Exercises for the lecture

Fundamentals of Simulation Methods

WS 2016/17

Lecturers: Frauke Gräter and Rüdiger Pakmor

Exercise sheet 2 ***(due date: Nov 3, 2016, 11:59pm)***

Integration of ordinary differential equations

1) Order of an ODE integration scheme (10 points)

Consider the differential equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y) \tag{1}$$

for the function $y(t)$ and a general right hand side $f(y)$. This may be integrated discretely with a Runge-Kutta scheme of the form:

$$\begin{aligned} k_1 &= f(y_n), &(2)\\ k_2 &= f(y_n + k_1 \Delta t), &(3)\\ y_{n+1} &= y_n + \frac{1}{2}(k_1 + k_2)\,\Delta t. &(4) \end{aligned}$$

Show that the truncation error per step is of third-order in the step-size Δt, or in other words, that the scheme is second-order accurate for the global integration error.

Hint: Consider a Taylor expansion of the difference $y_{n+1} - y(t_n + \Delta t)$ and assume that at the beginning of the step one starts out with the exact solution $y_n = y(t_n)$.

2) Integration of a stiff equation (20 points)

Consider an ionized plasma of hydrogen gas that radiatively cools. Its temperature evolution is governed by the equation

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -\frac{2}{3\,k_{\mathrm{B}}} n_{\mathrm{H}}\,\Lambda(T) \tag{5}$$

where $\Lambda(T)$ describes the cooling rate as a function of temperature, $k_{\mathrm{B}} = 1.38 \times 10^{-23}$ J/K is Boltzmann's constant, and n_{H} is the number density of hydrogen atoms. The cooling rate is a strong function of temperature T, which we here approximate by

$$\Lambda(T) = \begin{cases} \Lambda_0 \left(\frac{T}{T_0}\right)^{\alpha} & \text{for } T \le T_0 \\ \\ \Lambda_0 \left(\frac{T}{T_0}\right)^{\beta} & \text{for } T > T_0 \end{cases} \tag{6}$$

with $\Lambda_0 = 10^{-35}\,\mathrm{J\,m^3\,s^{-1}}$, $T_0 = 20000\,\mathrm{K}$, $\alpha = 10.0$, and $\beta = -0.5$. We consider isochoric cooling of gas at density $n_{\mathrm{H}} = 10^6\,\mathrm{m^{-3}}$, with an initial temperature of $T_{\mathrm{init}} = 10^7\,\mathrm{K}$.

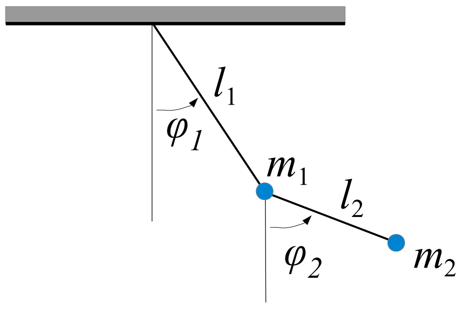
(a) Determine the temperature evolution $T(t)$ by integrating equation (1) with a second-order explicit RK predictor-corrector scheme and a fixed timestep, until the temperature has dropped below 6000 K. Use a timestep size of $\Delta t = 10^{10}$ sec. Make a plot of the time evolution of the temperature, with a logarithmic scale for temperature and a linear scale for the time.

(b) How many steps do you roughly need in (a) to reach the final temperature? Try to play with the timestep size and see whether you can significantly enlarge the timestep without becoming unstable.

(c) Now implement the second-order integration from (a) with an adaptive step size control, based on estimating the local truncation error by carrying out two half-steps for every step. Use an absolute local error limit $\Delta T_{\rm err}^{\rm max} = 50\,$K for every step. Overplot your result for the temperature evolution, on the plot for (a), using symbols or a different color. How many steps do you now need? Confirm that your scheme is robust to large changes of the timestep size given as input for the first step.

3) Double pendulum (20 points)

We consider a friction-less double pendulum that is constrained to move in one plane. The two masses m_1 and m_2 are connected via massless rods of length l_1 and l_2, respectively, as depicted in the sketch.



