Fys4150
Project 3

Peter Killingstad and Karl Jacobsen

https://github.com/kaaja/fys4150

October 22, 2017

Note to instructurs about Github repository

If the above Github-link does not work, it is eighter because you have not yet accepted our invite to the repository, or you have not yet provided us with an e-mail adress available at Github so that we can invite you. The Github user you will be invited from is "kaaja". If the latter applies to you, please send us an e-mail with an e-mailadress available in Github or your Github username so that we can send you an invite. Our e-mailadresses: peter.killingstad@hotmail.com, karljaco@gmail.com.

Abstract

Systems of 2nd order ordinary differential equations (ODEs), describing planetary motion, are scaled and reduced to 1st order ODEs. The equations are discretized using forward Euler's method and the Velocity Verlet method, and implemented in C++ by use of classes and class inheritance. The methods is shown to involve the same number of FLOPS. Velocity Verlet is shown to be the only method the preserves energy and angular momentum, and is also higher in convergence rate order compared to forward Euler. Both methods produces reasonable planetary orbits in a two body system consisting of the sun and the earth. The velocity Verlet method reproduces exact terminal velocities for the earth, under different assumptions on the gravity force, and also gives reasonable results in a three body system both with a stationary and non-stationary sun is modelled. The full solar system is well approximated with Velocity Verlet, and the observed perihelion precession of Mercury around the sun is reproduced by adding a relativistic correction in the gravitational force.

1 Introduction

Multibody systems governed by coupled 2nd order ordinary differential equations (ODEs) appear several places in physics, among other places in mechanics and astrophysics. Systems like these can be simplified by variable subsitution, leading to a larger system of only 1st order ODEs. This technique is demonstrated and applied here.

Often the more general a method is, the more useful it is. In this project, which deals with planetary orbits, the equations are scaled to astronomical units. In addition to making the equations easier to work with, the scaling also makes the equations and their solutions more general and potentially applicable to other physical problems without the need to do large changes to the computer programs. In addition scaling was necessary in this case, since use of SI units probably would lead to numerical problems with overflow.

Solving large systems like these cannot be done analytically, and must be done numerically. Several methods are available for calculating derivatives that appears in the equation system. Knowledge about the properties of the different methods is crucial for getting quality solutions. There are several examples where lack of knowledge about numerical methods has led to dissasters in the real world. In this project we use the forward Euler method and Velocity Verlet method, and it is shown that the Forward Euler method do not obey basic physical properties such as energy and angular momentum preservation. Also it is shown that the speed of convergence of the Velocity

Verlet method is twice that of Forward Euler.

Working with many equations on the computer, can increase the probability of errors lik e.g. typing errors dramatically. Methods that reduces the probability of such errors are essential for efficient and correct solutions. In physical systems like we are dealing with here, coupled 1st order ODE systems, most of the equations are almost identical. A perfect programming method for dealing with such cases are classes. With classes on typically writes many equations that is almost identical only once. This reduces the probability of typing errors considerably! Classes is used extensively in this project.

In this project we will deal with some special situations, like introduction of a relativistic correctional term to the gravitational force on Mercury (for calculation of the periihelion precession) and other alternative gravitational forces. Introduction of such modifications can be dealt with more smoothly, compared to adding lots of if-tests inside the classes, by utlizing class inheritance. With class ineritance, one simply construct new classes that inherits all the property of the old classes and then make simple changes to new class. This method potentially increases the readability and generality of the code, the last part implying that it can be used easier with other problems.

It is easy to lose control of where problems occours and how the different parts of the program works when solving large systems. Solving the full problem directly one quickly ends up with problems localizing errors, since it is hard to find errors in large systems. A step-wise implementation, incorporating only parts of the system at a time and checking different aspects at the different steps, increases the model users knowledge about the full program and system. This step wise method is implemented in this project. First a two-body system with only the sun and the earth, where the sun is stationary, is solved. On this system several things are tested: The performance of the solversm and the escape velocity for different gravitational forces. Then a three-body system is introduced, where checking for the effects of the mass of the third planet on the orbits. Finally a real center of mass system with a moving sun is introduced in the three body system, before the full solar system is simulated.

The next section of the report describes the equations and shows the scaling of these, in addition to describing the main idea of our class setup. The proceeding section includes all results, while the final section deals with conlcusions and discussion of limitations.

2 Theory

2.1 Order reduction

Lets describe the general procedure for reducing a 2nd order ODE to a 1st order system of ODEs. Say we have an equation describing Newton's 2nd law:

$$F/m = a = \frac{d^2}{dx^2} = -kx \tag{1}$$

Now make the "substitutions" $\frac{d^2x}{dt^2} = v$ and $v = \frac{dx}{dt}$ to get the system

$$\frac{dx}{dt} = v \tag{2a}$$

$$\begin{aligned} \frac{dv}{dt} &= \frac{d^2x}{dt^2} && \text{(2b)} \\ &= \frac{F}{m} && \text{(2c)} \end{aligned}$$

In the above the 2nd order ODE is reduced to the system (2a) and (2c). Actually, this method can be used to reduce any higher order equation to a system of 1st order ODEs! The solar system is described with Newton's 2nd law, so the reduction of order for the ODEs for the solar system follows the same recepie as listed above.

2.2 Numerical schemes

Now for the numerical scehmes, forward Euler and Velocity Verlet. Both these methods are derived from Taylor polynomials. The strength of these polynomials is that one has control over the order of the truncation error, the error one gets by truncating the Taylor series to get the wanted approximation.

For the forward Euler method, a Taylor expansion around the point $x + h$ is done for x and v, and one obtain the discretized system

$$x_{i+1} = x_i + hv_i + \mathcal{O}(h^2) \tag{3a}$$

$$v_{i+1} = v_i + ha_i + \mathcal{O}(h^2) \tag{3b}$$

We see from (3a) and (3a) that the truncation error is of order h^2. However, the above is repeated N times, there are N time steps, to get the full solution. Since h is proprtional to N^{-1} ($h = T/N$), the global error goes like $\mathcal{O}(h)$. The forward Euler method is conditionally stable, meaning that is stable only for certain Δt.

