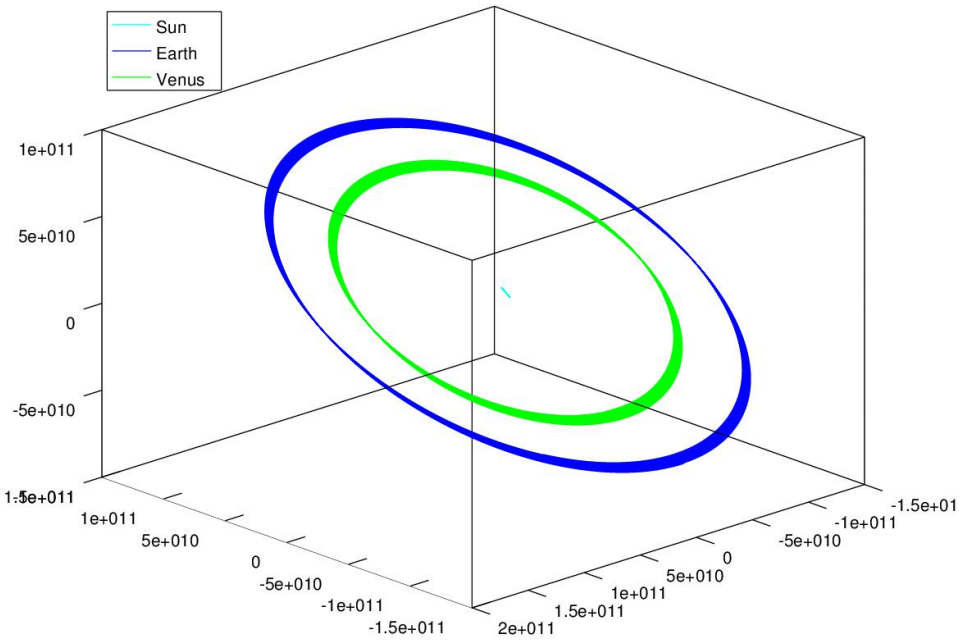
Modeling planetary motion
- a brief introduction to Keplers problem -

Arsen Kadyrov and Tanja Neder

Würzburg, August 2019



supervision: Prof. Dr. Alfio Borzì

Institute for Mathematics

Julius-Maximilians-University Würzburg

Contents

Abstract

The Kepler problem, named after Johannes Kepler, is often used in combination with planetary motion. In the classical sense it describes the motion of two particles whose orbits are affected through the gravitational forces they exert on each other.

In this paper we discuss the derivation of the reduced one-body problem from the two-body problem with a simple implementation. Then we will add a correction term introduced by Albert Einstein, Leopold Infeld and Banesh Hoffmann. Besides we will apply a second approach, which is known as the Kozlov-method. Afterwards we will proceed with the two- and the three-body problem. In the end one will find a quick example of how to implement our complete solar system.

All the algorithms are implemented in Matlab and are attached in the appendix.

1.Reduced Keplers one-body problem

In order to understand planetary motion, one got to be aware of how the moving bodies affect each others trajectories. Beginning with two bodies, one can generalize the equations of interactions to an arbitrary number of bodies. Therefore it's just natural to start with two moving objects. As we will see, this will lead to a system of two ordinary differential equations which can be reduced to one equation by considering relative displacement and making the most of some physical laws. This will be subject of the following section.

1.1. From Keplers two-body problem towards Keplers one-body problem - an introduction

The derivation of the reduced Kepler one-body problem as given in the following can be found in [1].

Starting with the two-body problem, one gets by Newton's third law and the gravitational law for the force F_{12}, which body one exerts on body two,

$$F_{12} = -G \cdot \frac{m_1 \cdot m_2}{|y_1(x) - y_2(x)|^3} \cdot (y_1(x) - y_2(x)) \tag{1.1}$$

where $y_i(x)$ denotes the position of body i with mass m_i at time x and G is the gravitational constant. Additionally the named laws give us $F_{12} = -F_{21}$. By Newton's second law we obtain the ordinary differential equations, we are interested in. That is

$$\begin{aligned} F_{12} &= m_1 \cdot y_1''(x) \\ F_{21} &= m_2 \cdot y_2''(x) \end{aligned} \tag{1.2}$$

For the reduction we introduce the relative displacement $q(x) = y_1(x) - y_2(x)$. With (1.2) it follows

$$q''(x) = \frac{F_{12}}{m_1} - \frac{F_{21}}{m_2} = \frac{m_1 + m_2}{m_1 \cdot m_2} \cdot F_{12} \tag{1.3}$$

Keep in mind that F_{12} is also a function in $q(x)$ given by (1.1). Therefore we get

$$q''(x) = -\frac{K^2}{|q(x)|^3} \cdot q(x) \tag{1.4}$$

with $K^2 = G \cdot (m_1 + m_2)$. As for the reduced one body problem there aren't given the two masses, in practice it is often chosen $K^2 = 1$.

To numerically solve (1.4), consider the system of first order ordinary differential equations

$$\begin{aligned} q'(x) &= p(x) \\ p'(x) &= -\frac{K^2}{|q(x)|^3} \cdot q(x) \end{aligned} \tag{1.5}$$

By applying the midpoint rule one gets

$$\begin{aligned} q_{k+1} &= q_k + \frac{h}{2} \cdot (p_{k+1} + p_k) \\ p_{k+1} &= p_k - h \cdot K^2 \cdot \frac{4}{|q_{k+1} + q_k|^3} \cdot (q_{k+1} + q_k) \end{aligned}$$

where h denotes the step size of the midpoint scheme. The resulting implicit function system is in the presented algorithm, which can be found in appendix A, solved with the Newton-method.

To judge the quality of the presented algorithm, one can consider the preserved quantities, namely the Hamiltonian function, the angular momentum and the so called Runge-Lenz vector, which will be introduced in the following. The Hamiltonian function $H(q,p)$ which is characterized by

$$\begin{aligned} \frac{\partial}{\partial p} H(q,p) &= q' &&= p \\ \frac{\partial}{\partial q} H(q,p) &= -p' &&= \frac{K^2}{|q|^3} \cdot q \end{aligned}$$

is therefore given by

$$H(q,p) = \frac{1}{2} \cdot |p|^2 - \frac{K^2}{|q|} \tag{1.6}$$

For the angular momentum l it holds

$$l(p(x), q(x)) = q(x) \times p(x)$$

The last preserved quantity we want to consider, is the Runge-Lenz vector R, which is defined as

$$R(p(x), q(x)) = p(x) \times l(x) - \frac{K^2}{|q(x)|} \cdot q(x)$$

Geometrically this vector points towards the perihelion of the orbit.

The description of the preserved quantities is given in [1].

As these functions are constant, it's convenient to use them to examine the quality of your algorithm. The numerical errors with respect to this quantities are therefore given by

$$\begin{aligned} err_H(x) &= H(q(x), p(x)) - H(q_0, p_0) \\ err_l(x) &= \|l(p(x), q(x)) - l(p_0, q_0)\|_2 \\ err_R(x) &= \|R(p(x), q(x)) - R(p_0, q_0)\|_2 \end{aligned} \tag{1.7}$$

where p_0 and q_0 denote the initial values.

Again this intuition can be found in [1].

Applying this method to an example, given in [1], with initial values $q_0 = (1, 0.5, 0)$ and $p_0 = (0, 1, 0.5)$, step size $h = 0.01$ and number of iterations $N = 5000$,

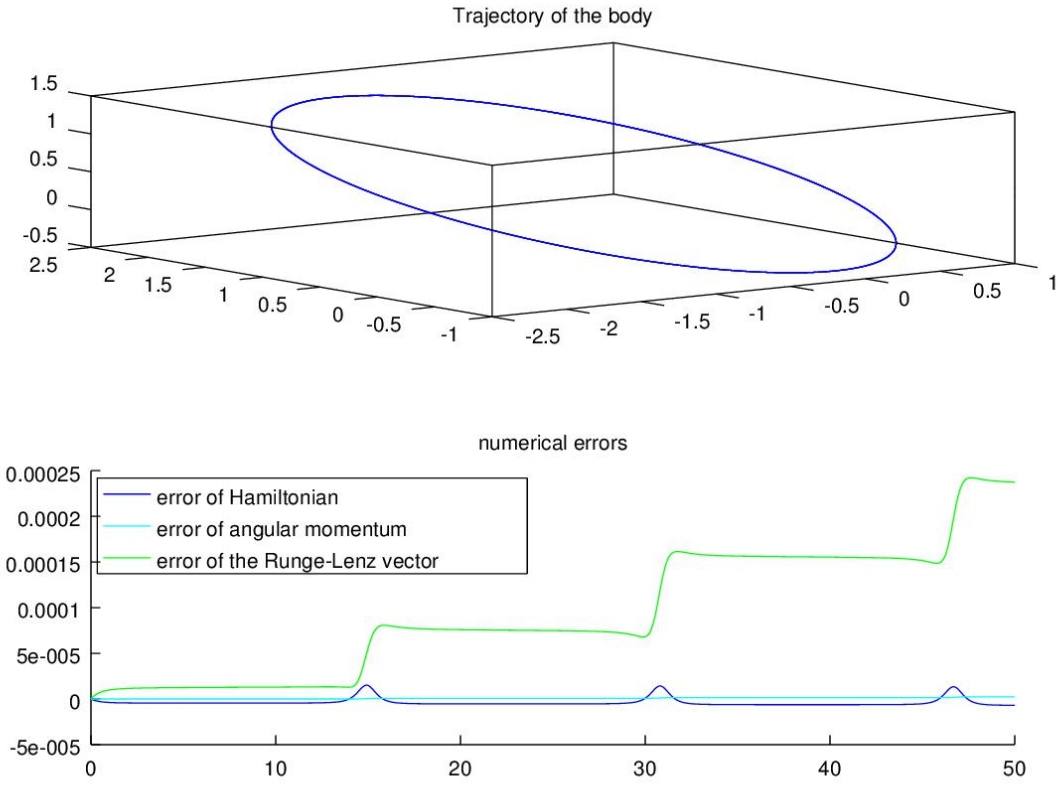


Figure 1.1: trajectory and numerical errors for the reduced one-body problem

one gets for the trajectory and the corresponding numerical errors the results shown in figure 1.1. One can see that the trajectory itself seems to be a perfect ellipsis reproducing the searched orbit. Only the diagram below in 1.1 reveals some numerical errors. While the angular momentum has apparently constant error which is about zero, one can recognize a change in error with every cycle in the other two quantities. Whereas the error of the Hamiltonian has a relatively small peak at every cycle, the error of the Runge-Lenz vector has a jump at every period. The latter being a sign of change in the inclination of the orbit, as the Runge-Lenz vector always points to the perihelion, like mentioned earlier that chapter. So as seen, this error can increase arbitrarily. To avoid this relatively big jump in the error, one can apply a correction term shown in the following chapter.

