Simulation of the Solar System

Massimo Giordano

February 1, 2015

Link to the repository - code of the program
https : //github.com/massimogiordano/repository

Abstract

The following article describes a way to simulate the solar system with consideration of the occurring many body problem. The physical theory is based on Newton's second law of motion, which gets discretized into certain time steps. A solution is then obtained by either making use of the Runge-Kutta-Four- or the Verlet-Algorithm. Furthermore, both algorithms are compared of stability.

Contents

1 Introduction

Differential equations build the basis of describing a physical system. There is for example the radioactive decay[1], the time-dependent Schrödinger equation[2], the Poisson-equation[3] or Newton's second law of motion[4], just to mention a few. As systems become complexer and differential equations couple, we often do not find a closed analytical solution anymore. We now consider such a case, namely the latter equation of N bodys, what is called a N-Body-Problem. We discretize the set of differential equations and solve them analytically by either using the Runge-Kutta-four (RK4) or the Verlet-Algorithm (VA). To start with, we first only describe the system Sun-Earth to check our algorithms and simulation for stability and then expand the problem to N bodys.

2 Theoretical Concept

2.1 Sun-Earth-System

As mentioned in the introduction, the first part of the work focuses on testing the Runge-Kutta-Four- and the Verlet-Algorithm, before the problem is expanded to the solar system in the second part.
To start therefore, we first consider a system of Sun and one planet, say Earth. The acting force between both bodys is given by the gravitational force F_G

$$F_G = \frac{G M_{\mathrm{Sun}} M_{\mathrm{Earth}}}{r^2}, \tag{1}$$

where M_{sun} is the mass of the Sun, M_{Earth} is the mass of Earth, G is the Gravitational Constant and r the distance between Sun and Earth. As we want to test different solvers for ordinary differential equations, we assume a much larger mass of the Sun than the mass of the Earth. Consequently, we can neglect the movement of the Sun. Furthermore, there exists no tangential force. For this reason, the movement takes place in a plane, namely the xy-plane. In this case, Newton's second law of motion reads

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_{\mathrm{Earth}}}, \tag{2a}$$

and

$$\frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_{\mathrm{Earth}}}, \tag{2b}$$

where $F_{G,x}$ and $F_{G,y}$ are the x and y components of the gravitational force.
These two second order differential equations can be rewritten as a set of four coupled first order differential equations. We define a velocity $v(t) = dx/dt$ and get

$$\frac{dx(t)}{dt} = v_x \tag{3}$$

$$\frac{v_x(t)}{dt} = \frac{F_{\mathrm{G,x}}}{M_{\mathrm{Earth}}} \tag{4}$$

$$\frac{dy(t)}{dt} = v_y \tag{5}$$

$$\frac{v_y(t)}{dt} = \frac{F_{\mathrm{G,y}}}{M_{\mathrm{Earth}}} \tag{6}$$

These first order differential equations are used in the Runge-Kutta-Algorithm, but not in the Verlet-Algorithm, as the latter just calculates with accelerations.

[1] $-dN/dt = \lambda N$

[2] $i\hbar \frac{\partial \vec{\Psi}(\vec{x},t)}{\partial t} = \frac{\hbar^2}{2m}\left(\frac{\partial^2\vec{\Psi}(\vec{r},t)}{\partial x^2} + \frac{\partial^2\vec{\Psi}(\vec{r},t)}{\partial y^2} + \frac{\partial^2\vec{\Psi}(\vec{r},t)}{\partial z^2}\right) + V(\vec{x})\vec{\Psi}(\vec{x},t)$

[3] $-\Delta u = f$

[4] $m\frac{d^2x}{dt^2} = F(x)$

The x and y component of the gravitational force can easily be derived by the theorem of intersecting lines. It holds

$$\frac{r}{x} = \frac{F_G}{F_{G,x}} \tag{7a}$$

$$\frac{r}{y} = \frac{F_G}{F_{G,y}} \tag{7b}$$

and therefore

$$F_{G,x} = \frac{x}{r} F_G \tag{8a}$$

$$F_{G,y} = \frac{y}{r} F_G \tag{8b}$$

with F being the absolute value of the gravitational force.
We can also estimate the Gravitational Constant by assuming a cicular orbit of the Earth around the Sun. This estimate sounds quite reasonable as the acutal orbit's eccentricity is about 0.017, namely almost a cicular orbit. In this case, the gravitational force is balanced by the centripetal force. Therefore, the absolute values must be equal

$$F_{\mathrm{ZP}} = \frac{M_{\mathrm{Earth}} v^2}{r} = \frac{G M_{\mathrm{Sun}} M_{\mathrm{Earth}}}{r^2} = F_G, \tag{9}$$

with v being the orbital velocity of Earth.