The velocity Verlet method is constructed in the same way as the forward Euler method, apllying Taylor polynomials. However, a more clever manipulation of the Taylor polynomials is applied, and one ends with

$$v_{i+1} = v_i + h/2(a_{i+1} + a_i + \mathcal{O}(h^3) \tag{4a}$$

$$x_{i+1} = x_i + hv_i + \frac{h^2}{2}a_i + \mathcal{O}(h^3) \tag{4b}$$

$$a_i = F_i/m \tag{4c}$$

In the above one must update position before velocity, since we need the acceleration on the current and next step to get the velocity. Calcuating x first, we can insert this into the equation for a, and then we calculate v. We observe that the truncation error, and hence the also the global error, is one order higher compared to forward Euler. This method is unconditionally stable, meaning it is stable for all Δt. In addition, this method conserves energy.

2.2.1 Algorithm Velocity Verlet and FLOP count

The velocity Verlet method is implemented with the following algorithm

```
for time in [0,...,T]:
   1) Make acceleration vector
   for planet in planets:
      accVec[planet] = F[planet]/mass[planet]

   2) Calculate position
   for planet in planets:
      x += hv + h^2/2 accVec[planet]

   3) Update vector with old accelerations
   accVecOld = accVec

   4) Calculate new acceleration
   for planet in planets:
      accVec[planet] = F[planet]/mass[planet]

   5) Calculate velocity
   for planet in planets:
      v += v + h/2(accVecOld[planet] + accVec[planet]])
```

Now we will count the FLOPS of the above algorithm, For reference, accelerations are calculated with the following formulae

```
rPlanetDistance = getRadialDistance(*planets_[planetNumber]);

accelerationX_ += -FourPi2*planets_[planetNumber]->getMass()/planets_[0]->getMass()*(xPosition -
    planets_[planetNumber]->getXPosition())/pow(rPlanetDistance,3);
```

From the above listing, we get that each acceleration calculation involves $1(+\ =) + 1(fourPi2 + planets_{)} + 1(fourPi2 + planets_{/}planets_{[}0]-\ >\ getMass) + 1(fourPi2 + planets_{/}planets_{[}0]-\ >\ getMass * (xPosition - planets_{)}) + 1(xPosition - planets_{)} + 3(.../r^3) = 8\ FLOPS.$

With P denoting the number of planets, N denoting the number of time steps, and the use of the result above for FLOPS per acceleration calculation, the above algorithm gives the following FLOP count for the Velocity Verlet algorithm

- 1) $P * (F)_{FLOPS} = P * 8\ FLOPS$
- 2) $P * [1(+=) + 1(+hv) + 1(hv) + 2(h^2) + 1(h^2/2) + 8(F)] = 14P\ FLOPS$
- 3) 0 FLOPS
- 4) Same as 1): $8P\ FLOPS$
- 5) $P[1(+=) + 1(h/2) + 1(accVecOld + accVec) + 1(h/2 * (accVecOld + accVec)] = 4P\ FLOPS$
- SUM 1) - 5) = 34 P FLOPS
- TOTAL FLOPS = SUM*N = 34 P N FLOPS $\approx \mathcal{O}(N)\ FLOPS$

The above is not fully exact, as how one implements e.g. $4\pi^2$ into the program affects the FLOP count. However, all necessary terms of the implemented algorithm should be included for the above count to give a correct order, which is $\mathcal{O}(N)\ FLOPS$. Also, in the program x and y are separated, so we get twice as many FLOPS as calculated above. Once again, this does not effect the order properties of the FLOPS, which will be analyzed.

We will do an explicit FLOP count for Forward Euler, which is easier to count for. However, the method for counting FLOPS for Forward Euler is the same as the one applied above. From the lectures, we know that also forward Euler is $\mathcal{O}(N)$ when it comes to FLOPS, with only a few FLOPS less than Velocity Verlet.

2.3 The solar system equations

We describe the solar system by Newton's 2nd law assuming circular orbits and Newton's gravitational law, which for two objects are

$$F_G = M_{earth}\frac{v^2}{r} \tag{5}$$

$$F_G = \frac{GM_{sun}G_{earth}}{r^2} \tag{6}$$

Dividing both equations above by M_{Earth} gives

$$a = \frac{v^2}{r} \tag{7}$$

$$a = \frac{GM_{sun}}{r^2} \tag{8}$$

To make the equations simpler, and also to avoid overflow issues that potentially would occour if we had used SI units, we now scale the equatons to Astronomical units to get $v = \frac{2\pi r}{T} = \frac{2\pi Au}{Yr}$. Inserting this scaling into the above system gives

$$a = \frac{2^2\pi^2 Au^2}{Au\,Yr^2} \tag{9}$$

$$a = \frac{GM_{sun}}{Au^2} \tag{10}$$

Setting the above two equations equal and solving for GM_{sun} gives

$$GM_{Sun} = 4\pi^2(Au)^3/Yr^2 \tag{11}$$

Now we can insert (11) into (8) to get

$$a = -\frac{4\pi^2}{r^2} \tag{12}$$

For the x-component we get

$$a_x = -\frac{4\pi^2}{r^2}\cos(\theta) \tag{13a}$$

$$= -\frac{4\pi^2}{r^2 r} r\cos(\theta) \tag{13b}$$

$$= -\frac{4\pi^2}{r^3} x \tag{13c}$$

The same type of calculation gies the y-component.

The next step is to add a planet to the system. This will add a new force between earth and the new planet, in addition to the force between the sun and the planets. The force between earth and the new planet will be of the exact same kind as the force between earth and sun, given in (6), and the x accleration term (13a) for the earth will now be

$$a_x = -\frac{4\pi^2}{r^3_{earth,sun}}(x_{earth} - x_{sun}) - \frac{4\pi^2 \frac{M_{jupiter}}{M_{sun}}}{r^3_{earth,jupiter}}(x_{earth} - x_{jupiter}), \tag{14}$$

where we see that the only difference between the two terms are the mass-factor, which in the earth-sun term is one. Now Jupiter will have the same kind of expression as above, only with Jupiter instead of earth in the subscripts. Note the similarity! Adding more planets happens in exactly the same way, nothing new happens! Also we note that the 2nd term will be equall for Jupiter and earth, which potentially can be taken advantage of when solving the equaions numerically.

3 Results

Before turning to the results, we give a short overview of our class implementation.

3.1 Class implementation

DO Insert a simple figure that contains boxes of the different classes and their inouts, and arrows between boxes.

3.2 Forward Euler versus Velocity Verlet

3.2.1 Positions

The below figures show the results for the x and y-positions as functions of time, forward Euler to the left and Velocity Verlet to the right.

