Fundamentals of Simulation Methods

Exercise Sheet 2

Daniel Rosenblüh, Janosh Riebesell

October 6th, 2015

Integration of ordinary differential equations

1 Order of an ODE integration scheme

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

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

Show that the truncation error per step $\mathcal{O}_s$ is of third-order in the step-size Δt, i.e. $\mathcal{O}_s(\Delta t^3)$, or in other words, that the scheme is second-order accurate for the global integration error, $\mathcal{O}_T(\Delta t^2)$.
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 $y_n = y(t_n)$.

Note: Equation (2) is the second-order Runge-Kutta method. It is a modification of the Euler method

$$y_{n+1} = y_n + f(y_n)\Delta t, \tag{3}$$

which is the simplest possible integrator that yields an approximate solution to an ordinary differential equation such as eq. (1). It is not particularly stable, nor accurate compared to more sophisticated methods but serves the purpose of exemplifying the basic concept of solving ODEs by discretization: We rewrite the differentials $\mathrm{d}y$ and $\mathrm{d}t$ as finite steps Δy and Δt. We then multiply the ODE by Δt. This yields the algebraic formula in eq. (3) for the change in the function y_n when t is increased by one stepsize Δt.
The Euler method is primitive in the sense that it advances the discretized solution y_n through the interval Δt but uses only information about $f(y_n)$ at the beginning of the interval. This is where Runga-Kutta becomes more advanced.

The Runge-Kutta method takes a *trial step* to the middle of the interval, samples the function $f[y_n + f(y_n) \cdot \Delta t]$ there as well, and then averages it with $f(y_n)$ to compute the change in y_n. This symmetrization precisely cancels out both the first- and second-order error terms. Since a method is called n-th order accurate if its local error introduced with every step appears with a power no lower than $n+1$, this makes Runge-Kutta second-order accurate.

To see this, we expand k_2,

$$\begin{aligned} k_2 &= f(y_n + k_1\Delta t) \\ &= f(y_n) + f'(y_n)k_1\Delta t + \frac{1}{2}f''(y_n)(k_1\Delta t)^2 + \mathcal{O}_s(\Delta t^3). \end{aligned} \tag{4}$$

Inserting this into y_{n+1} as given in eq. (2) and using $k_1 = f(y_n)$ yields

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{2}(k_1 + k_2)\Delta t \\ &= y_n + f(y_n)\Delta t + \frac{1}{2}f'(y_n)f(y_n)\Delta t^2 + \frac{1}{4}f''(y_n)[f(y_n)]^2\Delta t^3 + \mathcal{O}_s(\Delta t^4). \end{aligned} \tag{5}$$

By contrast, when expanding $y_{n+1} = y(t_{n+1}) = y(t_n + \Delta t)$ directly, we obtain[1]

$$\begin{aligned} y(t_n + \Delta t) &= y(t_n) + \dot{y}(t_n)\Delta t + \frac{1}{2}\ddot{y}(t_n)\Delta t^2 + \frac{1}{6}\dddot{y}(t_n)\Delta t^3 + \mathcal{O}_s(\Delta t^4) \\ &= y_n + f(y_n)\Delta t + \frac{1}{2}\left[\frac{\mathrm{d}}{\mathrm{d}t}f(y_n)\right]\Delta t^2 + \frac{1}{6}\left[\frac{\mathrm{d}^2}{\mathrm{d}^2 t}f(y_n)\right]\Delta t^3 + \mathcal{O}_s(\Delta t^4) \\ &= y_n + f(y_n)\Delta t + \frac{1}{2}f'(y_n)f(y_n)\Delta t^2 \\ &\quad + \frac{1}{6}\Big[f''(y_n)[f(y_n)]^2 + [f'(y_n)]^2 f(y_n)\Big]\Delta t^3 + \mathcal{O}_s(\Delta t^4). \end{aligned} \tag{6}$$

In eq. (6), we used the original ODE in between the first and second line and executed the chain rule to arrive at the third.

