COMPUTATIONAL PHYSICS
First assignment WS 2022/2023

Deadline: 23rd of October 2022
Final date for the mini-exam: 24th/25th of October

The aim of this assignment is to get familiar with the numerical solution of linear system of ordinary differential equations.

Note: The assignments contain questions and tasks, labeled (a), (b), etc. For some of these tasks, you are supposed to create and hand in one of the following:

- 💻 a script or a part of a script written in the computer language of your choice;

- 📈 a figure with legends and axis titles.

Other questions primarily serve to guide you through the exercises; you do not need to submit the answers in writing. However, you may be asked these or similar questions during the assessment discussions.

Note: Many tasks in the exercises can be performed using an existing library or toolbox. In line with the aim of the course, however, programming your own analysis tools is encouraged. If libraries or toolboxes are used, you are expected to be able to explain in detail how these algorithms work.

Note: When collaborating in groups of up to 3 students, you are allowed to hand in identical code. Please list all collaborators, for example in the header.

Note: The scripts in the programming exercises should be considered to be intended for general use, and a corresponding coding style will be appreciated. For example, using input parameters as variables, clear presentation of input and output, naming of variables and/or comprehensive comments in the script are among the grading criteria.

1 A simple pendulum

The Runge-Kutta-method may be used for the solution of the following system of differential equations:

$$\dot{\vec{y}}(t) \;=\; F(\,\vec{y}(t), t\,) \,. \tag{1}$$

Herein the time is discretized in terms of the time step ϵ:

$$t_n \;=\; \epsilon\, n \quad \text{with } n = 0, 1, 2, \, ... \tag{2}$$

The solution vector $\vec{y}(t_{n+1})$ at the time t_{n+1} is obtained from the vector at the previous moment in time t_n:

$$\vec{y}(t_{n+1}) \;=\; \vec{y}(t_n) \,+\, \epsilon \sum_{j}^{d} b_j F(\vec{k}_j, t_n + \epsilon c_j) \tag{3}$$

with

$$\vec{k}_i = \vec{y}(t_n) + \epsilon \sum_{j}^{d} a_{ij} F(\vec{k}_j, t_n + \epsilon c_j) \tag{4}$$

where b_j, c_j and a_{ij} depend on the method used. Butcher tableaus are very useful to characterize such methods. They provide a structured representation of the coefficient matrix a and the coefficient vectors b and c.

$$
\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1d} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2d} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_d & a_{d1} & a_{d2} & \cdots & a_{dd} \\
\hline
 & b_1 & b_2 & \cdots & b_d
\end{array}
$$

The Runge-Kutta method is explicit, if it is zero on and above the diagonal ($a_{ij} = 0$ for $j \geq i$). For the Euler and Runge-Kutta method of 4th order (RK-4) they are given as:

- RK-4:

$$
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

- Euler:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

(a) 💻 Write a routine for a general explicit Runge-Kutta solver, which takes as inputs:

- a function F or a function handle
- the initial conditions
- a time vector, or end time t_{max} and increment ϵ
- the parameters of the Butcher tableau a_{ij}, b_j, c_j.

(b) 💻 Consider a simple pendulum where a mass m is connected to a frictionless hinge by a massless rod of length l. For convenience, assume $m = 1\,\text{kg}$ and $l = 1\,\text{m}$. Use the methods from above to solve the differential equation of the simple pendulum:[1]

$$\ddot{\theta}(t) = -\omega^2 \sin\left(\theta(t)\right) , \tag{5}$$

where $\omega^2 = g/l$ with g being the gravitation constant.

(c) 📈 Create plots for the total system energy $E(t)$ (Hamilton function) and $\theta(t)$ for at least three different values of ϵ and both Euler and RK-4 methods. As starting conditions you should use $\theta(0) = 0$ and $\dot{\theta}(0) = 2\omega$ and small deviations from these values. Interpret your results in terms of energy conservation.

(d) 💻 Solve Eq. (5) numerically with the RK-4 routine from (a) for a sufficiently long time vector to resolve the swinging periodicity and compare it to the value expected from the analytical solution of Eq. (5) for small θ. The following starting conditions should be used: $\theta(0) = \pi/3$ and $\dot{\theta}(0) = 0$.