1.2. Relativistic EIH-correction

Considering the motion of two planets, Kepler referred to Newton's third law, which turned out to be non-correct regarding Einsteins special theory of relativity. Therefore Einstein together with Leopold Infeld and Banesh Hoffmann adapted Keplers formula

with some correction term, which leads to the second order ordinary differential equation

$$q''(x) = \frac{K^2}{|q(x)|^2} \cdot \left(-\frac{q(x)}{|q(x)|} + \frac{1}{c^2} \cdot A_1 \right)$$

where c is chosen such that $\epsilon = \left(\frac{|v|}{c}\right)^2$ is sufficiently small. Thereby v denotes the relative velocity of the two bodies, which we considered in the previous chapter for derivation of the reduced one-body problem. When putting $p(x) = q'(x)$ like before, it follows $v = p$. Furthermore the first post-Newtonian correction is given by

$$\begin{aligned} A_1 = \Bigg(& (4 + 2 \cdot \eta) \cdot \frac{K^2}{|q(x)|} - (1 + 3 \cdot \eta) \cdot |p(x)|^2 + \frac{3}{2} \cdot \eta \cdot \left(\frac{d|q(x)|}{dx} \right)^2 \Bigg) \cdot \frac{q(x)}{|q(x)|} \\ & + (4 - 2 \cdot \eta) \cdot \left(\frac{d|q(x)|}{dx} \right) \cdot p(x) \end{aligned}$$

where η denotes a mass proportion, $\eta = \frac{m_1 \cdot m_2}{(m_1 + m_2)^2}$ to be precise. In terms of the algorithm η will be a given constant.

This EIH-correction can be found in [1] with some further explanation.

For the numerical approach we proceed like in the previous chapter, which means we consider the system of first order differential equations

$$\begin{aligned} q'(x) &= p(x) \\ p'(x) &= \frac{K^2}{|q(x)|^2} \cdot \left(-\frac{q(x)}{|q(x)|} + \frac{1}{c^2} \cdot A_1 \right) \end{aligned}$$

By applying the midpoint rule one gets

$$\begin{aligned} q_{k+1} &= q_k + \frac{h}{2} \cdot (p_{k+1} + p_k) \\ p_{k+1} &= p_k + h \cdot \frac{4 \cdot K^2}{|q_{k+1} + q_k|^2} \cdot \Bigg\{ -\frac{q_{k+1} + q_k}{|q_{k+1} + q_k|} + \frac{1}{c^2} \cdot \Bigg[\Bigg((4 + 2 \cdot \eta) \cdot \frac{2 \cdot K^2}{|q_{k+1} + q_k|} - (1 + 3 \cdot \eta) \cdot |p_k|^2 \\ & \quad + \frac{3}{2} \cdot \eta \cdot \left(\frac{2}{|q_{k+1} + q_k|} \cdot \left\langle \frac{p_k + p_{k+1}}{2}, \frac{q_{k+1} + q_k}{2} \right\rangle \right)^2 \Bigg) \cdot \frac{q_k + q_{k+1}}{|q_{k+1} + q_k|} \\ & \quad + (4 - 2 \cdot \eta) \cdot \frac{2}{|q_{k+1} + q_k|} \cdot \left\langle \frac{p_k + p_{k+1}}{2}, \frac{q_{k+1} + q_k}{2} \right\rangle \cdot p_k \Bigg] \Bigg\} \end{aligned}$$

The full algorithm can be found in appendix B.

In practice this preserves much better the Runge-Lenz vector, as can be seen in figure 1.2. Here the error with respect to the Runge-Lenz vector is roughly periodical, whereas previously the error increased with every cycle. In both cases the error of the Hamiltonian has a peak at every period. After applying the correction, the error of the angular momentum shows similar behavior as the error of the Hamiltonian, changing its magnitude at every cycle, whereas previously it was nearly constant at zero. In absolute numbers the errors become in this example bigger by applying the correction in

comparison to the previous method. But as mentioned, it has the big advantage, that now we don't have that unbounded growth of error in the Runge-Lenz vector.

In both cases we used the same initial data, only adding $\eta = 10$ to the corrected version.

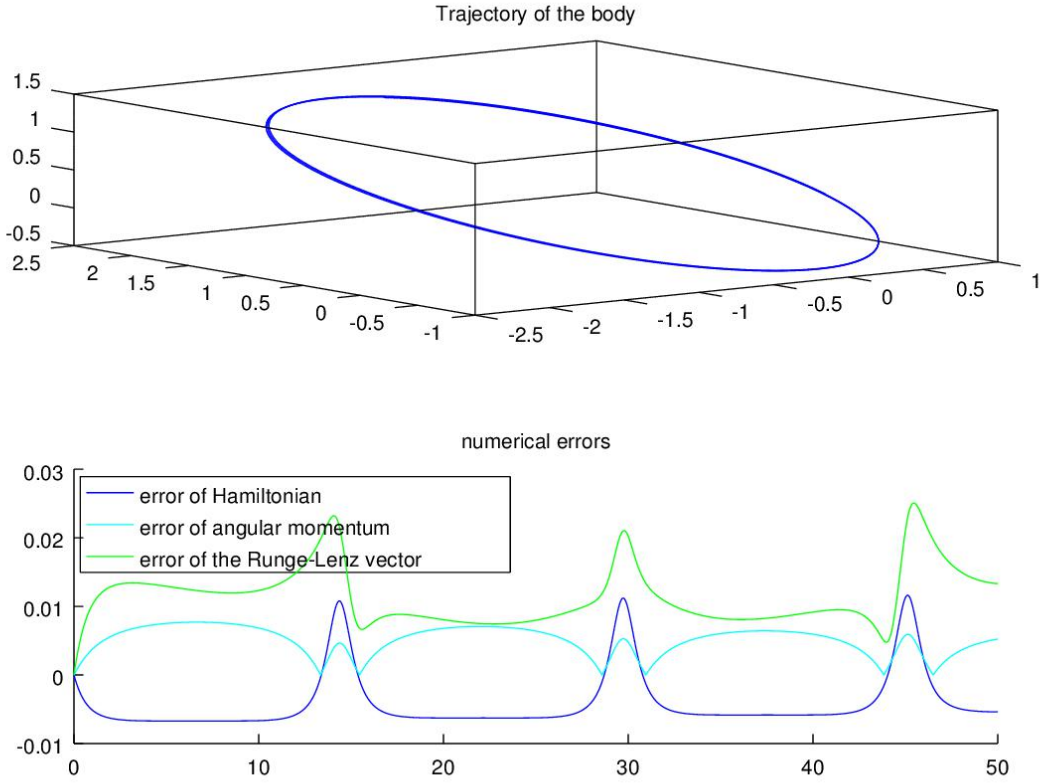


Figure 1.2: trajectory and numerical errors for the reduced one-body problem with EIH-correction

1.3. The Kozlov-method

The motivation behind adding a further method is to increase the order of approximation. So far we considered methods of second order of approximation, now we want to extend it to order four, (cf. [2]). This method is based on using the Kustaanheimo-Stiefel transformation, which puts the three-dimensional reduced one-body problem to a four-dimensional task.

The derivation is mainly based on [2].