The Lagrangian of this system is given by the expression

$$L = \frac{m_1}{2}(l_1\dot{\phi}_1)^2 + \frac{m_2}{2}\left[(l_1\dot{\phi}_1)^2 + (l_2\dot{\phi}_2)^2 + 2l_1l_2\dot{\phi}_1\dot{\phi}_2\cos(\phi_1-\phi_2)\right]$$
$$-m_1 g\, l_1(1-\cos\phi_1) - m_2 g\left[l_1(1-\cos\phi_1) + l_2(1-\cos\phi_2)\right] \quad (7)$$

(a) Derive the Lagrangian equations of motion,

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial\dot{\phi}} - \frac{\partial L}{\partial\phi} = 0, \quad (8)$$

for the angles ϕ_1 and ϕ_2. Hint: Declare conjugate momenta $q \equiv \frac{\partial L}{\partial\dot{\phi}}$ and *do not* explicitly carry out the absolute time derivative; it is sufficient if you give $\frac{\mathrm{d}q_1}{\mathrm{d}t}$ and $\frac{\mathrm{d}q_2}{\mathrm{d}t}$.

(b) Cast the system of equations into 1st-order form, such that the dynamics is described by the ODE

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}), \quad (9)$$

where $\mathbf{y}$ is a four-component vector. Hint: Use the conjugate momenta to eliminate the second derivatives, i.e. adopt $\mathbf{y} = (\phi_1, \phi_2, q_1, q_2)$ as state vector. Hint 2: When you define f_3, f_4, you can save time/effort if you "re-use" the values of f_1, f_2, no need to plug in their expressions again. You should do so when you are writing the program as well.

(c) Write a computer program that integrates the system with a second-order predictor-corrector Runge-Kutta scheme. Consider the initial conditions $\phi_1 = 50^{\rm o}$, $\phi_2 = -120^{\rm o}$, $\dot\phi_1 = \dot\phi_2 = 0$, and adopt $m_1 = 0.5$, $m_2 = 1.0$, $l_1 = 2.0$, and $l_2 = 1.0$. For simplicity, we shall use units where $g = 1$. Use a fixed timestep of size $\Delta t = 0.05$, and integrate for the period $T = 100.0$ time units (equivalent to 2000 steps). Plot the relative energy error, $(E_{\rm tot}(t) - E_{\rm tot}(t_0))/E_{\rm tot}(t_0)$, as a function of time.

(d) Produce a second version of your code that uses a fourth-order Runge-Kutta scheme instead. Repeat the simulation from (c) with the same timestep size, and again plot the energy error. How does the size of the error at the end compare, and is this consistent with your expectations?

(e) Let's make a visualization of our double pendulum in order to get a feel for its interesting and quite complex behavior. In fact, this pendulum is one of the simplest systems that shows non-linear chaotic behaviour. We would like to end up with a movie file if possible, so this part of the exercise is also meant to guide you through the steps that are necessary for this. But you may also hand in a sequence of still images if you prefer.

A standard method to make a digital movie is to produce a stack of images equally spaced in time, and then to encode them into a heavily compressed digital video stream. Suppose you have produced such images, named `pic_000.jpg`, `pic_001.jpg`, `pic_002.jpg`, ..., etc., perhaps the simplest method to make a movie file from them is to encode them with the `ffmpeg` program. A possible command for this is

```
ffmpeg -r 24 -i pic_%03d.jpg movie.mp4
```

which will make use of a high-quality MPEG-4 compression scheme and a frame rate of 24 images per second. Numerous alternative programs for this exist, including `mencoder` and others.

To produce the images you can for example use the Python template `plot.py` (that makes images based on "fake data": $\phi_1(t) = \sin(0.1 \cdot t)$, $\phi_2(t) = \cos(0.1 \cdot t)$) that is provided on Moodle and combine it (i.e., the function `frame(...)`) with your pendulum simulation code. For a nice result, you may want to plot besides the current position of the pendulum the track of all past positions of the masses, as shown in the Python example.