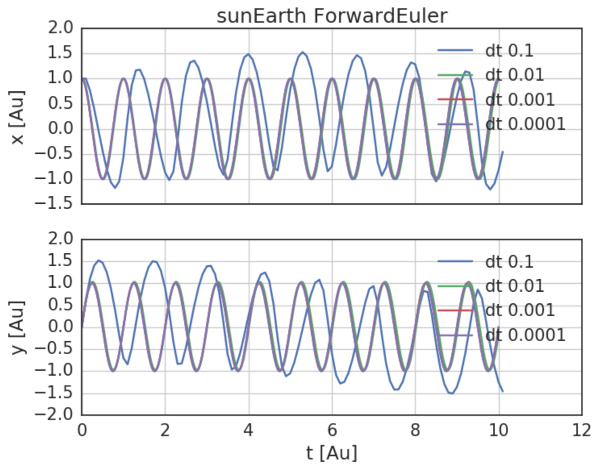

Figure 1: Sun-Earth system. Effect of Δt over a 10 year period.
The Forward Euler method seems to converge for the two smallest Δt

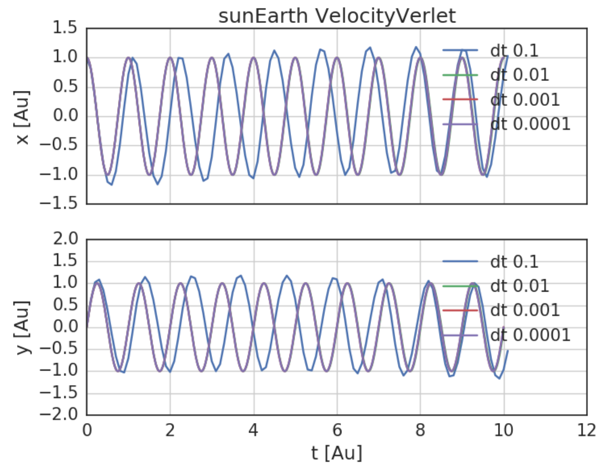

Figure 2: Sun-Earth system. Effect of Δt over a 10 year period.
The Velocity verlet method seems to converge faster than Forward Euler

As can be seen of the figures above, both methods converges when Δt is lowered. Also the Velocity Verlet method seems to converge faster.

The next two sets of figures shows the orbits for different Δt for the two methods.

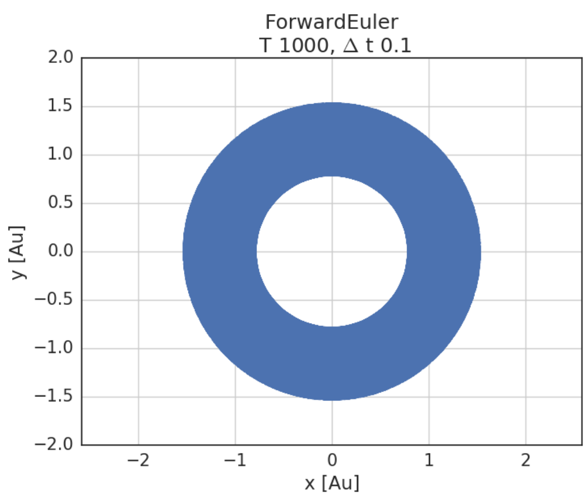

Figure 3: Sun-Earth system. Forward Euler. 1 000 years
Non-circulat orbits when time step is large.

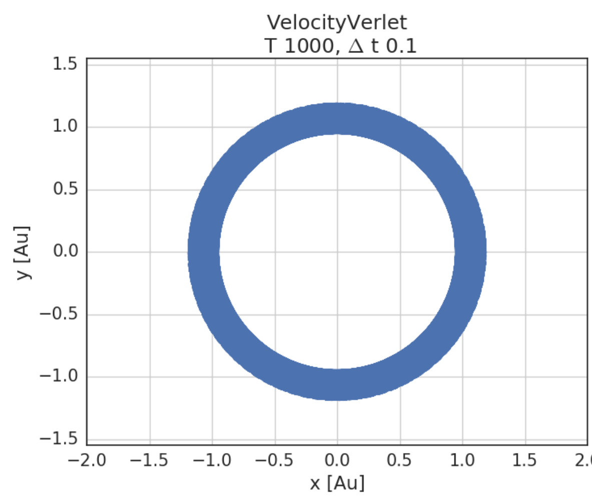

Figure 4: Sun-Earth system. Velocity Verlet. 1000 years.
Large time step gives bad solutions also for Velocity Verlet.

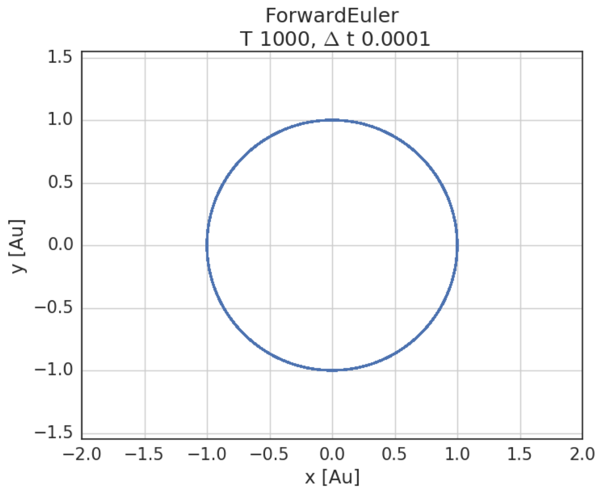

Figure 5: Sun-Earth system. Forward Euler. 1 000 years. *For $\Delta t = 0.0001$, the forward Euler seems to give circular orbits, but we can see that the solution changes.*

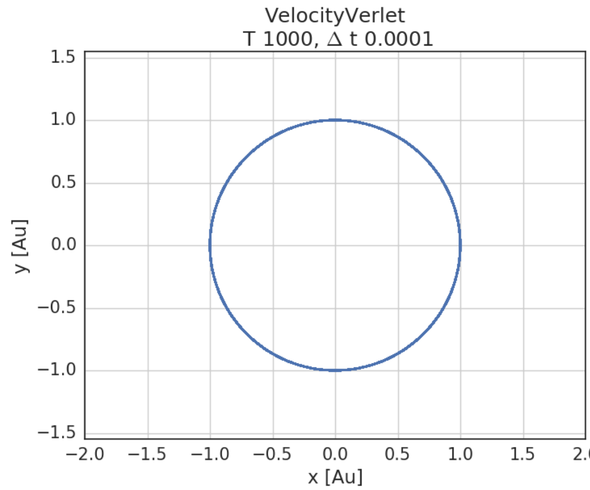

Figure 6: Sun-Earth system. Velocity Verlet. 1 000 years.
The solution looks similar to Forward Euler.