Starting from the ODE-system given by (1.5) with Hamiltonian (1.6), we first apply the Poincaré transform, which is a time transformation

$$\frac{dt}{ds} = |q|$$

This leads to the new Hamiltonian

$$\hat{H}(q,p) = |q| \cdot (H(q,p) + A) = \frac{1}{2} \cdot |q| \cdot |p|^2 + A \cdot |q| - K^2 \tag{1.8}$$

with $A = -H(q_0, p_0)$. Second, we continue with the transformation of the coordinates in the matter of

$$q = \Lambda(Q) \cdot Q \tag{1.9}$$

$$p = \frac{1}{2 \cdot |Q|^2} \cdot \Lambda(Q) \cdot P \qquad Q, P \in \mathbb{R}^4 \tag{1.10}$$

with

$$\Lambda(Q) = \begin{bmatrix} Q_1 & -Q_2 & -Q_3 & Q_4 \\ Q_2 & Q_1 & -Q_4 & -Q_3 \\ Q_3 & Q_4 & Q_1 & Q_2 \end{bmatrix}$$

By taking $|q| = |Q|^2$ and $|p|^2 = \frac{|P|^2}{4 \cdot |Q|^2}$ into consideration, one gets from (1.8) the Hamiltonian for the new coordinates

$$H_{KS}(Q,P) = \frac{1}{8} \cdot |P|^2 + A \cdot |Q|^2 - K^2$$

This will give us the ODE in terms of the new coordinates as follows

$$\begin{aligned} \frac{dQ}{ds} &= \tfrac{\partial}{\partial P} H_{KS}(Q,P) &&= \frac{1}{4} \cdot P \\ \frac{dP}{ds} &= -\tfrac{\partial}{\partial Q} H_{KS}(Q,P) &&= -2 \cdot A \cdot Q \end{aligned}$$

All together we get

$$\frac{d}{ds} \begin{bmatrix} Q \\ P \\ t \end{bmatrix} = \begin{bmatrix} \frac{1}{4} \cdot P \\ -2 \cdot A \cdot Q \\ |Q|^2 \end{bmatrix} =: f(Q,P,t)$$

To obtain the forth order method, we consider the so called modified vector field given in this case by

$$f_h = \begin{bmatrix} \frac{1}{4} \cdot \left(1 + \frac{h^2}{24} \cdot A\right) \cdot P \\ -2 \cdot A \cdot \left(1 + \frac{h^2}{24} \cdot A\right) \cdot Q \\ \left(1 + \frac{h^2}{24} \cdot A\right) \cdot |Q|^2 + \frac{h^2}{192} \cdot |P|^2 \end{bmatrix}$$

where h denotes the step size of the numerical integrator. To get a closer look at the derivation of this modified vector field, have a look at [2]. By applying the midpoint rule to the modified vector field one gets

$$Q_{j+1} = Q_j + h \cdot \frac{1}{4} \cdot \left(1 + \frac{h^2}{24} \cdot A\right) \cdot \frac{P_{j+1} + P_j}{2} \tag{1.11}$$

$$P_{j+1} = P_j - h \cdot 2 \cdot A \cdot \left(1 + \frac{h^2}{24} \cdot A\right) \cdot \frac{Q_{j+1} + Q_j}{2}$$

$$t_{j+1} = t_j + h \cdot \left(\left(1 + \frac{h^2}{24} \cdot A\right) \cdot \left| \frac{Q_{j+1} + Q_j}{2} \right|^2 + \frac{h^2}{192} \cdot \left| \frac{P_{j+1} + P_j}{2} \right|^2 \right) \tag{1.12}$$

In fact this can be rewritten in order to explicitly compute the solution by changing (1.11) equivalently to

$$Q_{j+1} = \frac{Q_j \cdot \left(1 - \frac{1}{8} \cdot h^2 \cdot \left(1 + \frac{h^2}{24} \cdot A\right)^2 \cdot A\right) + \frac{1}{4} \cdot P_j \cdot h \cdot \left(1 + \frac{h^2}{24} \cdot A\right)}{1 + \frac{1}{8} \cdot h^2 \cdot \left(1 + \frac{h^2}{24} \cdot A\right)^2 \cdot A}$$

So far we only considered the transformation from the four-dimensional to the three-dimensional coordinates. In fact for the implementation, given in appendix C, we will need the transformation from the initial coordinates towards the four-dimensional ones as well. At least to determine the initial values in terms of the transformed coordinates. Of course the backwards transformation from the three- to the four-dimensional case isn't unique. In our case we've chosen $Q_{0,4} = 0$. Under the assumption that the first component of the initial data q_0 is greater than zero, one gets from (1.9) for the remaining components

$$\begin{aligned} Q_{0,1} &= \sqrt{q_{0,1}} \\ Q_{0,2} &= \frac{q_{0,2}}{2 \cdot Q_{0,1}} \\ Q_{0,3} &= \frac{q_{0,3}}{2 \cdot Q_{0,1}} \end{aligned}$$

where $q_{0,i}$ denotes the i-th component of q_0 and $Q_{0,i}$ similarly. Note that we mentioned before, that this transformation is only necessary for the initial data. For the transformation of the initial momentum p_0 we use a condition given in [2], which is

$$\begin{bmatrix} Q_{0,4} & -Q_{0,3} & Q_{0,2} & -Q_{0,1} \end{bmatrix} \cdot P_0 = 0$$

As $\begin{bmatrix} Q_{0,4} & -Q_{0,3} & Q_{0,2} & -Q_{0,1} \end{bmatrix} \cdot \Lambda(Q_0)^T = 0$, we use the suggestion of [2] for the transformation:

$$P_0 = 2 \cdot \Lambda(Q_0)^T \cdot p_0 \tag{1.13}$$

One can easily recompute, that (1.13) fulfills (1.10) for the initial value p_0, as especially it holds $\Lambda(Q) \cdot \Lambda(Q)^T = |Q|^2 \cdot \mathbb{1}$.

So in the end with the same initial condition we used before, one gets figure 1.3. Once again, like in figure 1.1, the angular momentum has an apparently constant error, which is now greater than zero. Both other errors are periodically again, but the curves look different now. The error of the Hamiltonian still has their maxima at every cycle with respect to absolute value. But now the peaks point to the opposite direction, although the error is computed in the same way as previously. In absolute numbers the errors are in general greater than in the previous two chapters. Note that the numerical errors are computed with respect to the three-dimensional coordinates. A way to compute them with respect to the transformed coordinates is given in [2]. Additionally we computed the global error of the preserved quantities, so we can better observe the change of the error

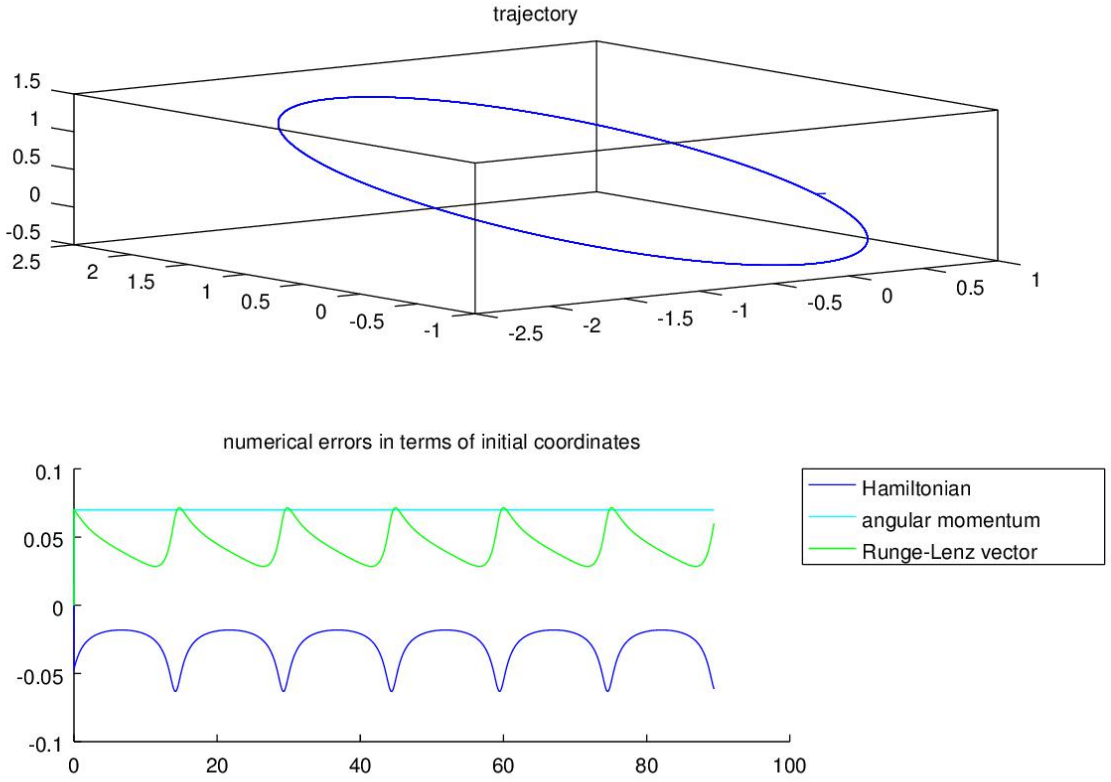


Figure 1.3: trajectory and numerical errors for the reduced one-body problem with the Kozlov-method

when varying the step size h and number of iterations N while keeping $h \cdot N$ constant. Global error in this case means approximating the error (here for the Hamiltonian) like

$$\left(\int_{t_0}^{t_N} |err_H(x)|^2 \, dx \right)^{1/2} \approx \sqrt{\sum_{i=0}^{N-1} (t_{i+1} - t_i) \cdot |err_H(t_{i+1})|^2}$$

where $err_H(x)$ is given by (1.7) and t_i are computed via (1.12) with initial value $t_0 = 0$.

In the example shown in table 1.1, where we used the initial coordinates like previously and $h \cdot N = 20$, one can see that the error of the Hamiltonian decreases, while increasing the number of iterations and thus decreasing the step size. The error of the angular momentum increases first a bit, but than stays constant and the one with respect to the Runge-Lenz-vector even increases a bit throughout every run.

h/N / error of	0.1/200	0.01/2000	0.005/4000
Hamiltonian	0.28722	0.28706	0.28705
Angular Momentum	0.66068	0.66072	0.66072
Runge-Lenz vector	0.43717	0.43861	0.43869

Table 1.1: global error of the preserved quantities

2.Keplers two-body problem

In this section we will go back to the initial problem of two moving bodies which affect each others orbits. Here we want to implement the two-body problem without reduction.