Now, taking the difference of y_{n+1} with the exact solution $y(t+\Delta t)$, we see that the terms linear and quadratic in Δt cancel, and the Runge-Kutta error e_{RK} is of order three:

$$\begin{aligned} e_{\mathrm{RK}} &= y_{n+1} - y(t_n + \Delta t) \\ &= \frac{1}{12}\left[f''(y_n)[f(y_n)]^2 - 2[f'(y_n)]^2 f(y_n)\right]\Delta t^3 + \mathcal{O}_s(\Delta t^4). \end{aligned} \tag{7}$$

Note: We can carry this scheme even further by adding more and more evaluations of $f(y)$ into each step. Adding up the right combinations of higher-order terms thus obtained, we can eliminate the error terms order by order. This is the fundamental idea of the Runga-Kutta method. By far in most common use is the *fourth-order* Runge-Kutta formula, because order $n = 4$ is the highest order that only requires n function evaluations. Methods of order $n \geq 5$ require *more than* n evaluations.

2 Integration of a stiff equation

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

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -\frac{2}{3k_{\mathrm{B}}}n_{\mathrm{H}}\Lambda(T), \tag{8}$$

where $\Lambda(T)$ describes the cooling rate as a function of temperature, $k_{\mathrm{B}} = 1.38 \times 10^{-16}\,\mathrm{erg\,K^{-1}}$ is Boltzmann's constant, and n_{H} is the number density of hydrogen atoms. The cooling rate

[1]As always, dots indicate derivatives w.r.t. time. Derivatives w.r.t. to $y(t)$ are marked by primes.

is a strong function of temperature T, which we here approximate by

$$\Lambda(T) = \begin{cases} \Lambda_0 \left(\frac{T}{T_0}\right)^{\alpha}, & \text{for } T \leq T_0 \\ \Lambda_0 \left(\frac{T}{T_0}\right)^{\beta}, & \text{for } T > T_0 \end{cases}, \tag{9}$$

with $\Lambda_0 = 10^{-22}\,\mathrm{erg\,cm^3\,s^{-1}}$, $T_0 = 20\,000\,\mathrm{K}$, $\alpha = 10.0$, and $\beta = -0.5$. We consider isochoric cooling of gas at density $n_\mathrm{H} = 1.0\,\mathrm{cm^{-3}}$, with an initial temperature of $T_\mathrm{i} = 10^7\,\mathrm{K}$.

(a) Determine the temperature evolution $T(t)$ by integrating eq. (8) with a second-order explicit Runge-Kutta scheme and a fixed timestep of $\Delta t = 10^{10}\,\mathrm{s}$, until the temperature has dropped below 6000 K. 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 part (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 part (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 $\Delta T_\mathrm{err}^\mathrm{max} = 50\,\mathrm{K}$ for every step. Overplot your result for the temperature evolution, on the plot for part (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.

(a) A plot of the temperature evolution of the system described by eq. (8) is shown in fig. 1.

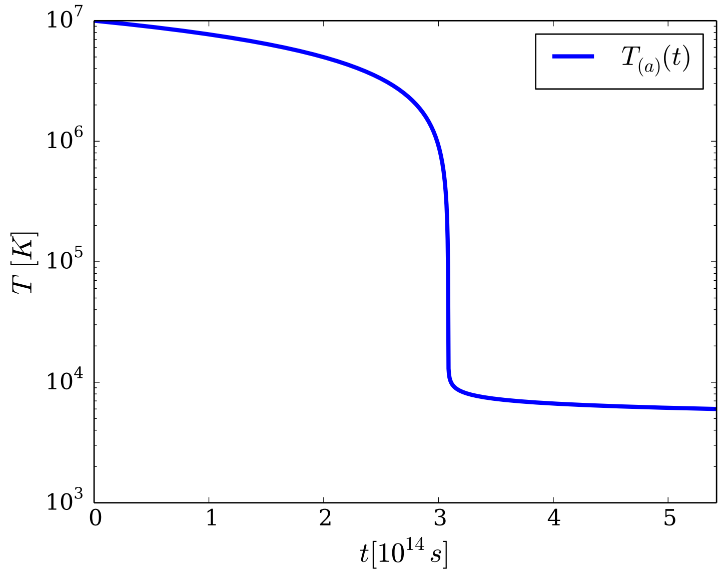
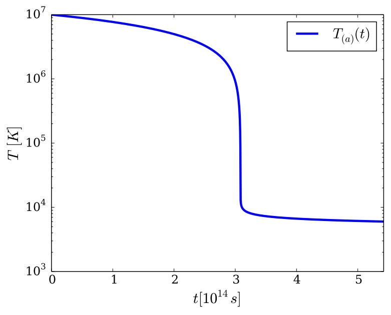