The 2 sets of figures above confirms what we found in the figures with x and y-position agains time: The precision is bettered when Δt is lowered.

3.2.2 Energy and angular momentum

Next we analyze the evolution of energy in the numerical solutions. Since all forces in our systems are conservative, meaning that the work done by the forces is path independent, total mecahnical energy, the sum of kinetic and potential energy, should be preserved. Consearvation of energy is a crucial physical principle which we want to maintain in our numerical solutions. The below figures show the difference in total energy compared to the initial total energy, forward Euler to the left and Velocity Verlet to the right.

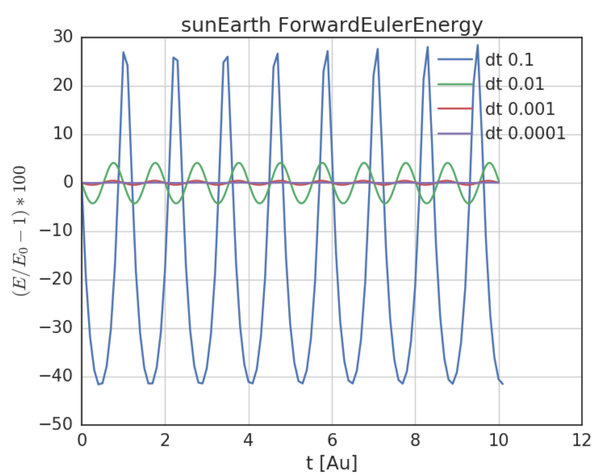

Figure 7: Sun-Earth system. Total Energy divided by total energy first time step. Forward Euler. 10 years.
Energy is not preserved with the Forward Euler method

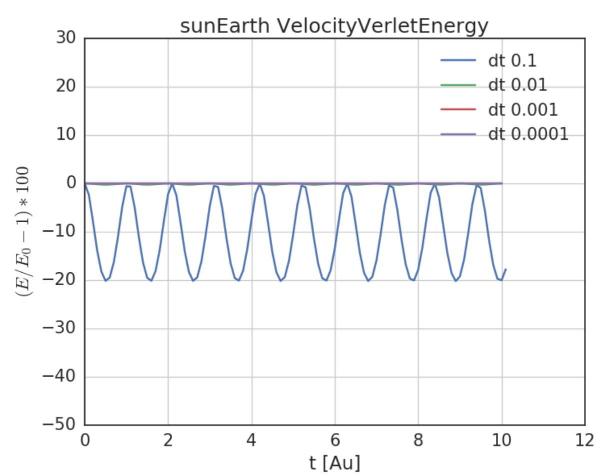

Figure 8: Sun-Earth system. Total Energy divided by total energy first time step. Velocity Verlet. 10 years.
Energy is preserved in Velocity Verlet provided fine enough time step.

The two figures above show that for forward Euler, energy can actually increase, even for smaller Δt. For Velocity Verlet, energy seems to be preserved faster compared to Forward Euler. Also, Velocity Verlet never make energy, as is the case for Forward Euler.

Another important physical property which we want to contain in the numerical simulations, is presevation of angular momentum conservation, $\frac{d\vec{L}}{dt} = \sum_i \vec{r}_i \times \vec{F_i^{ext}} = 0$. There is zero change in angular momentum, since the net external forces are zero. The next figure shows the same kind of figures as for energy above, but now for angular momentum.

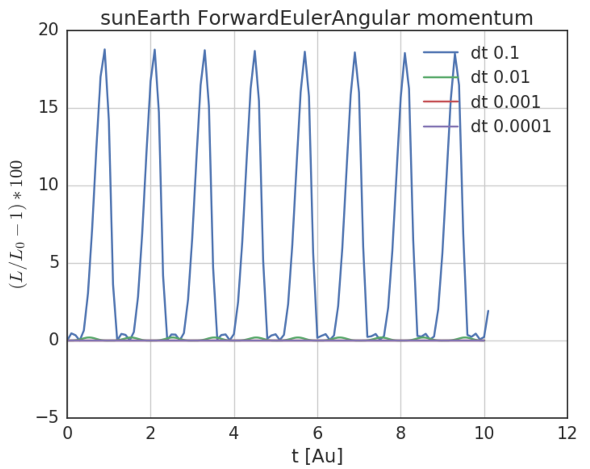

Figure 9: Sun-Earth system. Angular momentum divided by angular momentum first time step. Forward Euler. 10 years.
Angular momentum seems to be conserverd for the finest time step.

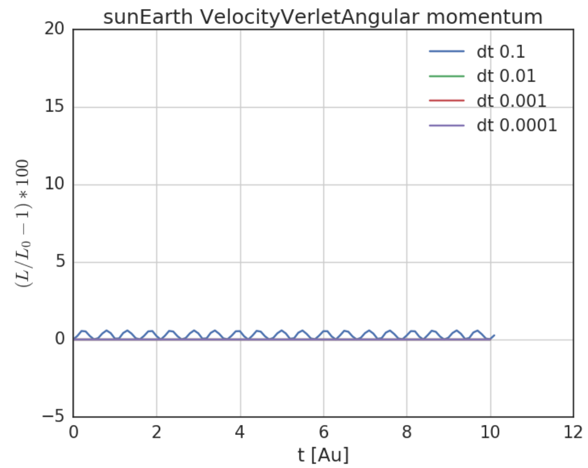

Figure 10: Sun-Earth system. Angular momentum divided by angular momentum first time step. Velocity Verlet. 10 years.
Angular momentum is conserverd given suffiently fine time steps. Conservation achieved faster than with Forward Euler.

The figures above show that as Δt gets low, angular momentum is preserved. Also, Velocity Verlet seems to give good angular momentum approximations compared to Forward Euler.