From (1.2) combined with (1.1) one gets the equations to begin with by

$$\begin{aligned} y_1''(x) &= -G \cdot \frac{m_2}{|y_1(x) - y_2(x)|^3} \cdot (y_1(x) - y_2(x)) \\ y_2''(x) &= G \cdot \frac{m_1}{|y_1(x) - y_2(x)|^3} \cdot (y_1(x) - y_2(x)) \end{aligned} \tag{2.1}$$

Once again using the approach of chapter 1.1, one gets the system of first order differential equations

$$\begin{aligned} y_i'(x) &= p_i(x) \\ p_i'(x) &= y_i''(x) \qquad i = 1, 2 \end{aligned}$$

By inserting (2.1) and applying the midpoint rule directly to the resulting system, one gets the iteration formula

$$\begin{aligned} y_{i,k+1} &= y_{i,k} + \frac{h}{2} \cdot (p_{i,k+1} + p_{i,k}) \\ p_{1,k+1} &= p_{1,k} + h \cdot \left(-4 \cdot G \cdot \frac{m_2}{|y_{1,k+1} + y_{1,k} - y_{2,k+1} - y_{2,k}|^3} \cdot (y_{1,k+1} + y_{1,k} - y_{2,k+1} - y_{2,k}) \right) \\ p_{2,k+1} &= p_{2,k} + h \cdot \left(4 \cdot G \cdot \frac{m_1}{|y_{1,k+1} + y_{1,k} - y_{2,k+1} - y_{2,k}|^3} \cdot (y_{1,k+1} + y_{1,k} - y_{2,k+1} - y_{2,k}) \right) \end{aligned}$$

To solve the resulting implicit equation system, one can use the fixed point iteration. In the algorithm presented in appendix D, 100 iterations were used to determine the next iteration. With initial values $q_{1,0} = (0, 0, 0)$, $q_{2,0} = (1, 0.5, 0)$, $p_{1,0} = (0.1, 0.2, 0)$, $p_{2,0} = (-0.1, 1.2, 0.5)$ and choice of parameters $m_1 = \frac{1}{2.6} \cdot 10^{11}$, $m_2 = 1.4 \cdot 10^6$, $h = 0.01$ and $N = 2000$, one gets the trajectories shown in 2.1. It's the motion of one relatively light body with comparatively high velocity around an heavy body with less velocity.

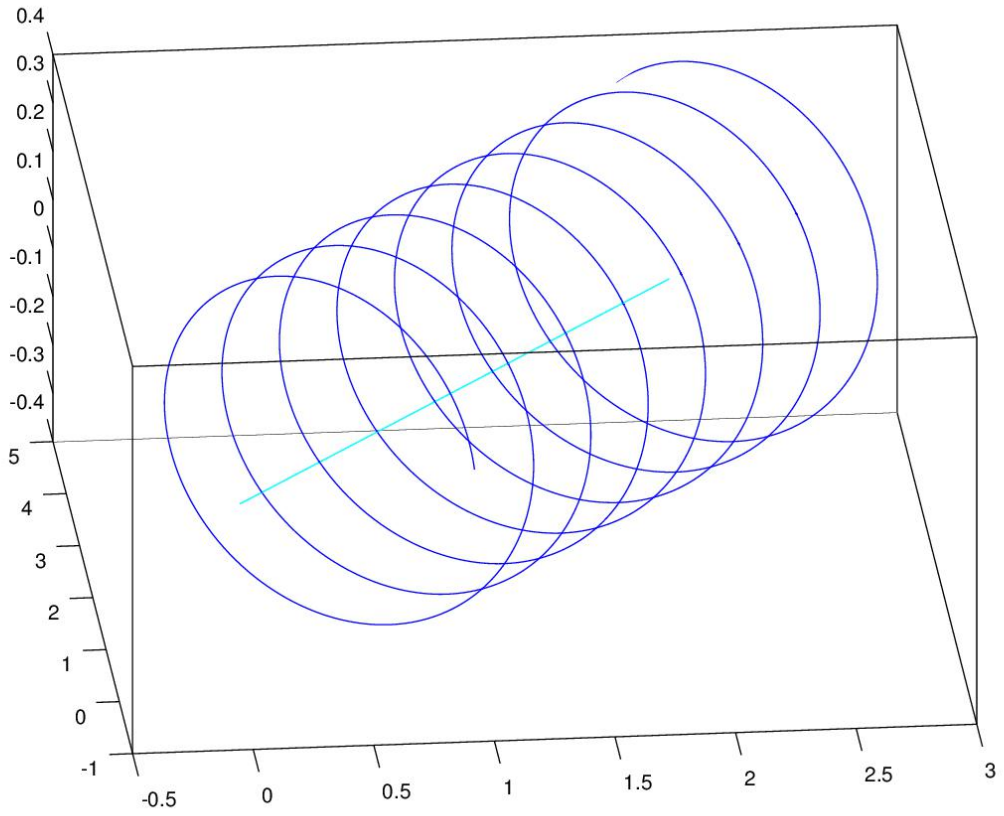


Figure 2.1: trajectories of the two-body problem

3.Keplers three-body problem

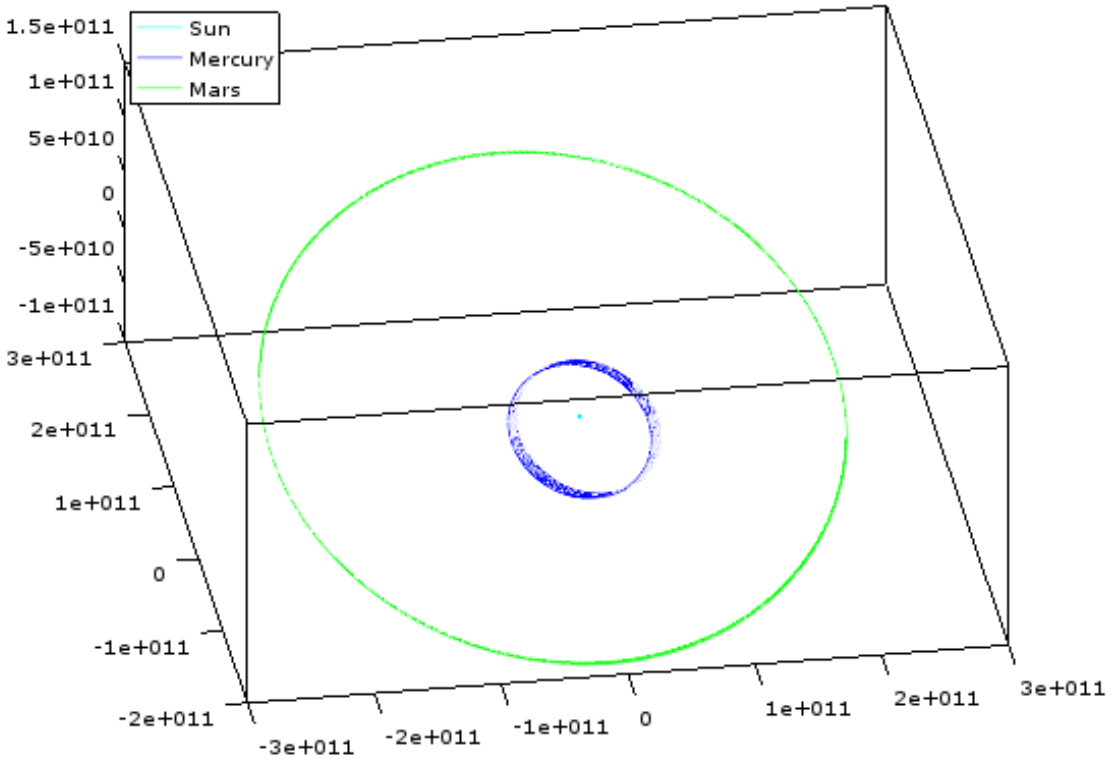
As we mentioned at the very beginning, the aim of this paper is to get a rough idea about modeling planetary motion of our solar system. So finally we want to put this into effect by considering the motion of three planets. The differential equations which got to be solved for getting the trajectories are

$$y_1''(x) = -G \cdot m_2 \cdot \frac{y_1(x) - y_2(x)}{|y_1(x) - y_2(x)|} - G \cdot m_3 \cdot \frac{y_1(x) - y_3(x)}{|y_1(x) - y_3(x)|}$$
$$y_2''(x) = -G \cdot m_3 \cdot \frac{y_2(x) - y_3(x)}{|y_2(x) - y_3(x)|} - G \cdot m_1 \cdot \frac{y_2(x) - y_1(x)}{|y_2(x) - y_1(x)|}$$
$$y_3''(x) = -G \cdot m_1 \cdot \frac{y_3(x) - y_1(x)}{|y_3(x) - y_1(x)|} - G \cdot m_2 \cdot \frac{y_3(x) - y_2(x)}{|y_3(x) - y_2(x)|}$$