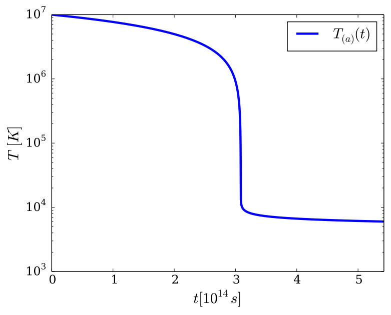
Figure 1: Second-order Runge-Kutta temperature evolution

(b) Starting from an initial temperature $T_\mathrm{i} = 10^7\,\mathrm{K}$, it took our simulation roughly 54 000 timesteps to reach the final temperature of $T_\mathrm{f} = 6000\,\mathrm{K}$.

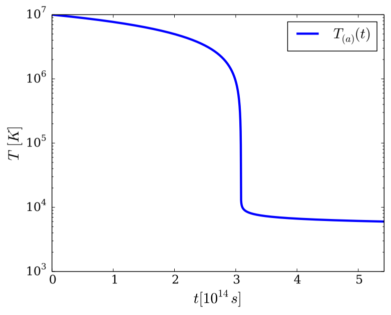
As expected, the simulation remains stable upon shrinking of the stepsize by a factor of 10 (see fig. 2a). For comparison, the original simulation is shown in fig. 2b. The output also remains invariant when increasing the stepsize by factors of 2 and 4 as shown in figs. 2c and 2d, respectively. Upon increasing Δt by a factor of 6, we introduce the first numerical artifacts (fig. 2e), and when going as far as a factor of 10, the simulation aborts prematurely, i.e. before reaching the final temperature $T_{\mathrm{f}} = 6000\,\mathrm{K}$ (fig. 2f).



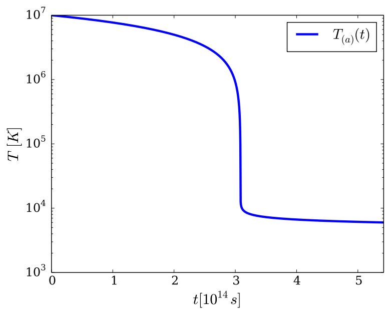
(a) $\Delta t = 10^9\,\mathrm{s}$



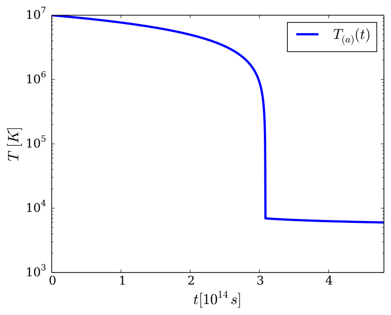
(b) $\Delta t = 10^{10}\,\mathrm{s}$



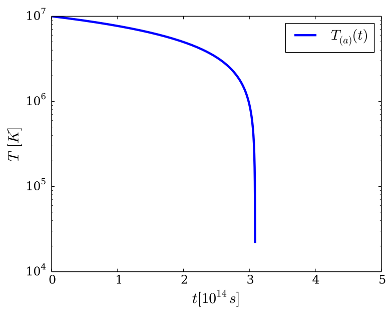
(c) $\Delta t = 2 \times 10^{10}\,\mathrm{s}$