Since we know the exact solution for energy and angular momentum, or more precisely, the exact solution for the change, we have constructed a norm measure for computing convergence rates of the schemes. The error-norm is the supremum of the largest difference in energy compared to initial energy. From above we know that the global error's for x and y in Forward Euler should go as respectively $\mathcal{O}(\Delta t)$ and $\mathcal{O}(\Delta t^2)$, so the orders of global errors in position and velocity is always twice as high in the Velocity Verlet method compared to the forward Euler method. Since both energy and angual momentum depend on position and velocity, we expect the convergence rate of norms related to energy and norms related to angular momentum to be twice as high for the Velocity Verlet method compared to the forward Euler method. The figures below shows the convergence for energy norm and angular momentum norm for Forward Euler and Velocity Verlet.

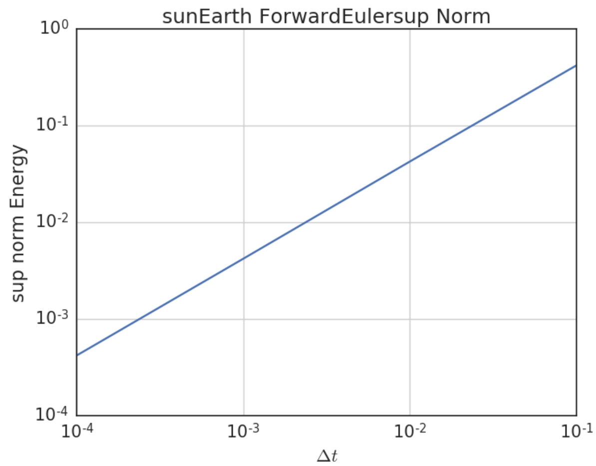

Figure 11: Sun-Earth system. Sup-norm total energy. Forward Euler.
Forward Euler's sup-norm goes like $\mathcal{O}(\Delta t)$

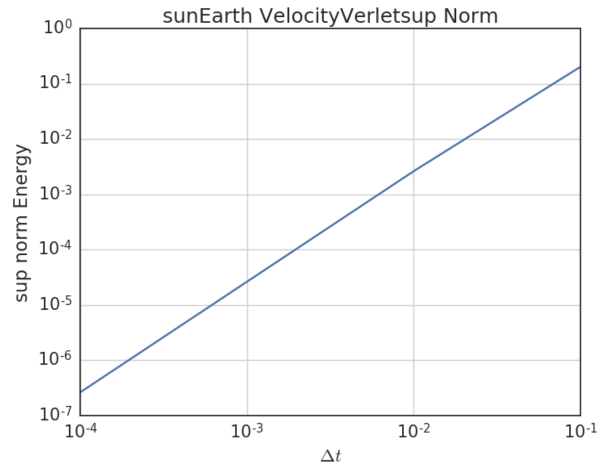

Figure 12: Sun-Earth system. Sup-norm total energy. Velocity Verlet
The sup-norm in energy for Velocity Verlet goes one higher order than Forward Euler

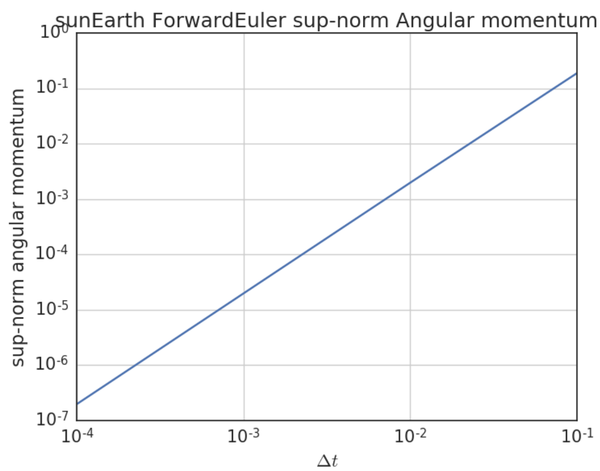

Figure 13: Sun-Earth system. Sup-norm Angular Momentum. Forward Euler
Forward Euler's sup-norm for angular momentum goes like $\mathcal{O}(\Delta t^2)$.

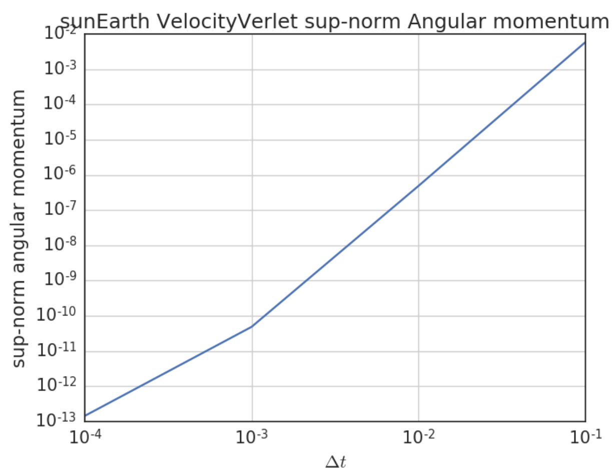

Figure 14: Sun-Earth system. Sup-norm Angular momentum. Velocity Verlet
Velocity Verlet's sup-norm error is $\mathcal{O}(\Delta t^4)$.

The figures on the first row show that the energy norm error goes as 1st order for Forward Euler, while Velocity Verlet goes like 2nd order, so the Velocity Verlet method has twice the convergence rate of forward Euler, as expected.

The figures on the 2nd row above show that also for the angular momentum norm, the order for Velocity Verlet is twice that of Forward Euler, as expected.

3.2.3 Algorithm efficiency

Knowledge about a methods correctness, discussed above, is crucial, but efficiency is also important. Now we want to see that the efficiency of the algorithms replicate the behaviour we would expect based on the FLOP-count we did in the section "Algorithm Velocity Verlet and FLOP count". In the mentioned section we calculated velocity Verlet to be $\mathcal{O}(N)$ FLOPS. Computational time should be more or less proprtional to FLOPS, so we expect the computational time to be of the same order as the FLOPS. The below figures show the computation time for the methods, forard Euler to the left and Velocity Verlet to the right.