2.2 Two-Body-Problem

In the previous considerations, we assumed a much larger mass of the Sun than the mass of the Earth. For that reason, we could neglect the motion of the Sun and easily derive relations. We now do consider Earth's mass and solve the occurring two body problem.
To start with, we let x_1, m_1 and x_2, m_2 be the position vectors and masses of body 1 and body 2. We can easily calculate the acting gravitational forces

$$\vec{F}_{2,1}(\vec{x}_1, \vec{x}_2) = -G m_1 m_2 \frac{\vec{x}_1 - \vec{x}_2}{|\vec{x}_1 - \vec{x}_2|^3} \tag{10a}$$

and

$$\vec{F}_{1,2}(\vec{x}_1, \vec{x}_2) = -G m_1 m_2 \frac{\vec{x}_2 - \vec{x}_1}{|\vec{x}_1 - \vec{x}_2|^3} \tag{10b}$$

what leads to

$$\frac{\partial^2 \vec{x}_1}{\partial t^2} = -G m_2 \frac{\vec{x}_1 - \vec{x}_2}{|\vec{x}_1 - \vec{x}_2|^3} \tag{11a}$$

and

$$\frac{\partial^2 \vec{x}_2}{\partial t^2} = -G m_1 \frac{\vec{x}_2 - \vec{x}_1}{|\vec{x}_1 - \vec{x}_2|^3}. \tag{11b}$$

We introduce new vectors for the relative coordinate $\vec{r} := \vec{x}_1 - \vec{x}_2$ and the Center of Mass (COM) vector $\vec{R} := (m_1\vec{x}_1 + m_2\vec{x}_2)/(m_1 + m_2)$ and call $M = m_1 + m_2$. We plug equations 11a and 11b into the definition of $\vec{r}$ as well as into the definition of $\vec{R}$ and obtain

$$\frac{\partial^2 \vec{R}}{\partial t^2} = 0 \tag{12}$$

and

$$\frac{\partial^2 \vec{r}}{\partial t^2} = -GM \frac{\vec{r}}{r^3} \tag{13}$$

Equations 12 and 13 are used in section 2.3 to demonstrate conservation of energy and angular momentum.

2.3 Conservation of Angular Momentum and Energy

Using equation 13, we can derive a relation between the angular momentum and the force, which is acting on the planet

$$\frac{d\vec{L}}{dt} = \vec{T} = \vec{F}_{1,2} \times \vec{r} = -\frac{GMm_1}{r^3}\vec{r} \times \vec{r} = 0 \tag{14}$$

for a two body system. As its derivative is zero, the angular momentum does not change in time. The demonstration of conservation of energy is borrowed from Krug (Köln, 2009). The total energy E of a system of N masses is given as the sum of the potential energy V and the kinetic energy T of all masses

$$E = T + V = \sum_{i=1}^{N} \frac{1}{2} m_i \left(\frac{d\vec{r}}{dt}\right)^2 + \sum_{i,j;i<j} V_{i,j}(\vec{r}_i, \vec{r}_j) + V^{ext}(\vec{r}_1, ...\vec{r}_N). \tag{15}$$

To show conservation of energy, we consider the change of energy in time

$$\frac{dE}{dt} = \frac{dT}{dt} + \frac{dV}{dt}. \tag{16}$$

First we calculate the change in kinetic energy

$$\begin{aligned}\frac{dT}{dt} = \sum_{i=1}^{N} m_i \dot{\vec{r}}_i \ddot{\vec{r}}_i = \sum_{i,j;i\neq j} \dot{\vec{r}}_i \vec{F}_{i,j} + \sum_{i=1}^{N} \dot{\vec{r}}_i \vec{F}_i &= \sum_{i,j;i<y} (\dot{\vec{r}}_i \vec{F}_{ij} + \dot{\vec{r}}_j \vec{F}_{ji}) + \sum_{i=1}^{N} \dot{\vec{r}}_i \vec{F}_i \\ &= \sum_{i,j;i<j} (\dot{\vec{r}_i} - \dot{\vec{r}_j}) \vec{F}_{ij} + \sum_{i=1}^{N} \dot{\vec{r}}_i \vec{F}.\end{aligned} \tag{17}$$

After that we calculate the change in potential energy

$$\begin{aligned}\frac{dV}{dt} = \sum_{i,j;i<j} (\dot{\vec{r}}_i \nabla_i V_{ij} + \dot{\vec{r}}_j \nabla_j V_{ij}) &+ \sum_{i=1}^{N} \dot{\vec{r}}_i \nabla_i V^{ext}(\vec{r}_1, ..., \vec{r}_N) \\ = -\sum_{i,j;i<j} (\dot{\vec{r}}_i - \dot{\vec{r}}_j) \vec{F}_{ij} &- \sum_{i=1}^{N} \dot{\vec{r}}_i \vec{F}_i\end{aligned} \tag{18}$$

It turns out, that

$$\frac{dT}{dt} = -\frac{dV}{dt} \tag{19}$$

and

$$\frac{dE}{dt} = \frac{dT}{dt} + \frac{dV}{dt} = 0. \tag{20}$$

Therefore, conservation of energy holds. The conservation of energy is tested in the simulation.