(d) $\Delta t = 4 \times 10^{10}\,\mathrm{s}$



(e) $\Delta t = 6 \times 10^{10}\,\mathrm{s}$

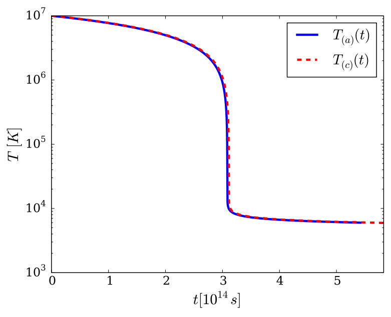


(f) $\Delta t = 10^{11}\,\mathrm{s}$

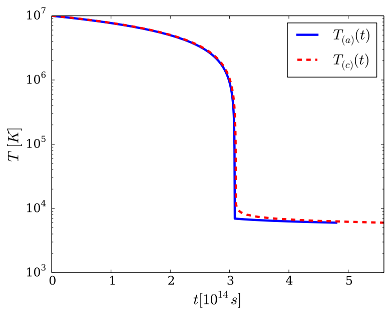
Figure 2: Temperature evolutions for different stepsizes Δt.

(c) Step doubling is a common means for adaptive stepsize control in Runge-Kutta. Comparing the result of a big step with that of two half-sized ones gives a convenient criterion for adjusting the stepsize for the next step, or for recalculating the current step with smaller stepsize if the mismatch was deemed too great.

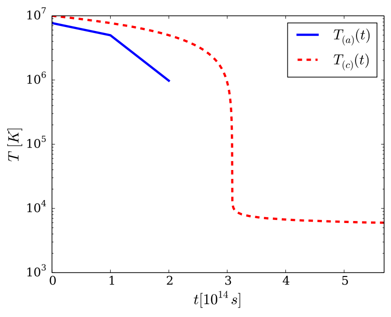
For the stepsize $\Delta t = 10^{10}\,\mathrm{s}$ as used in part (a) where a second-order Runge-Kutta implementation still showed ample stability, the temperature evolutions with and without adaptive stepsize control look identical (fig. 3a). However, increasing the stepsize to the limit of second-order stability as determined in part (b), we find in fig. 3b that the adaptive stepsize control upholds stability. This remains true even when increasing the initial stepsize by four orders of magnitude, as seen in fig. 3c.



(a) $\Delta t = 10^{10}\,\mathrm{s}$



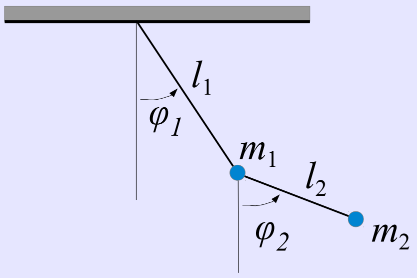
(b) $\Delta t = 6 \times 10^{10}\,\mathrm{s}$



(c) $\Delta t = 10^{14}\,\mathrm{s}$

Figure 3: Temperature evolution according to Runge-Kutta with adaptive stepsize control

3 Double pendulum



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

$$\begin{aligned} L = \frac{m_1}{2}\, l_1^2\, \dot\phi_1^2 + \frac{m_2}{2}\Big[l_1^2\,\dot\phi_1^2 + l_2^2\,\dot\phi_2^2 + 2\, l_1\, l_2\, \dot\phi_1\, \dot\phi_2\, \cos(\phi_1 - \phi_2)\Big] \\ - m_1\, g\, l_1\,(1-\cos\phi_1) - m_2\, g\Big[l_1(1-\cos\phi_1) + l_2\,(1-\cos\phi_2)\Big] \end{aligned} \tag{10}$$

(a) Derive the Lagrangian equations of motion,

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot\phi_i} - \frac{\partial L}{\partial \phi_i} = 0, \qquad i \in \{1,2\} \tag{11}$$

for the angles ϕ_1 and ϕ_2.
Hint: Declare conjugate momenta $q_i = \frac{\partial L}{\partial \dot\phi_i}$ 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 first-order form, such that the dynamics is described by the ODE

$$\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{y}) \tag{12}$$

where $\boldsymbol{y}$ is a four-component vector.
Hint: Use the conjugate momenta to eliminate the second derivatives, i.e. adopt $\boldsymbol{y} = (\phi_1, \phi_2, q_1, q_2)$ as state vector.