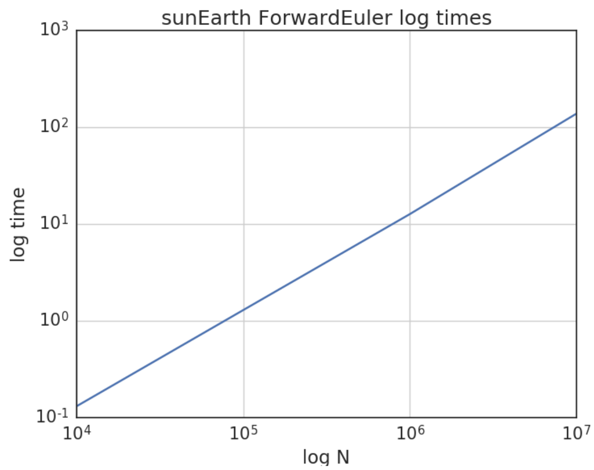

Figure 15: Sun-Earth system. Log time. Forward Euler *Forward Euler's $\mathcal{O}(N)$.*

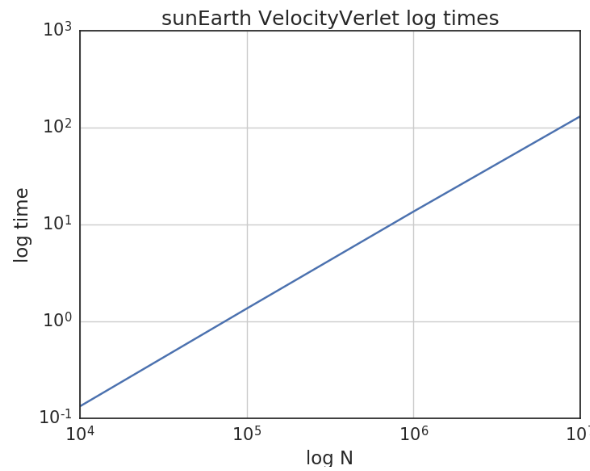

Figure 16: Sun-Earth system. Log time. Velocity Verlet *Velocity Verlet's log time is of the same order as Forward Euler.*

The two figures above show that the Forward Euler and the Velocity Verlet method both has solution times that is of order 1 in N. This is as expected.

3.3 Escape velocity

Another test to see if the solver obeys the physical laws, is to check if the escape velocity is correct. The escape velcoity equals the velcoity where potential energy equals kinetic energy

$$E_k = E_p \tag{15a}$$
$$\frac{1}{2} m_E v_{Escape}^2 = \frac{GM_{Sun}M_E}{r_0} \tag{15b}$$
$$\rightarrow v_E = \sqrt{\frac{2GM_{Sun}}{r_0}} \tag{15c}$$
$$= \sqrt{\frac{2 * 4\pi^2 AU^3}{AU(Yr)^2}} \tag{15d}$$
$$= 2\pi\sqrt{2}\frac{AU}{(Yr)} \tag{15e}$$

Based on the above equation, we expect the simulations to gove escape then $v > 2\pi\sqrt{2}$. The figures below show the results.

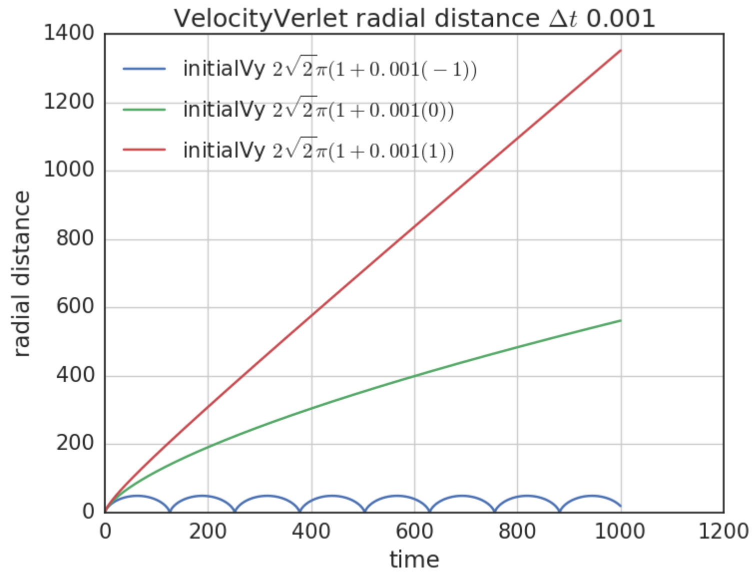

Figure 17: Sun-Earth system. Escape valocity. Radial distance earth sun.
Escape for velocity equal to 0.1% *of* $v = 2\sqrt{2}\pi$.

The figure above show that the simulated escape velocity is very close to the exact terminal velcocity, $2\pi\sqrt{2}$, supporting the previous results indicating the validity of our solver as a good approximation method.

Another test, to see if the numerical solutions corresponds with physics, is to simulate the two planet system with different gravitational forces. We noe assume the gravitational force $F_G = \frac{GM_{Sun}M_{Earth}}{r^\beta}$, $\beta \in [2, 3]$, so that the force is decreased from the original gravitational force when $\beta > 2$. This should give lower escape velocities, since the force is lower. The below figures show escapce velocities for the three different gravitational forces, all forces being lower than the standard force simulated above.

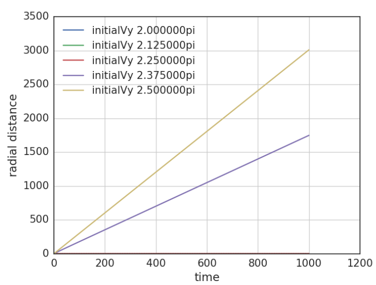

Figure 18: Sun-Earth system. Alternative force. Escape valocity. $\beta = 2.5$
Escape velocity is reduced compared to case with normal gravitiation.

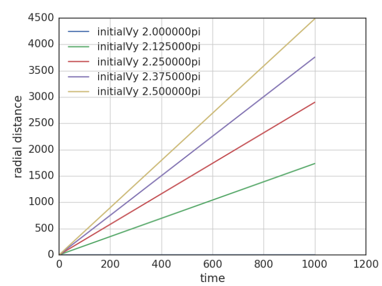

Figure 19: Sun-Earth system. Alternative force. Escape valocity. $\beta = 2.9$
The escape velocity is further reduced, and seems to be closer to 2π

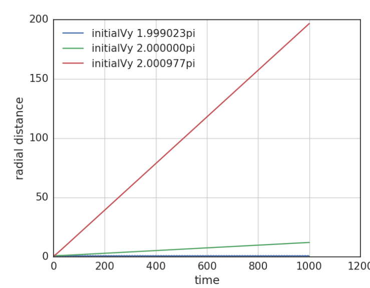

Figure 20: Sun-Earth system. Alternative force. Escape valocity. $\beta = 3.0$
We get escape at $v = 2\pi$.