Fundamentals of Simulation methods
Solution Sheet 2

Bhavya Joshi & Florian Jörg

November 3, 2016

Exercise 1:

Looking at the difference of the approximated solution and the exact solution for y at time step $t+1$:

$$\begin{aligned}\epsilon &= y_{t+1} - y(t_n + \Delta t) \\ &= y_n + \frac{1}{2}(k_1 + k_2)\Delta t - \left(y(t_n) + \int_{t_n}^{t_n+\Delta t} dt f(y(t)) \right)\end{aligned}$$

Assuming that the solution for time step t_n was exactly correct ($y_n = y(t_n)$) this simplifies to

$$\epsilon = \frac{1}{2}(k_1 + k_2)\Delta t - \left(\int_{t_n}^{t_n+\Delta t} dt f(y(t)) \right)$$

this needs to be somehow Taylor expanded... How?!

Exercise 2.

a)

The Integration of the equation

$$\frac{dT}{dt} = -\frac{2}{3k_B} n_H \Lambda(T)$$

is performed by a second order Runge-Kutta scheme with the script `ex2_1_a.cc`. The resulting values for the time and the temperature have been exported to a text file and then plotted with matplotlib using the script contained in `ex1_plots.ipynb`. The result is shown in figure 1.

b)

The solution in a) was performed using $\Delta t = 10^{10}\ s$ for the time steps. This results in roughly 54000 data points. We now want to see if the size of the time steps has an influence on the solution's stability. It can be seen from the plots in figure 2, that almost a factor of 10 in the reduction of the time step width is possible without the simulation behaving weired. For $\Delta t = 9.9 \times 10^{11} s$ one can see that the result features obvious deviations from the solution of a). For $\Delta t = 1 \times 10^{12}$ the temperature even becomes neagative.

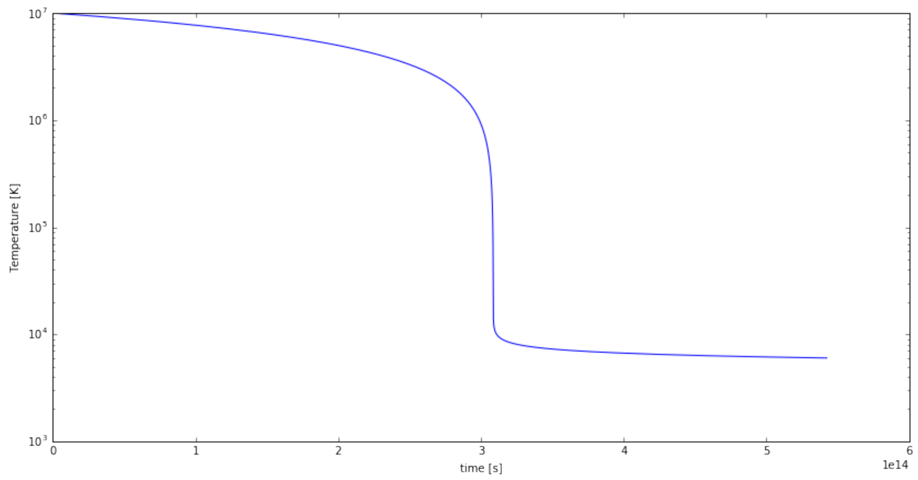


Figure 1: Solution of the runge kutta method. With $n = 54226$ data points.

c)

Now we need to implement an adaptive step size, that will automatically adjust the size of Δt) such that locally an error of $\Delta T_{err}^{max} = 50\ K$ won't be exeeded. This is achieved by the following algorithm:

1. Execute the normal Runge-Kutta integration step as in a) using the proposed step size Δt_A to get from the current value y_0 to the next value y_A.
2. Execute two RK steps using $\Delta t_B = \Delta t_A/2$. This yields $y_0 \to y' \to y_B$.
3. The difference between the two results gives an estimate for the error using the step size Δt_A. $\epsilon = |y_A - y_B|$.
4. if the achieved error is bigger than the permitted local error ϵ_{err}^{max}, do the same again using again half the step size.
5. If the error now is smaller than the permitted error, keep that result and use the new step size also in the next step.
6. if the error is much smaller $\epsilon_{err}^{max} >> \epsilon$, the step size should be increased in order to save computation cost. An estimate for "much smaller" is $epsilon_{err}^{max} < \epsilon \cdot 2^{p+1}$ where p is the order of the used RK scheme.