2.4 N-Body-Problem

If we expand our system to the solar system with consideration of the Sun and all nine planets, we are dealing with a ten body problem. Note that we count Pluto as a planet for historical reasons. For such a "complex" system, the by now only well known way to get quantitative results is a numerical approach. To start with, we calculate for every timestep for every body all forces, which are acting on it. We get:

$$\frac{d\vec{v}_j(t)}{dt} m_j = \vec{F}_j(t) = Gm_j \sum_{i\neq j} \left[\frac{m_i}{|\vec{r}_i(t) - \vec{r}_j(t)|^2} \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|} \right] \tag{21}$$

To solve equation 21, we integrate, discretize and finally obtain

$$\vec{v}_{n+1} = \vec{v}_n + G \int_{t_n}^{t_{n+1}} \sum_{i \neq j} \left[m_i \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i(t) - \vec{r}_j(t)|^3} \right] dt. \tag{22}$$

Note that the index i stands for the i-th body whereas the index n stands for discretized timesteps. Equation 22 will be solved in section 3.

3 Method

To solve a generic problem of n couple of differential equation we can perfome a so-called *Rugge Kutta* or $Verlet$Algorithm. These two algorithms use a different approach to solve the problem. We start to discuss the Verlet Algortihm that we use.

We can perfome a Taylor expantion of the unknown position at time $t + \Delta t$.

$$p_n(t + \Delta t) = p_n(t) + v_n(t)\Delta t + \frac{1}{2} a_n(t)\Delta t^2 + O(\Delta t^3) \tag{23}$$

Since the formulas that we use to describe the solar system let us to compute only the acceleration we have stop the Tylor's expantion at the third degree. Again with the Taylor expansion we can describe the velocity at $t + \frac{\Delta t}{2}$

$$v_n(t + \frac{\Delta t}{2}) = v_n(t) + t + \frac{a_n(t)\Delta t}{2} \tag{24}$$

and the acceleration at $t + \Delta t$:

$$a_n(t + \Delta t) = \sum_{1}^{all\ planets} - \frac{G\ M_i}{(r_i(t + \Delta t))^2} \tag{25}$$

Since the velocity follows this equation $v(t + \Delta t) = v(t) + a(t)\Delta t$ we can write that:

$$v_n(t + \Delta t) = v_n(t + \frac{\Delta t}{2}) + \frac{1}{2} a_n(t + \Delta t)\Delta t; \tag{26}$$

The Verlet algorithm provides a good preservation of the sympletic form on phase space and it doesn't require a big computational cost. This means it preserves the kinetic energy for a huge period of time.

Due to the fact that we slip the second order differential equation in two first order differential equation, the Rugge Kutta method only uses first derivates.

Before starting to explain this algorithm we make some observations:

$$\frac{dp(t)}{dt} = v(t) \quad and \quad \frac{dv(t)}{dt} = a(t) \tag{27}$$

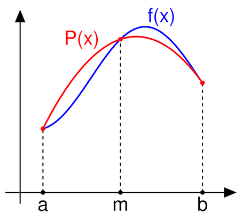
We discuss the case for solving the velocity knowing that it's the same for the position.

$$y(t) = \int a(t)dt \quad and \quad y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} a(t)dt \tag{28}$$

If we interpret the integral as the area under the curve we have that:

$$\int_{t_i}^{t_{i+1}} a(t)dt \simeq \Delta t\ a(t_{i+\frac{1}{2}}) \tag{29}$$

Using the Simpson's formula we can approximate an integral by a finite polynomial in this way:



$$\int_{t_i}^{t_{i+1}} a(t)dt \simeq \frac{\Delta t}{6}\left[a(t_i) + 4a(t_{i+\frac{1}{2}}) + a(t_{i+1})\right] \tag{30}$$

To have a higher precision we can estimate the middle point more often in this way:

$$k1 = a(t_i) \quad \rightarrow \quad p_{i+\frac{1}{2}} = p_i + \frac{\Delta t}{2}\ k1 \ \ \text{first approximation of middle point} \tag{31}$$

$$k2 = a(t_{i+\frac{1}{2}}) \quad \rightarrow \quad p_{i+\frac{1}{2}} = p_i + \frac{\Delta t}{2}\ k2 \ \ \text{second approximation of middle point} \tag{32}$$

$$k3 = a(t_{i+\frac{1}{2}}) \quad \rightarrow \quad p_{i+1} = p_i + \Delta t\ k3 \ \ \text{first approximation of point} \ \ i+1 \tag{33}$$

$$k4 = a(t_{i+1}) \tag{34}$$

Note that $K4$ can be founding using the values of the position computed in the previus step 33.
Since this algorithm doesn't provide a preservation of the symplectic form on phase space we expect that it could lose kinectic energiy after a long period of time. Instead, since the degree of local error is higher we expect that this algorithm is more precise for a short period of time.