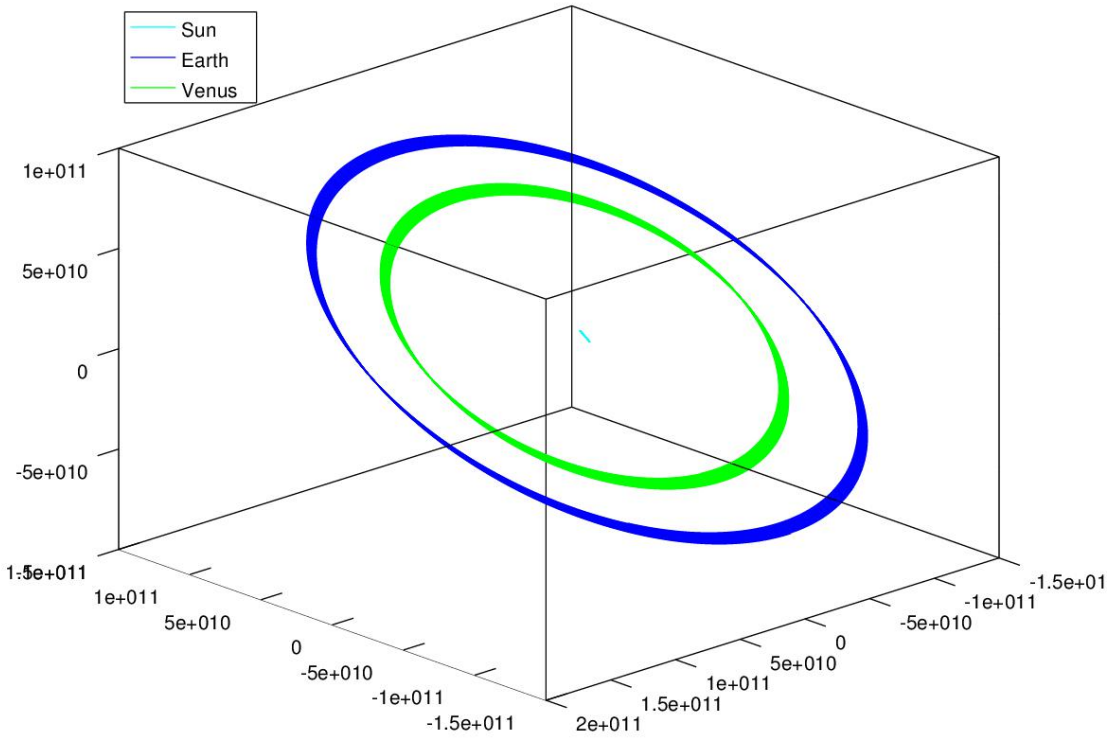
To get the algorithm presented in appendix E, one can proceed like in the previous chapter. By modeling the motion of sun, mercury and mars one gets 3.1a, and for the motion of sun, venus and earth one gets 3.1b. The initial data as given in [3] is shown in table 3.1.

	position	**velocity**	**mass**
sun	$5.5850 \cdot 10^8 \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -1.4663 \\ 11.1238 \\ 4.8370 \end{bmatrix}$	$1.99 \cdot 10^{33}$
mercury	$\begin{bmatrix} 5.1979 \cdot 10^{10} \\ 7.6928 \cdot 10^9 \\ -1.2845 \cdot 10^9 \end{bmatrix}$	$\begin{bmatrix} -1.5205 \\ 4.4189 \\ 2.5180 \end{bmatrix} \cdot 10^4$	$3.30 \cdot 10^{23}$
venus	$\begin{bmatrix} -1.5041 \\ 9.7080 \\ 4.4635 \end{bmatrix} \cdot 10^{10}$	$\begin{bmatrix} -3.4770 \cdot 10^4 \\ -5.5933 \cdot 10^3 \\ -316.8994 \end{bmatrix}$	$4.87 \cdot 10^{24}$
earth	$\begin{bmatrix} -1.1506 \cdot 10^9 \\ -1.3910 \cdot 10^{11} \\ -6.0330 \cdot 10^{10} \end{bmatrix}$	$\begin{bmatrix} 2.9288 \cdot 10^4 \\ -398.5759 \\ -172.5873 \end{bmatrix}$	$5.97 \cdot 10^{24}$

Table 3.1: initial data for modeling part of the solar system



(a) Orbit of sun, mercury and mars



(b) Orbit of sun, earth and venus

Figure 3.1: modeling part of the solar system

4. outlook: Modeling the solar system

Finally we want to give an example of how to implement our complete solar system. The equations of motion, a generalization of those of two respectively three bodies, are given by

$$y_i''(x) = G \cdot \sum_{\substack{k=1 \\ k \neq i}}^{n} m_k \cdot \frac{y_k(x) - y_i(x)}{|y_k(x) - y_i(x)|^3}$$

for each body. In our case $n = 9$ with the sun plus eight planets. The method and the numeric to solve this ODE-system is the same as this for the two- and the three-body problem. Merely the implementation in matlab is different. Here we used a matrix notation to save time. The resulting algorithm can be found in appendix F. Again the initial data is taken from [3]. Note that in the picture given in figure 4.1 for the planets close to the sun not all computed points were printed. This is because of the change in the inclination with every cycle we mentioned before, which is due to numerical errors. By the total number of iterations used, this would lead to a ring shaped presentation of the trajectories as it is indicated in picture detail 4.2. Apart from this, one gets a nice presentation of the planetary orbits shown in figure 4.1.