The figures above show that the escape velocities are lowered when the gravitational forces are lowered, as expected.

3.4 Center of mass system

Above we have seen that the solvers correspond well with expected behavoiur: Forward Euler does not preserve energy, while Velocity Verlet does, and basic exact results for escapce velocities, energy and angular momentum is well approximated. However, the above was only tested on a two-object system. To check whether the solver works for multibody systems, with more than two objects, a third planet, Jupiter, is added.

All the above simulations were based on the assumption that the sun was stationary. This is not the case in reality. To make the solver more realistic, we now introduce movement in the sun by letting the sun be modelled just as the other planets, gravity affecting also the sun's movement.

To make the plots easy to interpret, we make sure that the center of mass of the system is stationary. This we do by adjusting the initial velocity of the sun, utilizing the fact that the angular momentum of the system is preserved

$$0 = \frac{d\vec{R}}{dt} \tag{16a}$$

$$= \frac{d}{dt}\left(\frac{1}{M}\sum_{i}^{Planets+Sun} m_i\vec{v}_i\right) \tag{16b}$$

$$= \frac{1}{M}\sum_{i}^{Planets+Sun} m_i v_i \tag{16c}$$

$$\rightarrow 0 = m_{sun}\vec{v}_{sun} + \sum_{i}^{Planets} m_i\vec{v}_i \tag{16d}$$

$$\vec{v}_{Sun} = -\frac{\sum_i^{Planets} m_i\vec{v}_i}{m_{Sun}} \tag{16e}$$

The below figure shows the results for the three-body systems mentioned above, with fixed and with moving sun, for three different masses of Jupiter.

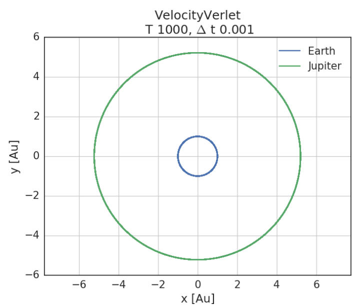

Figure 21: 3 bodies. Fixed sun. Normal Jupiter mass.
Stable orbits

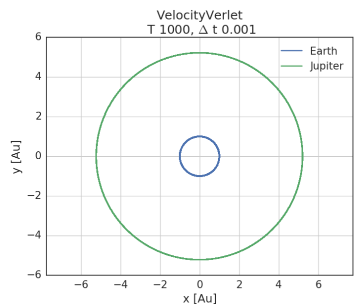

Figure 22: 3 bodies. Fixed sun. Jupiter mass times 10.
Little effect on orbits

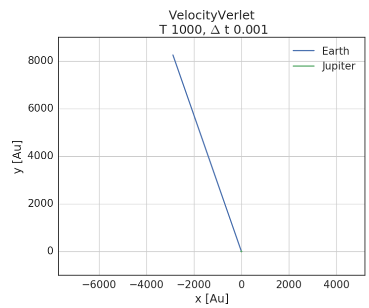

Figure 23: 3 bodies. Fixed sun. Jupiter mass times 1000.
Earth escapes.

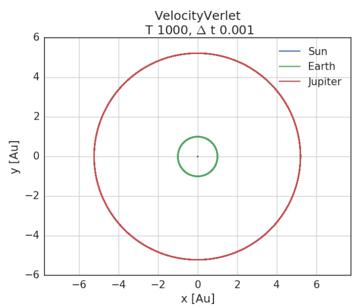

Figure 24: 3 bodies. Moving sun. Normal Jupiter mass.
Stable orbits, as for fixed sun scenario.

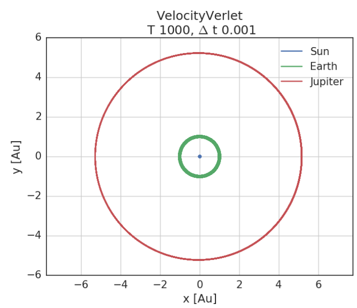

Figure 25: 3 bodies. Moving sun. Jupiter mass times 10.
Some movement in Sun, but still very similar too fixed Sun scenario.

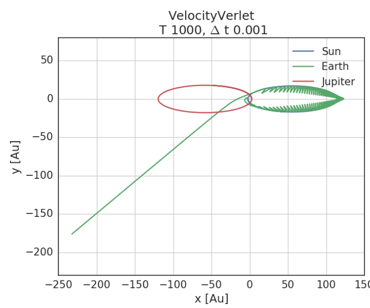

Figure 26: 3 bodies. Moving sun. Jupiter mass times 1000.
Large movement in the Sun. Earth escapes.

The first rows in the figures above shows that the solver works for 3 body systems, and that the Earth escapes in the case where Jupiter's mass equals the Sun' mass.

The 2nd row in the figures above shows the solutions for different masses of Jupiter in the case where the Sun is moving. We see that the solution for the two lowest masses of Jupiter resembles the fixed sun scenarios on the 1st row. For the case where the mass of Jupiter equals the mass of Sun, however, the solution is quite different in the case with moving sun compared to a fixed Sun. The earth still escapes, but now the Sun moves a lot, which is natural given Jupiter's mass equals the Sun's mass.

The movies below show that the center of mass is fixed in the moving sun system. To run the movies, simply write "animate threeBodiesMovingSunJupiter1000.mp4" for the moving Sun case, and "animate threeBodiesFixedSun-Jupiter1000.mp4" for the fixed Sun case.

3.5 Solar system

The solver producing reasonable results for the three-body system, we are now ready to simulate the full solar system. Since we are using classes, going from three objects to ten objects constitutes nearly no work at all! We read in all planets from file in a loop, and use the same method as earlier for adding planets. That is all. The below figures displays the results for the full solar system, with a moving sun.