4 Tuned Program

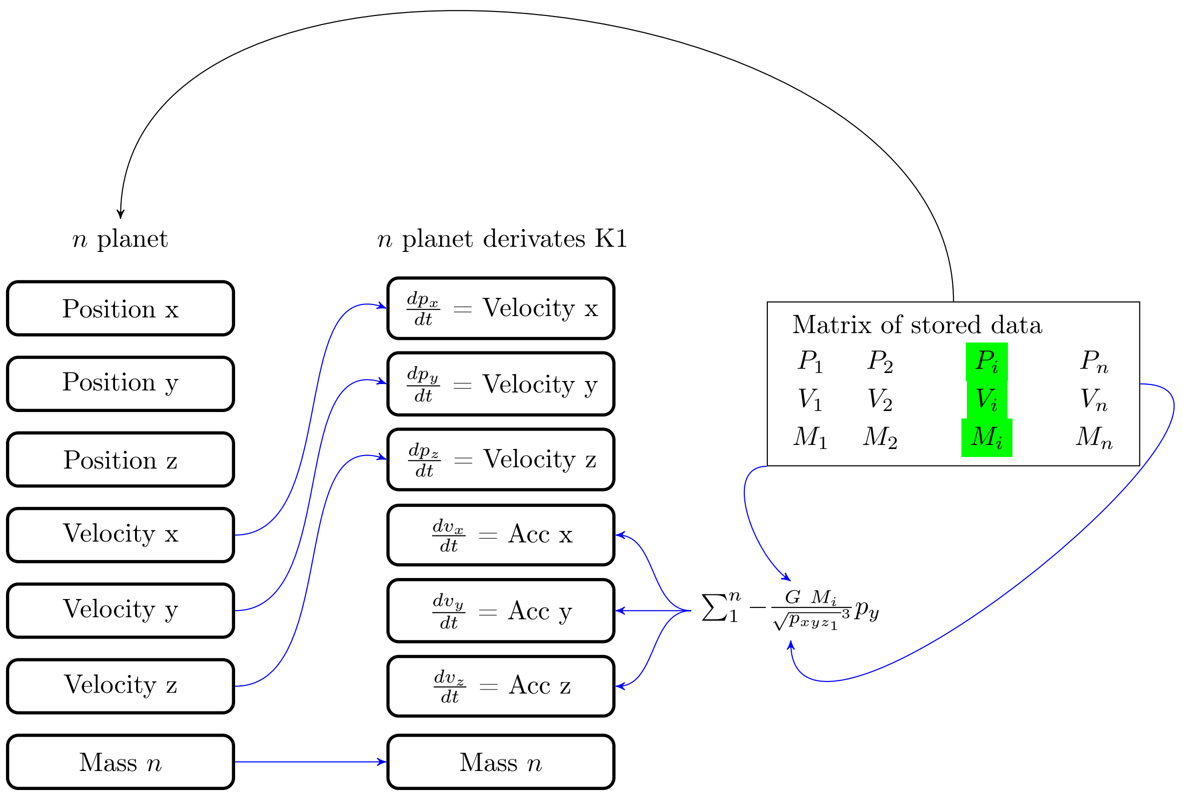
Writing the program we focus our attention to solve a $n - bodies$ problem. To do that we had to find the most general and convenient way. At first we devide the program in classes, we make a class for the planets where we included also the sun; we consider it as a planet with bigger mass.

1. classes

```
class planet
{
public:

    double position[3];
    double velocity[3];
    double mass;

    planet(double mas, double x,double y, double z, double vx, double vy, double vz);

};
class solarsystem
{
public:
void solarsystem::add(planet n){
     number_planets++;
     all_planets.push_back(n);
 };
 }
 main(){
       int RK4 = 1; //set zero for not use this method
       solarsystem mysystem;

       planet sun( 1, 0, 0, 0,0,0, 0 );
       planet earth(3e-6, 1, 0 ,0, 0, 2*M_PI ,0);
       mysystem.add(sun);
       mysystem.add(earth);

29      if(RK4) {
30              mysystem.solverRK4(mysystem.all_planets, 0.01, 4 );
31     }else{
32              mysystem.solverVERLET(mysystem.all_planets, 0.01, 4 );
33   }
34 }
35 }
```

The line 1.9 shows us how to insert data when we define a new planet as in line $1.23-24$. The function in 1.17 inserts an object "planet" in a vector in the "solarsystem" class that will contain all the planets. In line $1.29-31$ we call the function "solver". It is also possible to decide which algorithm shall be used to solve the problem and set up the value of Δt and the number of years for running the simulation.

We decide to store the data of position, velocity and mass in a matrix of dimension $[n, 7]$ where n is the number of planets that we compute in the simulation. Using the structure as it is shown below we can easily manage these data.


n planet
n planet derivates K1
Position x
Position y
Position z
Velocity x
Velocity y
Velocity z
Mass n
$\frac{dp_x}{dt}$ = Velocity x
$\frac{dp_y}{dt}$ = Velocity y
$\frac{dp_z}{dt}$ = Velocity z
$\frac{dv_x}{dt}$ = Acc x
$\frac{dv_y}{dt}$ = Acc y
$\frac{dv_z}{dt}$ = Acc z
Mass n
Matrix of stored data
P_1 P_2 P_i P_n
V_1 V_2 V_i V_n
M_1 M_2 M_i M_n
$\sum_1^n -\frac{G\ M_i}{\sqrt{p_{xyz_1}}^3} p_y$

Structure, storage and migration of data.

We now show how we compute the derivates. As we can see in the picture above for the derivates of the position, we just take the data of the velocity and we compute the accceleration considering the interaction with all planets. See code 3 and line $2.8to18$.