Running the algorithm is much faster now. For an initial step size of $\Delta_t^0 = 10^{10}\ s$ the algorithm yields 75 data points. The result however is very close to the one obtained in part a) as can be seen in figure 3 (top)!

Looking at the second plot (figure 3 (bottom)) one can see, how the size of the time step is automatically first increased to save computation time and in the area, where the function changes rapidly, the step size is reduced by many orders of magnitude. After it reaches the flat bottom, the step size increases again. In this plot one can also see, how different initial values for Δt all lead to the same following values for the time steps. This shows that this method is robust in this case.

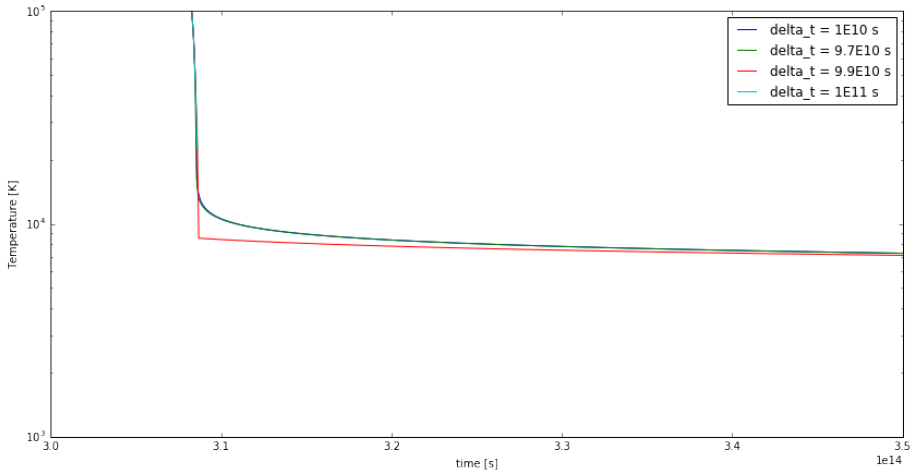


Figure 2: Solution of the runge kutta method using different values for Δt. The figure shows the part of the plot where the deviation of the single solutions is most obvious.

Exercise 3:

a)

The equations of motion follow from the Euler Lagrange equations.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\phi_i}} = \frac{\partial L}{\partial \phi_i} \tag{1}$$

Lets first compute the conjugate momenta $q_1 \equiv \partial L/\partial \dot{\phi_1}$ and $q_2 \equiv \partial L/\partial \dot{\phi_2}$.

$$q_1 = \frac{\partial L}{\partial \dot{\phi_1}} = m_1 l_1^2 \dot{\phi_1} + m_2 l_1^2 \dot{\phi_1} + m_2 l_1 l_2 \dot{\phi_2}\cos(\phi_1 - \phi_2) \tag{2}$$

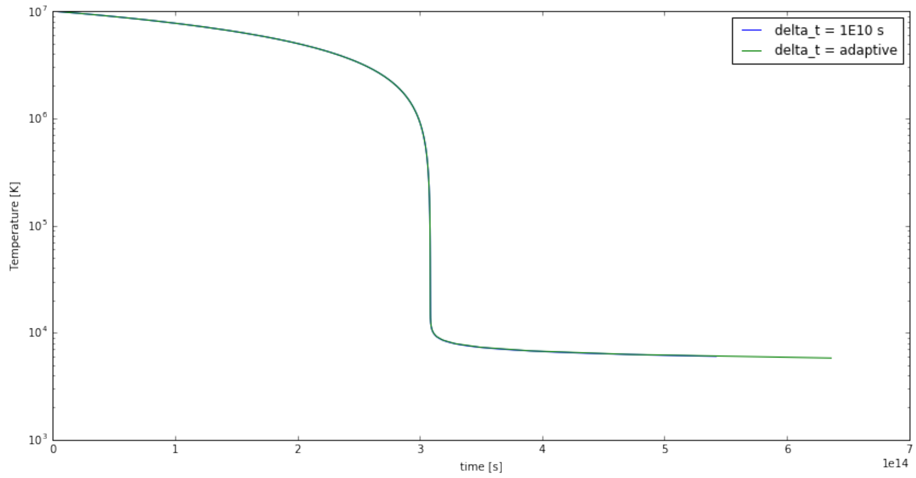
$$q_2 = \frac{\partial L}{\partial \dot{\phi_2}} = m_2 l_2^2 \dot{\phi_2} + m_2 l_1 l_2 \dot{\phi_1}\cos(\phi_1 - \phi_2) \tag{3}$$