(c) Write a program that integrates the system with a second-order predictor-corrector Runge-Kutta scheme. Consider the initial conditions $\phi_1 = 50°$, $\phi_2 = -120°$, $\dot\phi_1 = \dot\phi_2 = 0$, 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,

$$\eta_E = \frac{E_{\mathrm{tot}}(t) - E_{\mathrm{tot}}(t_0)}{E_{\mathrm{tot}}(t_0)},$$

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 part (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, 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 -qscale 1 -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 that is provided on Moodle and combine it 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.

(a) The conjugate momenta are

$$q_1 \equiv \frac{\partial L}{\partial \dot\phi_1} = (m_1+m_2)\, l_1^2\, \dot\phi_1 + m_2\, l_1\, l_2 \dot\phi_2 \cos(\phi_1-\phi_2) \tag{13}$$

$$q_2 \equiv \frac{\partial L}{\partial \dot\phi_2} = m_2\, l_2^2\, \dot\phi_2 + m_2\, l_1\, l_2\, \dot\phi_1 \cos(\phi_1-\phi_2) \tag{14}$$

By the Euler-Lagrange equations, their time derivatives have to equal

$$\dot q_1 = \frac{\partial L}{\partial \phi_1} = -m_2\, l_1\, l_2\, \dot\phi_1\, \dot\phi_2 \sin(\phi_1-\phi_2) - (m_1+m_2)\, g\, l_1 \sin\phi_1 \tag{15}$$

$$\dot q_2 = \frac{\partial L}{\partial \phi_2} = m_2\, l_1\, l_2\, \dot\phi_1\, \dot\phi_2 \sin(\phi_1-\phi_2) - m_2\, g\, l_2 \sin\phi_2 \tag{16}$$

(b) By rewriting eqs. (13) and (14) for the conjugate momenta in a vector notation, we obtain

$$\begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} (m_1+m_2)l_1^2 & m_2 l_1 l_2 \cos(\phi_1-\phi_2) \\ m_2 l_1 l_2 \cos(\phi_1-\phi_2) & m_2 l_2^2 \end{pmatrix} \begin{pmatrix} \dot\phi_1 \\ \dot\phi_2 \end{pmatrix} \equiv A \begin{pmatrix} \dot\phi_1 \\ \dot\phi_2 \end{pmatrix}. \tag{17}$$

Since,

$$\begin{aligned} \det(A) &= (m_1+m_2)l_1^2 \cdot m_2 l_2^2 - m_2^2 l_1^2 l_2^2 \cos^2(\phi_1-\phi_2) \\ &= m_1 m_2 l_1^2 l_2^2 + m_2^2 l_1^2 l_2^2 \sin^2(\phi_1-\phi_2) \\ &= m_2 l_1^2 l_2^2 \Big(m_1 + m_2 \sin^2(\phi_1-\phi_2) \Big) \neq 0, \end{aligned} \tag{18}$$

A is invertible and we can easily find an expression for $\dot\phi_1, \dot\phi_2$ by multiplying eq. (17) from the left with A^{-1}:

$$\begin{pmatrix} \dot\phi_1 \\ \dot\phi_2 \end{pmatrix} = A^{-1} \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}, \tag{19}$$

where the inverse of A is given by

$$A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A) = \frac{1}{\det(A)} \begin{pmatrix} m_2 l_2^2 & -m_2 l_1 l_2 \cos(\phi_1-\phi_2) \\ -m_2 l_1 l_2 \cos(\phi_1-\phi_2) & (m_1+m_2) l_1^2 \end{pmatrix}. \tag{20}$$

By inserting eq. (20) into eq. (19), we can read off the canonical coordinates,

$$\dot{\phi}_1 = A^{-1}_{11} q_1 + A^{-1}_{12} q_2, \tag{21}$$
$$\dot{\phi}_2 = A^{-1}_{21} q_1 + A^{-1}_{22} q_2. \tag{22}$$