2. derivates function

```
1 void solarsystem::derivate(mat &dat, mat &de, int n){
2
3    double accelleration_x=0,accelleration_y=0,accelleration_z=0, mod_force;
4    for(int i=0; i<n; i++){
5        accelleration_x=0,accelleration_y=0,accelleration_z=0;
```

```
        for(int j=0; j<n; j++){ // this loop compute the force with each planet
            if(i!=j){ // this avoid to compute the force with itself
                mod_force = //since the force is infinit
                force(dat(j,0)-dat(i,0),dat(j,1)-dat(i,1), dat(j,2)-dat(i,2), dat(j,6));

                accelleration_x += mod_force*(dat(j,0)-dat(i,0));
                accelleration_y += mod_force*(dat(j,1)-dat(i,1));
                accelleration_z += mod_force*(dat(j,2)-dat(i,2));
          }
          }
          de(i,3) = accelleration_x;
          de(i,4) = accelleration_y;
          de(i,5) = accelleration_z;
          }
          for(int i=0; i<n; i++){
                de(i,0) = dat(i,3); //velx
                de(i,1) = dat(i,4); //vely
                de(i,2) = dat(i,5); //velz
         }
}
```

3. function that compute forces and Function that sum matrix

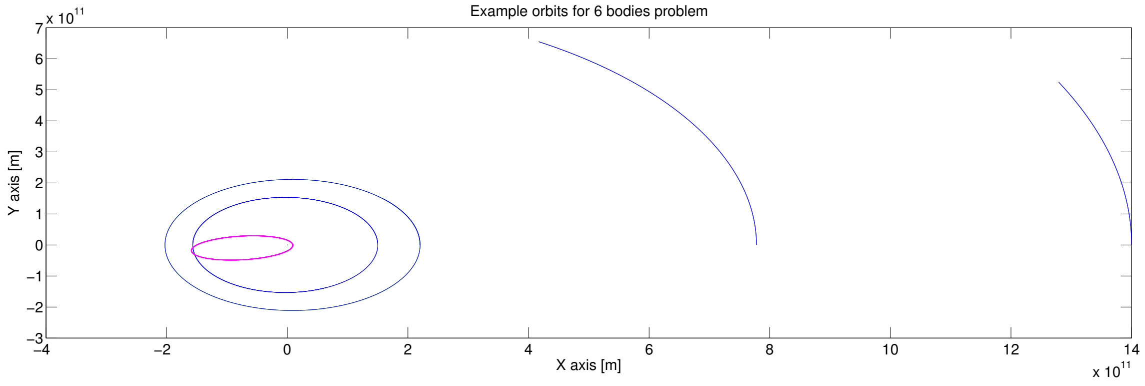
```
}
double solarsystem::force(double x, double y, double z, double Mothers){
    double G= 4*M_PI*M_PI;
    double force=0;
    double distance=0;

    distance = x*x + y*y + z*z;

    force = G*Mothers/pow(distance, 1.5);

    return force;
}
void solarsystem::sum_matrix(mat &result, double coeff_one, mat &first,
                                          double coeff_two, mat &second, int n){
    for(int j=0; j<n; j++){
        for(int i=0; i<6; i++){
          result(j,i) = coeff_one*first(j,i) + coeff_two*second(j,i);
        }
        result(j,6) = first(j,6); //report mass
    }
}
```

We now show how we have implemented the RK4 algorithm to solve the solar-system simulation.

4. Rugge Kutta sover

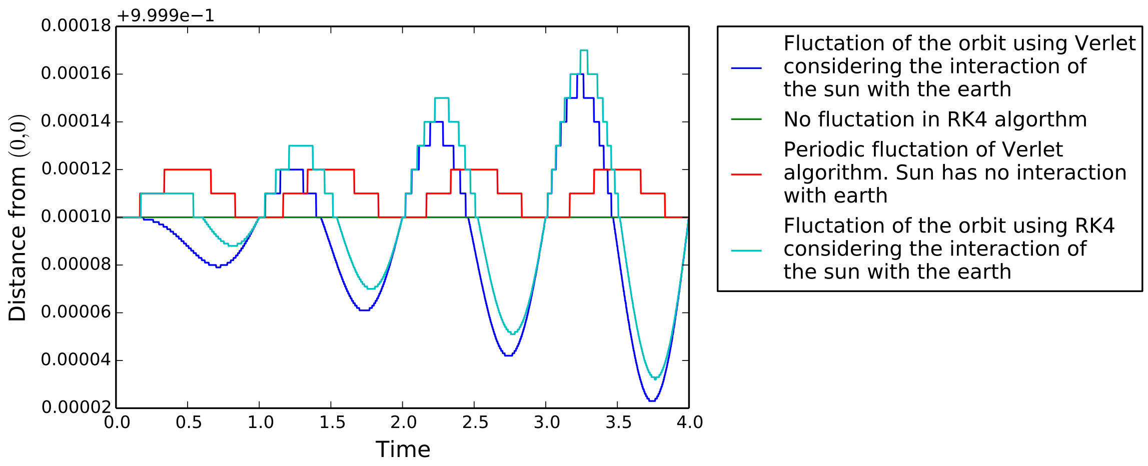
```
void solarsystem::solverRK4(vector<planet> vec, double h, double tmax){

    mat y_i(number_planets,7);
    mat y_i_temp(number_planets,7);
    mat k1(number_planets,7);
    mat k2(number_planets,7);
    mat k3(number_planets,7);
    mat k4(number_planets,7);

    insert_data(vec , y_i);
    double t=0;

    while(t<tmax){

        derivate(y_i, k1, number_planets);

        sum_matrix(y_i_temp, 1, y_i, 0.5*h, k1, number_planets);

        derivate(y_i_temp, k2, number_planets);

        sum_matrix( y_i_temp, 1, y_i, 0.5*h, k2, number_planets);

        derivate( y_i_temp, k3, number_planets);

        sum_matrix( y_i_temp, 1, y_i, h, k3, number_planets);

        derivate( y_i_temp, k4, number_planets);

        for(int j=0; j<number_planets; j++){
             for(int i=0; i<6; i++){
                 y_i(j,i) = y_i(j,i) + h*(k1(j,i) + 2*k2(j,i) + 2*k3(j,i) + k4(j,i))/6;
             }
             //syncroniz position and velocity.
             planet &questo = vec[j];
             for(int i=0; i<3; i++){
             questo.position[i] = y_i(j,i);
             questo.velocity[i] = y_i(j,i+3);
             }
        }
 t+=h;
}
}
```