The right hand side of equation 1 reads and must be equal to the left hand side i.e. the total time derivative of the conjugate momenta.

$$\frac{\partial L}{\partial \phi_1} = -m_2 l_1 l_2 \dot{\phi_1}\dot{\phi_2}\sin(\phi_1 - \phi_2) - l_1 g \sin(\phi_1)(m_1 + m_2) = \dot{q}_1 \tag{4}$$

$$\frac{\partial L}{\partial \phi_2} = m_2 l_1 l_2 \dot{\phi_1}\dot{\phi_2}\sin(\phi_1 - \phi_2) - l_2 g \sin(\phi_2) m_2 = \dot{q}_2 \tag{5}$$

These results are checked with Wolfram :).



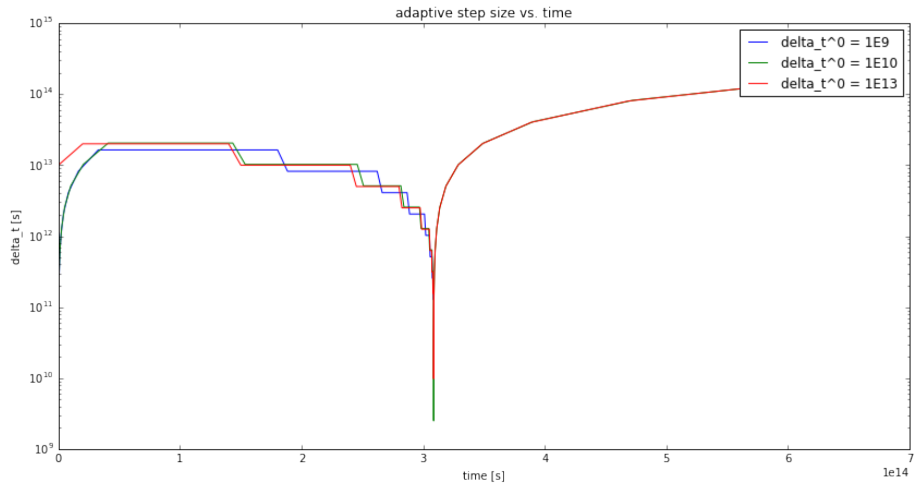


Figure 3: Results for the Runge Kutta method using adaptive step refinement. Top: comparison with the result from a). Bottom: adaptive step size vs. time using three different values for Δt_0.

b)

Now we define the state vector $\mathbf{y} \equiv (\phi_1, \phi_2, q_1, q_2)$. Using this definition, the Euler-Lagrange equation (1) can be written in the following form:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y})$$
$$\frac{d}{dt}\begin{pmatrix}\phi_1\\ \phi_2\\ q_1\\ q_2\end{pmatrix} = \begin{pmatrix}\dot{\phi_1}\\ \dot{\phi_2}\\ \dot{q_1}\\ \dot{q_2}\end{pmatrix} \tag{6}$$

