamic time-scales in the simulations.

4.5 All models

To compare all the models, i.e. Euler, Taylor and Leapfrog integrator, we setup up an experiment which uses the same time-step for all the models. The time-step is set at 0.01 and figure 4 shows the relative energy error for all models. Here we can clearly observe that the Taylor integrator with jerk gives the best results in terms of relative energy error. And as expected the Leapfrog and Euler integrator are approximately the same. However, the Taylor integrator comes with extra costs in computation time as can be seen in figure 5. The calculation time for calculating one extra Taylor term, i.e. the jerk, doubles the total computation time. Also, the figure shows that the increase in bodies has a quadratic behavior which is to be expected with n^2 complexity in the Euler integrator.

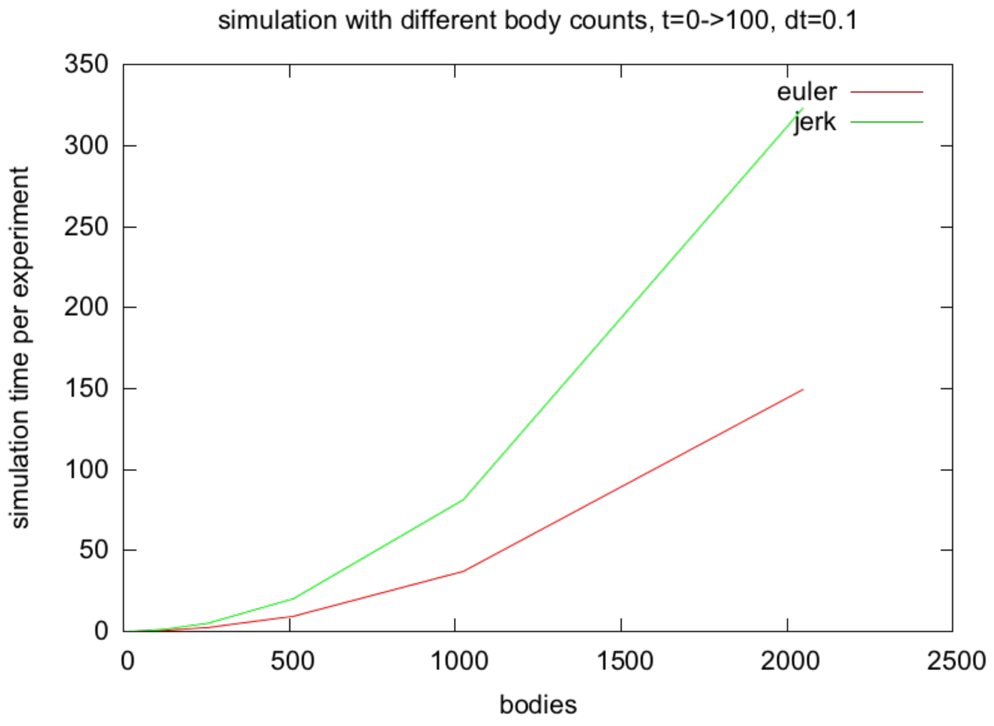

Figure 5: Computation time to finish calculating the total model simulation time for a given number of bodies.

5 Conclusions

We have simulated different integrators and found a metric to compare the different models with each other. However, there is not one best model to select. Depending on the intentions of the simulation different models might be more applicable. For example, when speed has more priority than accuracy the Euler model might be more appropriate. Nevertheless, this research shows the complex considerations you have to take into account to build a simulation model. Also, that theoretical improvements do not always lead to actual improvements in a computer simulation.

6 Future work

For future work we would like to recommend to try a distinct time-step scale for each body, meaning that close encounters have smaller time-step scale than other bodies. This should lead to a smaller energy error while not slowing down the whole process. Also, many calculations are suitable for parallelization to speed up the computing process.

References

[BH86] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *Nature*, 324:446 – 449, 1986.

[DR11] W. Dehnen and J.I. Read. N-body simulations of gravitational dynamics. *The European Physical Journal Plus*, 126(5), 2011.

[Gle13] F. Gleisner. Three solutions to the two-body problem. Bachelor's thesis, Lineaus University, 2013.

[HGM$^+$07] Stefan Harfst, Alessia Gualandris, David Merritt, Rainer Spurzem, Simon Portegies Zwart, and Peter Berczik. Performance analysis of direct n-body algorithms on special-purpose supercomputers. *New Astronomy*, 12(5):357 – 377, 2007.

[HMM95] P. Hut, J. Makino, and S. McMillan. Building a better leapfrog. *apjl*, 443:L93–L96, April 1995.