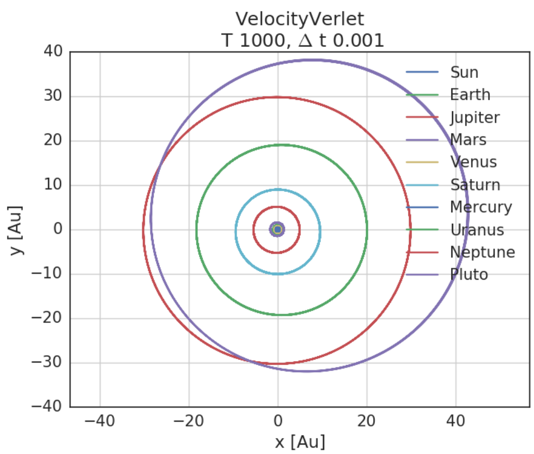

Figure 27: Solar System. Moving sun.All planets 1000 years
Orbits looks reasonable

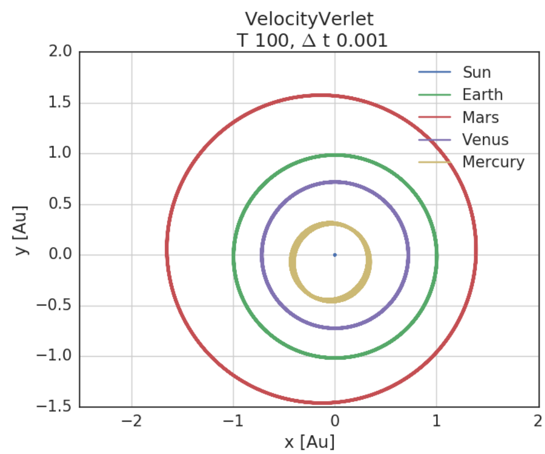

Figure 28: Solar System. Moving sun. Inner most planets. 100 years
Extreme perihelion precession for Mercury

The above figure shows that the program nicely simulates the basic dynamics of the solar system. This is as expected, since the most significant forces are included in the program. However, the right figure shows an unnatural movement for Mercury. This has to do with the ommission of the relativistc correction term in the graviational force. See next section for more on this.

3.6 Perihelion precession

Perihelion is the closest point between a planet and sun during one rotation. This precession is supposed to be very small, 43 arcseconds over a century. Since the solver is working so well, we now want to try to reproduce the observed perihelion precession of Mercury by adding a relativistic gravitational term, $\frac{GM_{Sun}M_{Mercury}}{r^2}\frac{3l^2}{r^2c^2}$, where $l = |\vec{r} \times \vec{v}|$, in Mercury's acceleration term. Over the period of a century, the position of Perhelion of Mercury is 43 arcsecconds.

We implement the relativistic term by creating a new class, "PlanetMercury", which inherits everything from the Planet class. Then we correct the acceleration function by overrideing it in the new class. We use only two objects for this simulation ,the sun and Mercury. Since the sun's mass is so much larger than the mass of Mercury, we solve this problem with a stationary sun. We saw that in the three body system, with larger planets than Mercury, the effect of introducing a moving sun was neglieble, so haivng a stationary sun in this scenario should not affect the results significantly.

We assume initial position of Mercury at perihelion, and localize perihelion at $(x, y) = (0.3075Au, 0)$. Initial velocity is set to 12.44 Au/Yr. When doing this, we get the following results for the perihelion precession for different Δt.

DO: Insert table with preihelion precession and Δt.
DO: Comment on the results in the table above.

4 Conclusions

In this project we solve problems for planetary motion, which mathematically are systems of 2nd order differential equations. To make the equations easier to work with, we scale the equations to astronomical units.

The systems of 2nd order equations are reformulated to systems of 1st order equations by use of variable substitution.

Solving the large system of equations in C++, we take apply classes and class inheritance.

Two numerical methods are used: the forward Euler method and the Velocity Verlet method. We show that the velocity Verlet method reproduces physical results like conservation of energy and angular momentum, while the Forward Euler method does not reproduce these results. The Velocity Verlet method converges faster to correct solutions, for preservation of energy and angular momentim, compared to the Forward Euler method. The velocity Verlet has twice the convergence rate of forward Euler when it comes to sup-norm in energy and angular momentum preservation, the velocity Verlet method being of order two and four for these quantitites respectively. Both methods gives bad results for the largest time steps. The methods are shown to involve approximatly the same amount of FLOPS.

The Velocity Verlet method reproduces the exact terminal velocity of a two object sun-earth system. Exact results are reproduced by Velocity Verlet also in scenarios where the gravitational force is changed.

Adding a third planet, Jupiter, to the stationary Sun earth system, gives reasonable results using Velocity Verlet: The orbits stay circular and stable. Also here the largest time steps give strange results, while the finer time steps gives convergent solutions. Making Jupiter's mass equal to the Sun's mass makes the earth escape. More realistic scenarios, that includes movement in the sun, keeping the center of mass of the system constant, produces reasonable results: When Jupiter's mass is enlarged, the sun starts to move. The center of mass is shown to be fixed.

The full solar system is simulated with the Velocity Verlet solver and with movement in the sun. The result seems reasonable, all orbits being stable.

By adding a relativistic correction to the gravitational force from sun on Mercury, the perihelion precession of Mercury around the sun is reproduced.

For future reference, we add the we could have improved the speed of the solvers by taking into account that the gravitational force between two planets are equal in magnitude. This implies that we could have halved the number of FLOPS!

5 Feedback

5.1 Project 1

This project has been extremely educational. We learned about about c++, especially pointers and dynamic memory allocoation. Also which for us was a well forgotten subject, we learned about dangerous of numerical round-off errors.

We feel the size of the project is large, much larger than typical assignments in other courses. However, the quality and quantity of the teaching without a doubt made the workload managable. The detailed lectures, combined with the fast and good respones on Piazza helped a lot!

We think the project could have gone even smoother, if we on the 2nd lab-session had learned basic branching in Github. We used a considerable amount of time finding out of this.

All in all, two thumbs up!

5.2 Project 2

- catch: We ended up using a lot of time making this work properly. Still we have some problems with catch and Qt. We think we might had benefited from a demonstration at the lab.
- We were not able to understand the revised Sturm-Bisection algorithm from Barth et al.'s [1] paper on the revised Sturm-Bisection.
- Apart from the small details above, we are very happy about this project. How would have thought linear algebra could be fun?!

5.3 Project 3

- Classes: Very useful!

6 Bibliography

[1] Barth, Martin, Wilkinson (1967) Calculation of eigenvalues of a symmetric tridiagonal matrix by the method of bisection. *Numeriche mathematik 9, 386 - 393 (1967)*

[2] Hjorth-Jensen, M.(2015) Computational physics. Lectures fall 2015. `https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Lectures`

[3] Hjorth-Jensen, M.(2017) Project 2, fys4150 2017. `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Projects/2017/Project2/pdf/Project2.pdf`

[4] Kiusalaas, J.(2013) Numerical Methods in Engineering with Python 3. 3rd edition.

[5] Taut, M. (1993) Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem *Phys. Rev. A 48, 3561 (1993).*