Solving equation (2) for $\dot{\phi}_1$

$$\begin{aligned} q_1 &= (m_1 l_1^2 + m_2 l_1^2)\dot{\phi}_1 + m_2 l_1 l_2 \dot{\phi}_2 \cos(\phi_1 - \phi_2) \\ \Rightarrow \qquad \dot{\phi}_1 &= \frac{q_1 - m_2 l_1 l_2 \dot{\phi}_2 \cos(\phi_1 - \phi_2)}{(m_1 l_1^2 + m_2 l_1^2)} = f_1 \end{aligned} \tag{7}$$

and equation (3) for $\dot{\phi}_2$

$$\begin{aligned} q_2 &= m_2 l_2^2 \dot{\phi}_2 + m_2 l_1 l_2 \dot{\phi}_1 \cos(\phi_1 - \phi_2) \\ \Rightarrow \qquad \dot{\phi}_2 &= \frac{q_2 - m_2 l_1 l_2 \dot{\phi}_1 \cos(\phi_1 - \phi_2)}{m_2 l_2^2} = f_2 \end{aligned} \tag{8}$$

We now use these results to eliminate the appearance of $\dot{\phi}_2$ in eq(7) and of $\dot{\phi}_1$ in eq(8) by inserting them.

$$\begin{aligned} q_1 &= (m_1 l_1^2 + m_2 l_1^2)\dot{\phi}_1 + m_2 l_1 l_2 \cos(\phi_1 - \phi_2) \cdot \frac{q_2 - m_2 l_1 l_2 \dot{\phi}_1 \cos(\phi_1 - \phi_2)}{m_2 l_2^2} \\ &= (m_1 l_1^2 + m_2 l_1^2)\dot{\phi}_1 - l_1^2 m_2 \dot{\phi}_1 \cos^2(\phi_1 - \phi_2) + \frac{l_1 \cos(\phi_1 - \phi_2) q_2}{l_2} \\ &= \dot{\phi}_1 \left(m_1 l_1^2 + m_2 l_1^2 - l_1^2 m_2 \cos^2(\phi_1 - \phi_2)\right) + \frac{l_1 \cos(\phi_1 - \phi_2) q_2}{l_2} \\ \Rightarrow \qquad \dot{\phi}_1 &= \frac{q_1 l_2 - l_1 q_2 \cos(\phi_1 - \phi_2)}{l_2(m_1 l_1^2 + m_2 l_1^2 - l_1^2 m_2 \cos^2(\phi_1 - \phi_2))} \end{aligned} \tag{9}$$

And for the expression for $\dot{\phi}_2$ follows:

$$\begin{aligned} q_2 &= \dot{\phi}_2 \left(m_2 l_2 - \frac{m_2^2 l_1^2 l_2^2 \cos^2(\phi_1 - \phi_2)}{m_1 l_1^2 + m_2 l_1^2}\right) + \frac{q_1 m_2 l_1 l_2 \cos(\phi_1 - \phi_2)}{m_1 l_1^2 + m_2 l_1^2} \\ \Rightarrow \qquad \dot{\phi}_2 &= \frac{q_2(m_1 l_1^2 + m_2 l_1^2) - q_1 m_2 l_1 l_2 \cos(\phi_1 - \phi_2)}{m_2 l_2 (m_1 l_1^2 + m_2 l_1^2) - m_2^2 l_1^2 l_2^2 \cos^2(\phi_1 - \phi_2)} \end{aligned} \tag{10}$$

Now f_1 and f_2 only depend on quantities present in the state vector $\mathbf{y}$ which describes the state of the system. Equations (4) and (5) are the right hand sides of the total time derivatives of q_1 and q_2. If the obtained results for $\dot{\phi}_1$ and $\dot{\phi}_2$ are inserted there, the system can be solved using the RK-scheme.

$$\begin{aligned} \frac{d}{dt}\mathbf{y} = \frac{d}{dt}\begin{pmatrix} \phi_1 \\ \phi_2 \\ q_1 \\ q_2 \end{pmatrix} &= \begin{pmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{pmatrix} \\ &= \begin{pmatrix} \frac{q_1 l_2 - l_1 q_2 \cos(\phi_1 - \phi_2)}{l_2(m_1 l_1^2 + m_2 l_1^2 - l_1^2 m_2 \cos^2(\phi_1 - \phi_2))} \\ \frac{q_2(m_1 l_1^2 + m_2 l_1^2) - q_1 m_2 l_1 l_2 \cos(\phi_1 - \phi_2)}{m_2 l_2(m_1 l_1^2 + m_2 l_1^2) - m_2^2 l_1^2 l_2^2 \cos^2(\phi_1 - \phi_2)} \\ -m_2 l_1 l_2 f_1 f_2 \sin(\phi_1 - \phi_2) - l_1 g \sin(\phi_1)(m_1 + m_2) \\ m_2 l_1 l_2 f_1 f_2 \sin(\phi_1 - \phi_2) - l_2 g \sin(\phi_2) m_2 \end{pmatrix} = \mathbf{f}(\mathbf{y}) \end{aligned} \tag{11}$$

d)