Now we show an example of n-body problem where the algorithm can find the orbit of each planet with good precision. We notice that since the sun has a mass 10^6 times bigger than others it almost does not move.



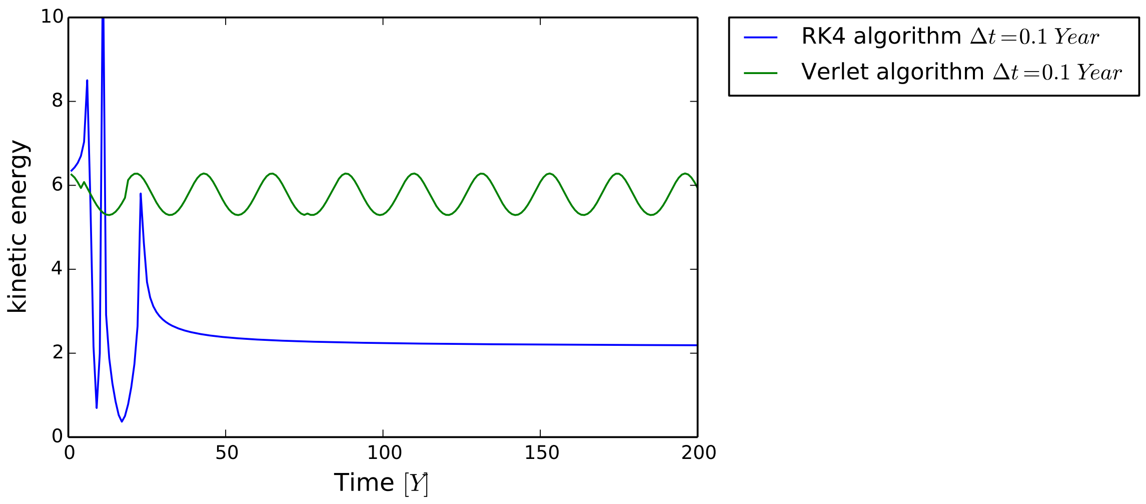
If we want to find a orbit that is exactly circular we have to set a velocity so that the centripetal acceleration is equal to the gravitational force. If we use astronomic units it is easy to find that this velocity for the earth is 2π. Setting this velocity we know that the earth should have an exactly circular orbit. Knowing that we can compare the accuracy of the two algorithm.

In the graphic below we see that, if we don't consider the force due to the mass of earth on the sun, with the $RK4$ method the distance between the two planets is exacly constant. If we see the result of $Verlet$ algorithm we see that it fluctuates around a value that remains constant in time. So we can say that in this case the $RK4$ method is more precise. See $[Figure\ 1]$



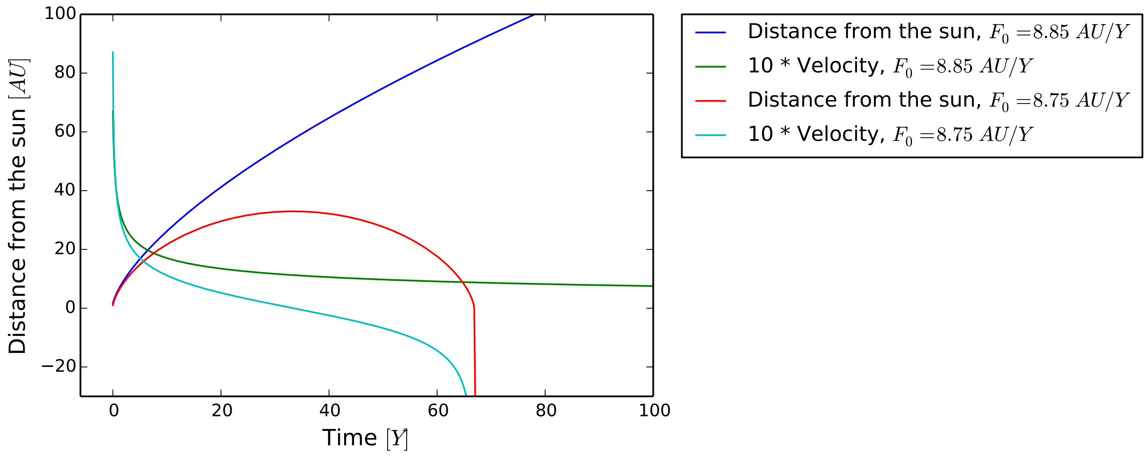
[Figure 3.1] Comparison of the conservation of kinetic energy using different algorithms