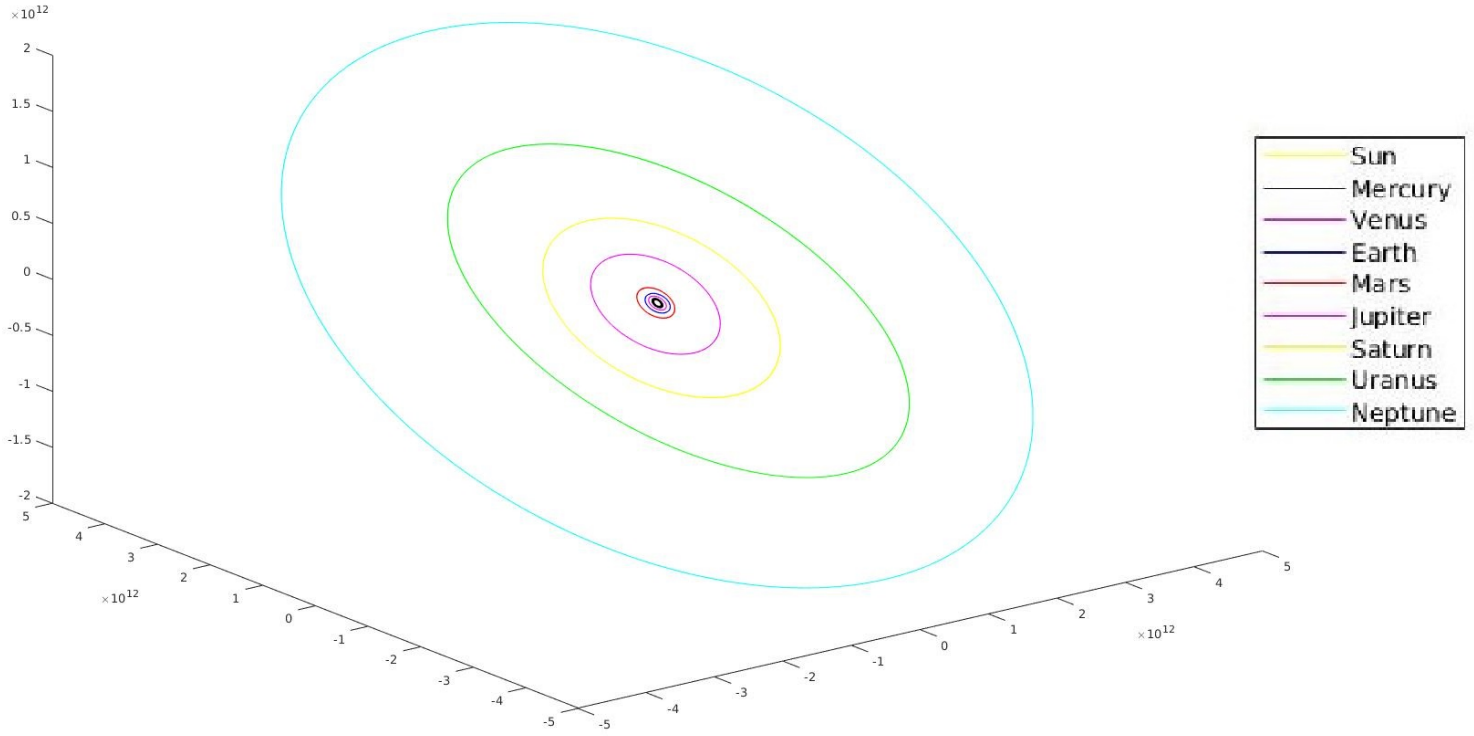


Figure 4.1: planetary orbits

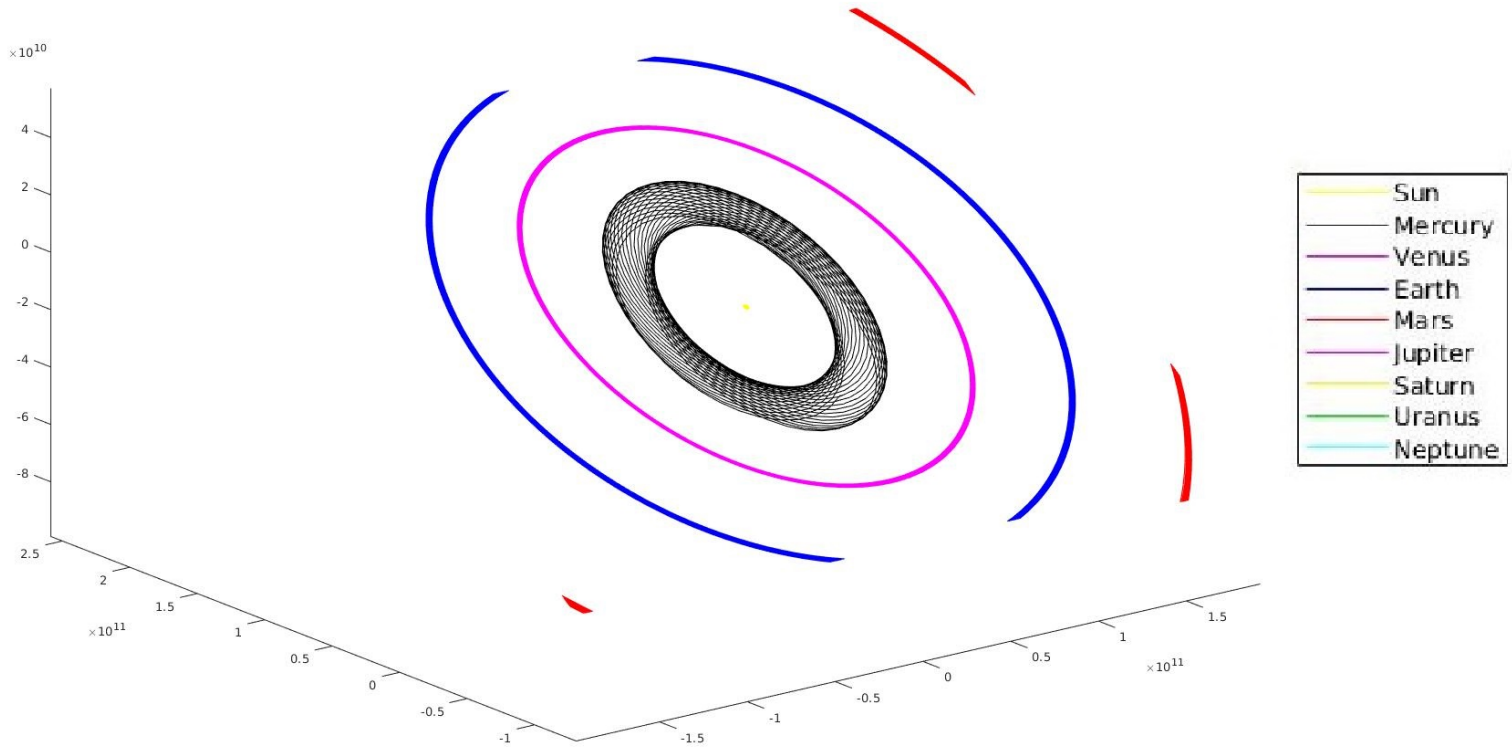


Figure 4.2: picture detail of the planets close to the sun

Conclusion

Beginning with Keplers reduced one-body problem, we compared three different methods with respect to their numerical errors, for which computation we used their deviation of the preserved quantities. First we introduced a quite simple method by just applying the midpoint rule to the considered ODE system. Although the errors for the angular momentum and the Hamiltonian are really small, the method is not to be recommended in general as one gets a deviation in the inclination with every cycle. Therefore the error of the Runge-Lenz vector can increase unbounded. This observation lead us to the relativistic EIH-correction which avoids the latter problem. However, the error of the Hamiltonian and the angular momentum at least in our example is in absolute numbers bigger than previously, but still relatively small. Same can be said about the Kozlov-method, which was discussed afterwards. So in the end both, Kozlov-method and the EIH-corrected variant of the midpoint rule, produce good results.

We then gave an idea of how to model the solar system by first considering the classical problem of two moving bodies and then extend it to three moving bodies. In the latter case we explicitly had a look at the orbits of mars and mercury respectively earth and venus around the sun. To finish, we offered a quick example of how to model our complete solar system.

References

[1] Borzì, Alfio. *Modelling with Ordinary Differential Equations: A Comprehensive Approach*. London: CRC/Chapman and Hall, to be published in 2019. Chapter 8.3.

[2] Kozlov, Roman. "Conservative discretizations of the Kepler motion". In: *Journal of Physics A: Mathematical and Theoretical* 40.17 (2007), pages 4529–4539. DOI: 10.1088/1751-8113/40/17/009.

[3] Inc. The MathWorks, editor. *Model Gravity in a Planetary System*. URL: https://de.mathworks.com/help/physmod/sm/ug/model-planet-orbit-due-to-gravity.html (visited on 07/31/2019).

List of Figures

A. Reduced Keplers one-body problem: midpoint rule

```
function KeplerOneBodyMidpoint(q0,p0,h,N)
% computes the reduced Kepler one-body problem, i.e. gives the trajectory
% for given startposition q0 and startmomentum p0; h is step size and N
% number of iterations;
% additionally the numerical errors w.r.t the preserved quantities
% (Hamiltonian, angular momentum and Runge-Lenz vector) are computed
% and given in a graphic
% the ODEs are approximated by the midpoint scheme

    K=1;
    % determination of the variable

    eps=10^(-3);
    % determination of the precision of the Newton iteration

    q(1:3,1)=q0;
    p(1:3,1)=p0;
    % saving the initial values in the matrix, edited later on

    for i=1:N
        n0=q(1:3,i);
        F=@(n)(n-q(1:3,i)-h*(p(1:3,i)-h*K^2*2/((n+q(1:3,i))'
                *(n+q(1:3,i)))^(3/2)*(n+q(1:3,i))));
        % next iteration of q is root of function above; solve this problem
        % with newton iteration underneath
        err=(F(n0)'*F(n0))^(1/2);
        while err > eps
            J=eye(3)+2*h^2*K^2*(1/((n0+q(1:3,i))'*(n0+q(1:3,i)))^(3/2)
                        *eye(3)-3*1/((n0+q(1:3,i))'*(n0+q(1:3,i)))^(5/2)
                        *(n0+q(1:3,i))*(n0+q(1:3,i))');
            x=J\(-F(n0));
            n0=x+n0;
            err=(F(n0)'*F(n0))^(1/2);
        end
        q(1:3,i+1)=n0;
        p(1:3,i+1)=p(1:3,i)-h*K^2*4/((n0+q(1:3,i))'*(n0+q(1:3,i)))^(3/2)
                *(q(1:3,i+1)+q(1:3,i));
        % compute next p with midpoint rule
    end

    plot3(q(1,:),q(2,:),q(3,:));

    H=@(q,p)(1/2*(p'*p)-K^2/(q'*q)^(1/2));
    % Hamilton function
```

```
45
46      R=@(p,l,q)(cross(p,l)-K^2/(q'*q)^(1/2)*q);
47      % Runge-Lenz vector
48
49      l0=cross(q0,p0);
50      for i=1:N+1
51          errH(i)=H(q(1:3,i),p(1:3,i))-H(q0,p0);
52          % error of Hamilton function
53          l=cross(q(1:3,i),p(1:3,i));
54          % angular momentum
55          errl(i)=((l-l0)'*(l-l0))^(1/2);
56          % error of the angular momentum
57          R0=R(p0,l0,q0);
58          Ri=R(p(1:3,i),l,q(1:3,i));
59          errR(i)=((Ri-R0)'*(Ri-R0))^(1/2);
60          % error of Runge-Lenz vector
61      end
62
63
64      x=0:h:N*h;
65
66      subplot(2,1,1);
67      plot3(q(1,:),q(2,:),q(3,:));
68      title('Trajectory of the body');
69      subplot(2,1,2);
70      hold on
71      plot(x,errH);
72      plot(x,errl);
73      plot(x,errR);
74      title('numerical errors');
75      legend('error of Hamiltonian','error of angular momentum','error of
76          the Runge-Lenz vector','Location','northwest');
77      hold off
78      % plotting the trajectory and the numerical errors
79
80  end
```

B. Reduced Keplers one-body problem with EIH-correction

```
function KeplerOneBodyEIH2(q0,p0,h,N,eta)
% computes the reduced Kepler one-body problem, i.e. gives the trajectory
% for given startposition q0 and startmomentum p0 under consideration of
% the EIH-correction; h is step size and N number of iterations; eta a
% mass proportion
% additionally the numerical errors w.r.t the preserved quantities
% (Hamiltonian, angular momentum and Runge-Lenz vector) are computed
% and given in a graphic
% the ODEs are approximated by the midpoint scheme

    K=1;
    % determination of the variable

    eps=10^(-3);
    % determination of the precision of the Newton iteration

    c=(p0'*p0)^(1/2)/sqrt(eps);
    % choosing the parameter c with respect to given eps, eps sufficiently
    % small

    q(1:3,1)=q0;
    p(1:3,1)=p0;
    % saving the initial values in the matrix, edited later on

    for i=1:N
        n0=q(1:3,i);
        F=@(n)(n-q(1:3,i)-h*(p(1:3,i)-h*K^2*2/((n+q(1:3,i))'
                *(n+q(1:3,i)))^(3/2)*(n+q(1:3,i))));
        % next iteration of q is root of function above; solve this problem
        % with newton iteration underneath
        err=(F(n0)'*F(n0))^(1/2);
        while err > eps
            J=eye(3)+2*h^2*K^2*(1/((n0+q(1:3,i))'*(n0+q(1:3,i)))^(3/2)
                        *eye(3)-3*1/((n0+q(1:3,i))'*(n0+q(1:3,i)))^(5/2)
                        *(n0+q(1:3,i))*(n0+q(1:3,i))');
            x=J\(-F(n0));
            n0=x+n0;
            err=(F(n0)'*F(n0))^(1/2);
        end
        q(1:3,i+1)=n0;

        y0=p(1:3,i);
        s=q(1:3,i+1)+q(1:3,i);
        n=(s'*s)^(1/2);
```

```
45          % constants which are used often in the following formula
46
47          G=@(y)(p(1:3,i)+h*K^2*4*1/n^2*(-s/n+1/(c^2)*(((4+2*eta)*K^2
48                  *2/n-(1+3*eta)*(p(1:3,i)'*p(1:3,i))+3/2*eta*((p(1:3,i)+y)'
49                  *s/(2*n))^2)*s/n+(4-2*eta)*((p(1:3,i)+y)'*s/(2*n))
50                  *p(1:3,i)))-y);
51          %use Newton iteration to find the next iteration of p
52          err2=(G(y0)'*G(y0))^(1/2);
53          while err2 > eps
54              J2=-eye(3)+h*K^2*4/n*1/(c^2)*(3/2*eta/n*((y0+p(1:3,i))'*s)
55                  *s/(2*n)*(s/n)'+(4-2*eta)/n*p(1:3,i)*s'/2);
56              x2=J2\(-G(y0));
57              y0=x2+y0;
58              err2=(G(y0)'*G(y0))^(1/2);
59          end
60          p(1:3,i+1)=y0;
61      end
62      plot3(q(1,:),q(2,:),q(3,:));
63
64      H=@(q,p)(1/2*(p'*p)-K^2/(q'*q)^(1/2));
65      % Hamilton function
66
67      R=@(p,l,q)(cross(p,l)-K^2/(q'*q)^(1/2)*q);
68      % Runge-Lenz vector
69
70      errH=zeros(1,N+1);
71      errl=zeros(1,N+1);
72      errR=zeros(1,N+1);
73      % preallocation
74
75      l0=cross(q0,p0);
76      for i=1:N+1
77          errH(i)=H(q(1:3,i),p(1:3,i))-H(q0,p0);
78          % error of Hamilton function
79          l=cross(q(1:3,i),p(1:3,i));
80          % angular momentum
81          errl(i)=((l-l0)'*(l-l0))^(1/2);
82          % error of the angular momentum
83          R0=R(p0,l0,q0);
84          Ri=R(p(1:3,i),l,q(1:3,i));
85          errR(i)=((Ri-R0)'*(Ri-R0))^(1/2);
86          % error of Runge-Lenz vector
87      end
88
89
90      grid=0:h:N*h;
```

```
91
92      subplot(2,1,1);
93      plot3(q(1,:), q(2,:), q(3,:));
94      title('Trajectory of the body');
95      subplot(2,1,2);
96      hold on
97      plot(grid, errH);
98      plot(grid, errl);
99      plot(grid, errR);
100     title('numerical errors');
101     legend('error of Hamiltonian', 'error of angular momentum', 'error of
102         the Runge-Lenz vector', 'Location', 'northwest');
103     hold off
104     % plotting the trajectory and the numerical errors
105
106 end
```