The implementation of the Runge Kutta scheme to solve equation (11) is found in files `ex2_3_c.cc` and `ex2_3_d.cc`. The relative energy difference between the initial state and the system after time t is shown for both methods in figure 4. As it can be seen, the relative error obtained using the fourth order RK scheme is very much smaller than the one obtained using the second order scheme. In terms of computational cost there was no obvious difference but detailed measurement would probably show the higher computational effort needed for the fourth order implementation. Figure 5 shows a still image of the simulated system towards the end of the simulation.

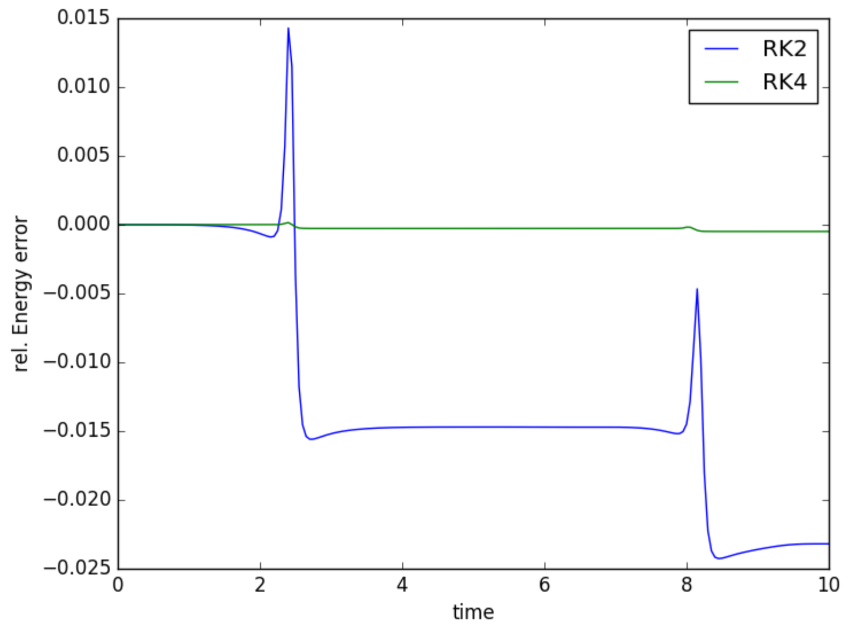


Figure 4: Comparison for the Runge Kutta method using second order (blue) and fourth order (green) integration.

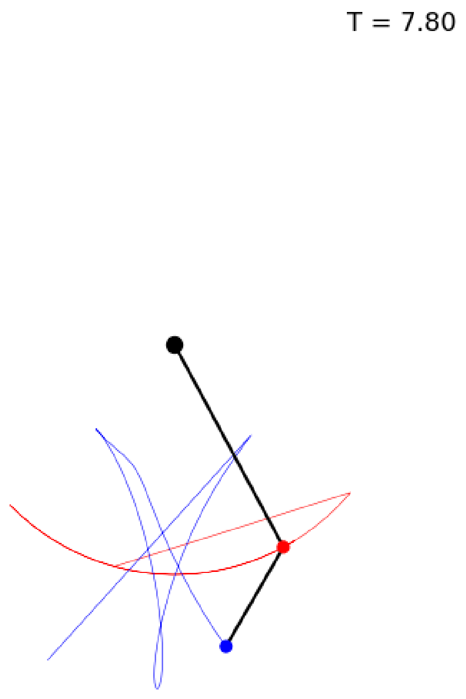


Figure 5: Still image of the simulated system.