But if we try to compute the kinetic energy with the same parameters for a huge period of time we see that the $Verlet$ method is better. In fact it doesn't lose energy also for $10^4 years$, whereas we see how the RK4 method already loses energy after some years. See $[Figure\ 2]$



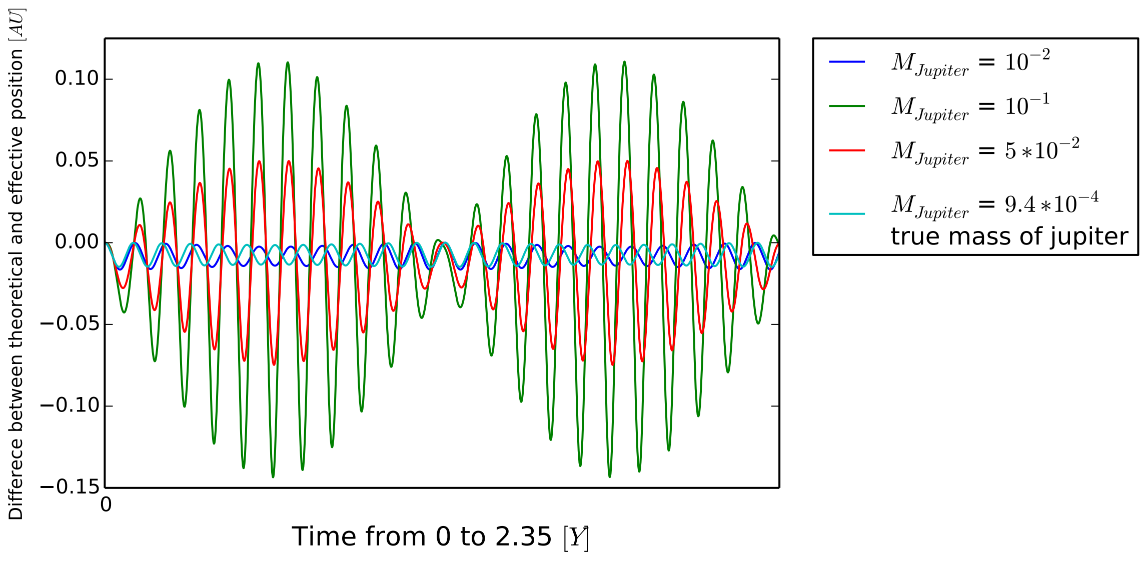
[Figure 3.2]comparison of the conservation of kinetic energy using different algorithms

Now we want to compare the theorical escape velocity with the one computed by our algorithm. For this we use the $RK4$ algorithm. For the Sun and at the distance of 1 AU the escape velocity is $V_e = 8.8858\frac{AU}{Y}$. If we set the velocity of the earth $V_x = V_e$ $(Theorical)$ and if we run the simulation with $\Delta t = 0.001$ for 3×10^4 years we see that the velocity is always positive even if it decreases continuously. But if we set $V_x = 8.880$ the velocity becomes negative after some years. In the graphic below we plot the distance from the sun and the velocity for two different values of initial velocity. See $[Figure3]$. We can say that our algorithm perfomes very well in this problem.



[Figure 3.3] comparison between two different velocity to prove the validity of theorical escape velocity

Now we want see how a planet with a big mass can influence the orbit of the earth. For that reason we start to solve a $3-bodies$ problem considering also the $Jupiter$ planet. Infact this planet is the one with the greater mass after the sun. We set the velocity of the earth exactly 2π in order to have an orbit that is exactly a circle to use as a reference. Note that if there were no Jupiter the following graphic would be exacly flat. We also observe that if the mass of Jupiter is comparable to the one of the sun, the system becomes unstable since the Sun and Jupiter attract each other and the earth moves from one orbit to the another.



[Figure 3.4] This graphic show how the orbit of the earth changes under the influence of different masses at a distance of $\sim 5AU$

4.1 The entire solar system

To start a simulation, we need to set initial values. If we, for example, did not set the velocities to a value, which is not equal zero, all planets would just fall into the sun. This case is not of physical importance. Either, the velocity of a planet is too slow, then the planet spirals into the Sun, or the velocity is too large, then the planet spirals away from Sun and escapes, or the velocity is in a certain range, where we have a bounded orbit. We express distances in Astronomical Units (AU), which is the average distance between Earth and Sun, all masses in Solar Masses and all velocities in Astronomical Units per year. To compare with SI-Units, we set