C. Reduced Keplers one-body problem with Kozlov-method

```
function KeplerKozlov(q0,p0,h,N)
% computes the reduced Kepler one-body problem, i.e. gives the trajectory
% for given startposition q0 and startmomentum p0; h is step size and N
% number of iterations;
% in this case the Kepler problem is approximated by the Kozlov-method,
% i.e. first we transform the coordinates in terms of Kustaanheimo-Stiefel
% and then we apply the midpoint rule to approximate the ODEs
% additionally the numerical errors w.r.t the preserved quantities
% (Hamiltonian, angular momentum and Runge-Lenz vector) are computed
% and given in a graphic; also the global errors are given w.r.t. this
% quantities

    K=1;
    % determination (constant)

    L=@(v)([v(1),-v(2),-v(3),v(4);v(2),v(1),-v(4),-v(3);v(3),v(4),v(1),
            v(2)]);

    Q0=zeros(4,1);
    Q0(1)=sqrt(q0(1));
    Q0(2)=q0(2)/(2*Q0(1));
    Q0(3)=q0(3)/(2*Q0(1));
    Q0(4)=0;
    P0=2*L(Q0)'*p0;
    % transformation of the initial values into the new parametrization;
    % we chose Q0(4)=0; necessary condition to make this work is q0(1)>0

    A=-1/2*(p0'*p0)+K^2/(q0'*q0)^(1/2);
    a=1+h^2/24*A;
    alpha=a;
    beta=h^2/192;
    % constants for the iteration (for fourth-order method)

    Q=zeros(4,N+1);
    P=zeros(4,N+1);
    q=zeros(3,N+1);
    p=zeros(3,N+1);
    t=zeros(1,N+1);
    % preallocating

    Q(1:4,1)=Q0;
    P(1:4,1)=P0;
    q(1:3,1)=q0;
    p(1:3,1)=p0;
```

```
45      t(1)=0;
46      % saving the initial values in the matrix, used later on for edition
47      % and numerical errors
48
49      H1=@(q,p)(1/2*p'*p-K^2/(q'*q)^(1/2));
50      % Hamiltonian
51      l1=@(p,q)(cross(q,p));
52      % angular momentum
53      R1=@(p,q)(cross(p,l1(p,q))-K^2/(q'*q)^(1/2)*q);
54      % Runge-Lenz vector
55      % conserved quantities (corresponding functions) in terms of the
56      % initial coordinates
57
58      errH1=zeros(1,N+1);
59      errl1 =zeros(1,N+1);
60      errR1=zeros(1,N+1);
61      errHam=0;
62      errAngMom=0;
63      errRLV=0;
64      % preallocating for the numerical errors
65      % note, that the first component indeed always equals zero
66
67      h0=H1(q0,p0);
68      l0=l1(p0,q0);
69      r0=R1(p0,q0);
70      % initial data, for comparison needed to compute the numerical errors
71
72      for i=1:N
73          Q(1:4,i+1)=(Q(1:4,i)*(1-h^2/8*a^2*A)+P(1:4,i)*h/4*a)/(1+h^2/8
74                  *a^2*A);
75          q(1:3,i+1)=L(Q(1:4,i+1))*Q(1:4,i+1);
76          P(1:4,i+1)=P(1:4,i)-h*2*A*a*(Q(1:4,i+1)+Q(1:4,i))/2;
77          p(1:3,i+1)=1/(2*Q(1:4,i+1)'*Q(1:4,i+1))*L(Q(1:4,i+1))*P(1:4,i+1);
78          t(i+1)=t(i)+h*(alpha/4*((Q(1:4,i)+Q(1:4,i+1))'*(Q(1:4,i)
79                  +Q(1:4,i+1)))+beta/4*((P(1:4,i)+P(1:4,i+1))'*(P(1:4,i)
80                  +P(1:4,i+1))));
81          % computing the next iterations and the backwards transformation to
82          % the initial coordinates
83
84          errH1(i+1)=H1(q(1:3,i+1),p(1:3,i+1))-h0;
85          errl1 (i+1)=((l1(p(1:3,i+1),q(1:3,i+1))-l0)'*(l1(p(1:3,i+1),
86                  q(1:3,i+1))-l0))^(1/2);
87          errR1(i+1)=((R1(p(1:3,i+1),q(1:3,i+1))-r0)'*(R1(p(1:3,i+1),
88                  q(1:3,i+1))-r0))^(1/2);
89          % numerical errors
90
```

```
91          errHam=errHam+(t(i+1)-t(i))*errH1(i+1)^2;
92          errAngMom=errAngMom+(t(i+1)-t(i))*errl1(i+1)^2;
93          errRLV=errRLV+(t(i+1)-t(i))*errR1(i+1)^2;
94          % for computing the global error
95      end
96
97      errHam=sqrt(errHam)
98      errAngMom=sqrt(errAngMom)
99      errRLV=sqrt(errRLV)
100     % global error
101
102     subplot(2,1,1)
103     plot3(q (1,:), q (2,:), q (3,:));
104      title (' trajectory ' );
105     subplot(2,1,2)
106     hold on
107     plot(t,errH1,'g' );
108     plot(t, errl1 ,'b' );
109     plot(t,errR1,'c' );
110     hold off
111      title ('numerical errors in terms of initial coordinates' );
112     legend('Hamiltonian','angular momentum','Runge-Lenz vector',
113          'Location',' northeastoutside ' );
114
115 end
```

D. Keplers two-body problem

```
function KeplerTwoBody(q01,q02,p01,p02,m1,m2,h,N)
% computes Keplers two-body problem, i.e. gives the trajectories
% for given start positions q01 and q02 and startmomentum p01 and p02;
% m1 and m2 name the mass of each body, h is step size and N number of
% iterations;
% the ODEs are approximated by the use of the midpoint rule

    G=6.6743*10^(-11);
    % gravitational constant

    q1=zeros(3,N+1);
    p1=zeros(3,N+1);
    q2=zeros(3,N+1);
    p2=zeros(3,N+1);
    % preallocating

    q1(1:3,1)=q01;
    p1(1:3,1)=p01;
    q2(1:3,1)=q02;
    p2(1:3,1)=p02;
    % take over the initial values

    for i=1:N
        s1=p1(1:3,i);
        s2=p2(1:3,i);
        t1=q1(1:3,i);
        t2=q2(1:3,i);
        % start values for fixed point iteration

        for k=1:100
            T1=q1(1:3,i)+h/2*(s1+p1(1:3,i));
            T2=q2(1:3,i)+h/2*(s2+p2(1:3,i));
            a=t1+q1(1:3,i)-t2-q2(1:3,i);
            S1=p1(1:3,i)-4*h*G*m2*a/(a'*a)^(3/2);
            S2=p2(1:3,i)+4*h*G*m1*a/(a'*a)^(3/2);
            s1=S1;
            s2=S2;
            t1=T1;
            t2=T2;
        end
        % fixed point iteration

        p1(1:3,i+1)=s1;
        p2(1:3,i+1)=s2;
```

```
45          q1(1:3, i+1)=t1;
46          q2(1:3, i+1)=t2;
47          % next iterations
48
49      end
50
51      hold on
52      plot3(q1(1,:), q1(2,:), q1(3,:), 'c');
53      plot3(q2(1,:), q2(2,:), q2(3,:), 'b');
54      hold off
55
56  end
```