Defining $\boldsymbol{y} = (\phi_1, \phi_2, q_1, q_2)^T$, we obtain the ordinary differential equation

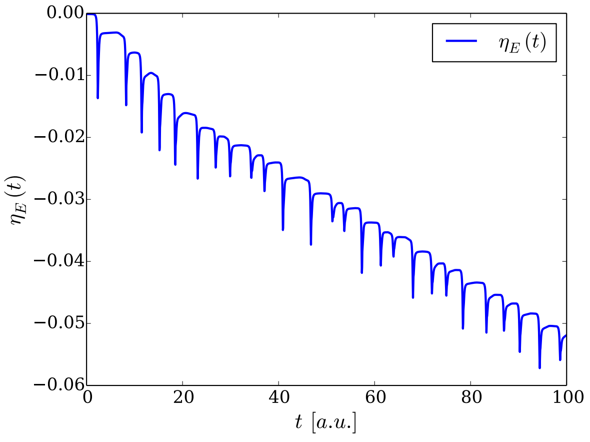
$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}) \tag{23}$$

where the right-hand side is given by

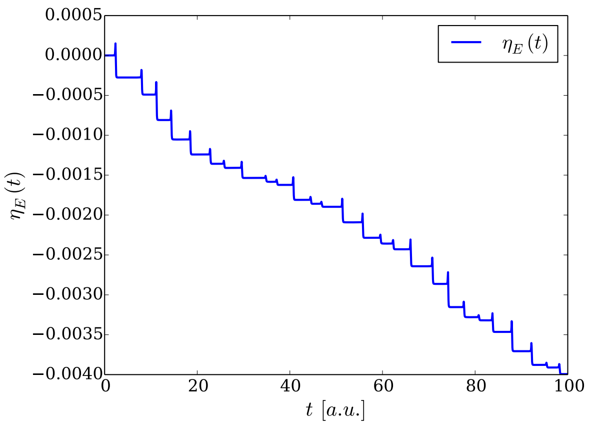
$$\boldsymbol{f}(\boldsymbol{y}) = \left(\dot{\phi}_1, \dot{\phi}_2, \dot{q}_1, \dot{q}_2\right)^T = \left(A^{-1}_{11} q_1 + A^{-1}_{12},\ A^{-1}_{21} q_1 + A^{-1}_{22} q_2,\ \frac{\partial L}{\partial \phi_1},\ \frac{\partial L}{\partial \phi_2}\right)^T =$$
$$\begin{pmatrix} A^{-1}_{11} q_1 + A^{-1}_{12} q_2 \\ A^{-1}_{21} q_1 + A^{-1}_{22} q_2 \\ -m_2 l_1 l_2 (A^{-1}_{11} q_1 + A^{-1}_{12} q_2)(A^{-1}_{21} q_1 + A^{-1}_{22} q_2) \sin(\phi_1 - \phi_2) - (m_1 + m_2) g l_1 \sin\phi_1 \\ m_2 l_1 l_2 (A^{-1}_{11} q_1 + A^{-1}_{12} q_2)(A^{-1}_{21} q_1 + A^{-1}_{22} q_2) \sin(\phi_1 - \phi_2) - m_2 g l_2 \sin\phi_2 \end{pmatrix} \tag{24}$$

where we inserted eqs. (21) and (22) for $\dot{\phi}_1$ and $\dot{\phi}_2$. Notably, A^{-1} depends only on ϕ_1 and ϕ_2.

(c) The relative energy error's time-evolution under a second-order Runge-Kutta integration scheme for the specified initial conditions is shown in fig. 4a.



(a) Second-order accuracy



(b) Fourth-order accuracy

Figure 4: Time evolution of the Runge-Kutta relative energy error

(d) Figure 4b displays the relative energy error of a calculation with matching initial conditions but performed at fourth-order accuracy. As expected, the absolute error still increases over time, but much slower than compared to that of second-order Runge-Kutta.

(e) See `double-pendulum.mp4`.