Comparison to SI-Units

Astr.	SI
AU	$1.5e11$
M_s	$2e30$
AU/y	k

In these units, the initial values of the bodys read

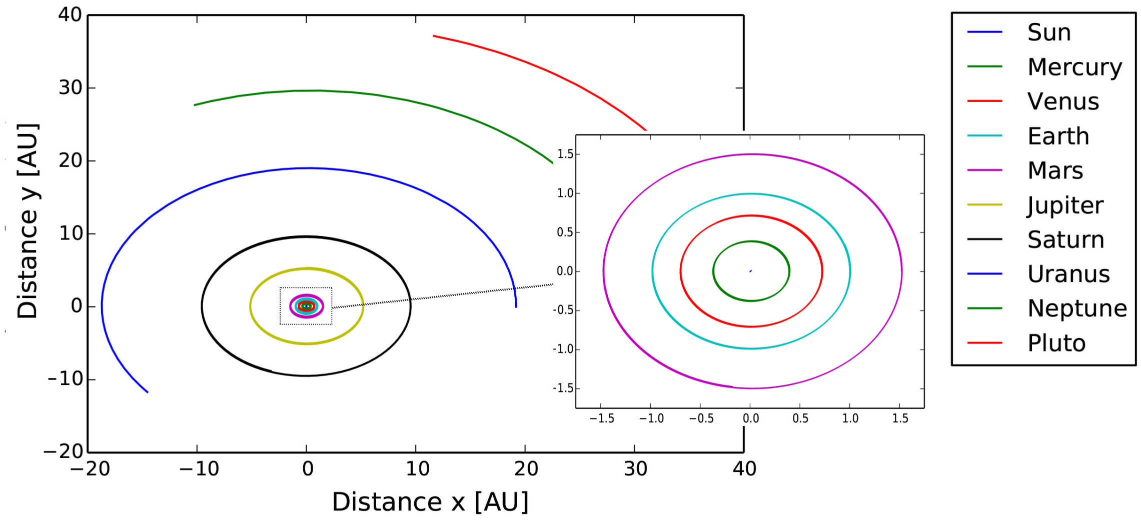
Initial Values of Bodys

Name	Mass	Position x	Position y	Position z	Velocity x	Velocity y	Velocity z
Sun	1	0	0	0	0	0	0
Mercury	1.2e-7	0.39	0	0	0	9.96	0
Venus	2.4e-6	0.72	0	0	0	7.36	0
Earth	1.5e-6	1	0	0	0	6.26	0
Mars	3.3e-7	1.52	0	0	0	5.06	0
Jupiter	9.5e-4	5.20	0	0	0	2.75	0
Saturn	2.75e-4	9.54	0	0	0	2.04	0
Uranus	4.4e-5	19.19	0	0	0	1.43	0
Neptune	5.1e-5	30.06	0	0	0	1.14	
Pluto	5.6e-9	39.53	0	0	0	0.99	0

To put it into machine code, we write that in the form Name(mass, px, py, pz, vx , vy, vz)

Machine code for Initial Values of Bodys

```
planet Sun(1,0,0,0,0,0,0);
planet Mercury(1.2e-7, 0.39, 0, 0,0,9.96,0);
planet Venus(2.4e-6, 0.72, 0, 0,0,7.36,0);
planet Earth(1.5e-6,1,0,0, 0, 6.26, 0);
planet Mars(3.3e-7, 1.52, 0, 0,0,5.06,0) ;
planet Jupiter(9.5e-4, 5.20, 0,0,0,2.75,0) ;
planet Saturn(2.75e-4, 9.54, 0, 0,0,2.04,0) ;
planet Uranus(4.4e-5, 19.19, 0, 0,0,1.43,0);
planet Neptune(5.1e-5, 30.06, 0, 0,0,1.14,0);
planet Pluto(5.6e-9, 39.53, 0, 0,0,0.99,0);
```



Simulation of the entire solar system runned for 50 year

5 Conculsions

The article simulated the solar system with consideration of the occurring N-Body-Problem. After discretizing the set of differential equations, the main algorithms were the Runge-Kutta-Four and the Verlet-Algorithms. We implemented both, to compare them in questions of total numbers of timesteps. We figured out, that first Runge-Kutta produces a smaller oscillation in total energy than Verlet, which is oscillating from beginning on. After a point, somehow, the total energy calculated by Runge-Kutta blows up, whereas the amplitute of oscillation in total energy in Verlet stays bounded. This curious case can be explained by the socalled preservation of the symplectic form in phase space, which is not covered in this article. Still, the errors are quite small. To estimate them, we simulated one orbit of the Earth around the Sun and got an error of the order of 10^{-4}. For this reason, both algorithms can be used to calculate solutions for the problem, but one needs to carefully consider the total time for the simulation and the timesteps to avoid parts, where the preservation of the symplectic form in phase space holds.

6 References

- M. Hjorth-Jensen, Computational Physics, University of Oslo (2013).
- M. Hjorth-Jensen, Project3, University of Oslo (2013).
- Symplectic Integrator, from Wikipedia.
- J. Krug, Theoretische Physik in zwei Semestern I: Teil A: Klassische Mechanik, University of Köln, retrieved October 18, 2014, from http://www.thp.uni-koeln.de/ miv/SS09/Mechanik.pdf.