E. Keplers three-body problem

```
function KeplerThreeBody(q01,q02,q03,p01,p02,p03,m1,m2,m3,h,N)
% computes Keplers three-body problem, i.e. gives the trajectories
% for given start positions q01, q02 and q03 and startmomentum p01,p02
% and p03; m1, m2 and m3 name the mass of each body, h is step size and
% N number of iterations;
% the ODEs are approximated by the use of the midpoint rule

    G=6.6743*10^(-11);
    % gravitational constant

    q1=zeros(3,N+1);
    p1=zeros(3,N+1);
    q2=zeros(3,N+1);
    p2=zeros(3,N+1);
    q3=zeros(3,N+1);
    p3=zeros(3,N+1);
    % preallocating

    q1(1:3,1)=q01;
    p1(1:3,1)=p01;
    q2(1:3,1)=q02;
    p2(1:3,1)=p02;
    q3(1:3,1)=q03;
    p3(1:3,1)=p03;
    % take over the initial values

    for i=1:N
       s1=p1(1:3,i);
       s2=p2(1:3,i);
       s3=p3(1:3,i);
       t1=q1(1:3,i);
       t2=q2(1:3,i);
       t3=q3(1:3,i);
       % start values for fixed point iteration

       for k=1:100
           T1=q1(1:3,i)+h/2*(s1+p1(1:3,i));
           T2=q2(1:3,i)+h/2*(s2+p2(1:3,i));
           T3=q3(1:3,i)+h/2*(s3+p3(1:3,i));
           a=t1+q1(1:3,i)-t2-q2(1:3,i);
           b=t1+q1(1:3,i)-t3-q3(1:3,i);
           c=t2+q2(1:3,i)-t3-q3(1:3,i);
           S1=p1(1:3,i)-4*h*G*(m2*a/(a'*a)^(3/2)+m3*b/(b'*b)^(3/2));
           S2=p2(1:3,i)-4*h*G*(m3*c/(c'*c)^(3/2)-m1*a/(a'*a)^(3/2));
```

```
45              S3=p3(1:3,i)+4*h*G*(m1*b/(b'*b)^(3/2)+m2*c/(c'*c)^(3/2));
46              s1=S1;
47              s2=S2;
48              s3=S3;
49              t1=T1;
50              t2=T2;
51              t3=T3;
52          end
53          % fixed point iteration
54
55          p1(1:3, i+1)=s1;
56          p2(1:3, i+1)=s2;
57          p3(1:3, i+1)=s3;
58          q1(1:3, i+1)=t1;
59          q2(1:3, i+1)=t2;
60          q3(1:3, i+1)=t3;
61          % next iterations
62
63      end
64
65      hold on
66      plot3(q1 (1,:), q1 (2,:), q1 (3,:), 'c' );
67      plot3(q2 (1,:), q2 (2,:), q2 (3,:), 'b' );
68      plot3(q3 (1,:), q3 (2,:), q3 (3,:), 'g' );
69      hold off
70
71  end
```

F. Modeling the solar system

```
function SolarSystem
% computes and plots the trajectories of the planets and sun of our solar
% system;
% the ODE system is solved with the midpoint scheme and the resulting
% implicit equation system is solved with the fixed point iteration (100
% iterations)

  N=50000;
  % number of iterations
  h=240000;
  % step size

  b=9;
  % number of bodies (sun + 8 planets)

  G=6.6743*10^(-11);
  % gravitational constant

  q=zeros(b,3,N+1);
  p=zeros(b,3,N+1);
  m=zeros(1,b);
  % preallocating for trajectories (q), velocity (p) and mass (m)
  % dimension: bodies - vector - iterations

  q(1,:,1)=5.585*10^8*[1,1,1];
  p(1,:,1)=[-1.4663,11.1238,4.8370];
  m(1)=1.99*10^30;
  % sun
  q(2,:,1)=[5.1979*10^10,7.6928*10^9,-1.2845*10^9];
  p(2,:,1)=[-1.5205,4.4189,2.518]*10^4;
  m(2)=3.3*10^23;
  % mercury
  q(3,:,1)=[-1.5041,9.708,4.4635]*10^10;
  p(3,:,1)=[-3.477*10^4,-5.5933*10^3,-316.8994];
  m(3)=4.87*10^24;
  % venus
  q(4,:,1)=[-1.1506*10^9,-1.391*10^11,-6.033*10^10];
  p(4,:,1)=[2.9288*10^4,-398.5759,-172.5873];
  m(4)=5.97*10^24;
  % earth
  q(5,:,1)=[-4.8883*10^10,-1.9686*10^11,-8.8994*10^10];
  p(5,:,1)=[2.4533*10^4,-2.7622*10^3,-1.9295*10^3];
  m(5)=6.42*10^23;
  % mars
```

```
45    q(6,:,1)=[-8.1142*10^11,4.5462*10^10,3.9229*10^10];
46    p(6,:,1)=[-1.0724*10^3,-1.1422*10^4,-4.8696*10^3];
47    m(6)=1.9*10^27;
48    % jupiter
49    q(7,:,1)=[-4.278*10^11,-1.3353*10^12,-5.3311*10^11];
50    p(7,:,1)=[8.7288,-2.4369,-1.3824]*10^3;
51    m(7)=5.68*10^26;
52    % saturn
53    q(8,:,1)=[2.7878*10^12,9.9509*10^11,3.9639*10^8];
54    p(8,:,1)=[-2.4913,5.5197,2.4527]*10^3;
55    m(8)=8.68*10^25;
56    % uranus
57    q(9,:,1)=[4.2097*10^12,-1.3834*10^12,-6.7105*10^11];
58    p (9,:,1)=[1.8271,4.7731,1.9082]*10^3;
59    m(9)=1.02*10^26;
60    % neptune
61    % initial  data and mass
62
63    M1=(ones(b,1)*m)';
64    M2=M1(eye(b)~=1);
65    M3=reshape(M2,b-1,b);
66    M=zeros(b-1,3,b);
67    M(:,1,:)=M3;
68    M(:,2,:)=M3;
69    M(:,3,:)=M3;
70    % auxiliary matrix for mass (diagonal eliminated) for later computations
71
72    H=zeros(b,3,b);
73    H(:,1,:)=eye(b);
74    H(:,2,:)=eye(b);
75    H(:,3,:)=eye(b);
76    % auxiliary matrix for later computations (elimination of diagonal)
77
78    S45=zeros(b-1,3,b);
79    % preallocating for a matrix used later on
80
81    for i=1:N
82      s=p(:,:,i);
83      t=q(:,:,i);
84      % start values for fixed point iteration
85
86        for k=1:100
87          T=q(:,:,i)+h/2*(s+p(:,:,i));
88          % next iteration for q(:,:,i+1)
89
90          % next: computing the next iteration for p(:,:,i+1)
```

```
91          S1=(t+q(:,:,i )).*ones(b,3,b);
92          % auxiliary matrix: entry :,:,x is equal to t+q(:,:,i) for every x
93          S2=permute(S1,[3 2 1]);
94          % auxiliary matrix: entry :,:,x now has in each row the same entry,
95          % (namely t(x,:)+q(x,:,i))
96          S3=S1-S2;
97          % this matrix contains the numerator without prefactor (mass) of
98          % every summand of the 9-body formula (plus zero on its diagonal)
99          S4=reshape(S3(H~=1),b-1,3,b);
100         % eliminating the zeros on its diagonal (the summand for the index
101         % coinciding with the considered body doesn't occur in the formula)
102         % next: computing the denominators of the summands
103         S42=permute(S4,[2 1 3]);
104         S43=vecnorm(S42);
105         % contains the wished norms for computing the denominators in
106         % form of a 1x(b-1)xb tensor
107         S44=reshape(S43,b-1,b);
108         % builds a (b-1)xb matrix with the searched norms
109         S45(:,1,:)=S44;
110         S45(:,2,:)=S44;
111         S45(:,3,:)=S44;
112         % creating the matrix with the norms for further computations
113         S5=S4./(S45.^3);
114         % matrix with all the summands without prefactor mass
115         S6=M.*S5;
116         % adding mass prefactor
117         S7=sum(S6);
118         % summing of; result is 1x3xb tensor: contains wished part of the
119         % midpoint scheme of every body
120         S8=reshape(S7,3,b)';
121         % transforming to bx3 matrix for further computations
122         S=p(:,:,i)+h*4*G*S8;
123         % next iteration for p(:,:,i+1)
124
125         t=T;
126         s=S;
127         % update
128
129       end
130       % fixed point iteration
131
132       q(:,:,i+1)=t;
133       p(:,:,i+1)=s;
134       % next iterations
135
136   end
```

```
137
138     hold on
139     plot3(reshape(q (1,1,:),1, N+1),reshape(q(1,2,:),1,N+1),
140           reshape(q (1,3,:),1, N+1),'y');
141     plot3(reshape(q(2,1,1:1000),1,1000),reshape(q(2,2,1:1000),1,1000),
142           reshape(q(2,3,1:1000),1,1000),'k');
143     plot3(reshape(q(3,1,1:5000),1,5000),reshape(q(3,2,1:5000),1,5000),
144           reshape(q(3,3,1:5000),1,5000),'m');
145     plot3(reshape(q(4,1,1:10000),1,10000),reshape(q(4,2,1:10000),1,10000),
146           reshape(q(4,3,1:10000),1,10000),'b');
147     plot3(reshape(q(5,1,1:15000),1,15000),reshape(q(5,2,1:15000),1,15000),
148           reshape(q(5,3,1:15000),1,15000),'r');
149     plot3(reshape(q(6,1,1:20000),1,20000),reshape(q(6,2,1:20000),1,20000),
150           reshape(q(6,3,1:20000),1,20000),'m');
151     plot3(reshape(q(7,1,1:25000),1,25000),reshape(q(7,2,1:25000),1,25000),
152           reshape(q(7,3,1:25000),1,25000),'y');
153     plot3(reshape(q(8,1,1:35000),1,35000),reshape(q(8,2,1:35000),1,35000),
154           reshape(q(8,3,1:35000),1,35000),'g');
155     plot3(reshape(q (9,1,:),1, N+1),reshape(q(9,2,:),1,N+1),
156           reshape(q (9,3,:),1, N+1),'c');
157     legend('Sun','Mercury','Venus','Earth','Mars','Jupiter','Saturn','Uranus',
158           'Neptune','Location','eastoutside');
159     hold off
160     view(-37.5,30)
161
162 end
```