---

[1]Hint: This equation is equivalent to a set of first order differential equations.

2 Double Pendulum

The differential equations of the double pendulum were obtained in the lecture:[2]

$$\dot{\theta}_1 = \frac{\tilde{p}_1 - \tilde{p}_2 \cos(\theta_1 - \theta_2)}{1 + \sin^2(\theta_1 - \theta_2)} \,,$$

$$\dot{\theta}_2 = \frac{2\tilde{p}_2 - \tilde{p}_1 \cos(\theta_1 - \theta_2)}{1 + \sin^2(\theta_1 - \theta_2)} \,,$$

$$\dot{\tilde{p}}_1 = -A + B - 2\omega^2 \sin \theta_1 \,,$$

$$\dot{\tilde{p}}_2 = A - B - \omega^2 \sin \theta_2 \,,$$

$$A = \frac{\tilde{p}_1 \tilde{p}_2 \sin(\theta_1 - \theta_2)}{1 + \sin^2(\theta_1 - \theta_2)} \,,$$

$$B = \frac{\tilde{p}_1^2 + 2\tilde{p}_2^2 - 2\tilde{p}_1 \tilde{p}_2 \cos(\theta_1 - \theta_2)}{(1 + \sin^2(\theta_1 - \theta_2))^2} \sin(\theta_1 - \theta_2) \cos(\theta_1 - \theta_2) \,, \tag{6}$$

where we have assumed $l_1 = l_2 \equiv l$ and $m_1 = m_2 \equiv m$ and defined $\tilde{p}_i = p_i/(ml^2)$, with p_i being the momentum of mass i.

(a) Derive these equations of motion from the Hamilton function of this mechanical system.[3]

(b) 🖥 Solve the system of Eq. (6) numerically using the RK-4 method for various initial conditions (e.g.: $\theta_1(0) = 0$, $\theta_2(0) = 0$, $\tilde{p}_1(0) = 4$ and $\tilde{p}_2(0) = 2$ or $\theta_1(0) = 0$, $\theta_2(0) = 0$, $\tilde{p}_1(0) = 0$ and $\tilde{p}_2(0) = 4$).[4]

(c) 📈 For the calculated trajectories, make a plot of the points $(\theta_1(t_n), \tilde{p}_1(t_n))$ where the conditions $\theta_2(t_n) = 0$ and $\tilde{p}_2(t_n) > 0$ are fulfilled (Poincaré - map). Chaotic behaviour may be found if these points are area-filling. If the points are concentrated on single lines the system is non-chaotic. You should try a number of different starting conditions different from the ones mentioned in (b) and generate at least two Poincaré maps, one of which shows chaotic behaviour and one of which does not.

(d) 📈 Plot the total energy of the system (Hamilton function) over time to verify to what extent the energy is conserved and adjust ϵ if necessary.

---

[2] The masses and lengths of the two pendulums are all set to 1.
[3] This task is optional.
[4] These conditions lead to very instructive examples if used in combination with $\omega^2 = 9.81$.

3 Stability analysis

Considering two trajectories in phase space, $y(t) = (\theta_1(t), \theta_2(t), \tilde{p}_1(t), \tilde{p}_2(t))$ and $y'(t) = (\theta'_1(t), \theta'_2(t), \tilde{p}'_1(t), \tilde{p}'_2(t))$, resulting from two slightly different sets of initial conditions. The difference of these two trajectories in phase space is given by:

$$\delta(t) = \sqrt{(\theta_1(t) - \theta'_1(t))^2 + (\theta_2(t) - \theta'_2(t))^2 + (\tilde{p}_1(t) - \tilde{p}'_1(t))^2 + (\tilde{p}_2(t) - \tilde{p}'_2(t))^2}, \qquad (7)$$

(a) 💻 Use your Runge-Kutta program to calculate two trajectories in phase space for only slightly deviating initial conditions.

(b) 📈 Calculate and plot $\delta(t)$ for chaotic and non-chaotic initial conditions.[5]

4 Adaptive Runge-Kutta (Bonus-Task)

(a) 💻 Implement a Runge Kutta method of 4th order with adaptive time stepping.

(b) 💻 Use your new routine to find a numerical solution to the equations of motion of Eq. (6). What is the effect on the running time of your code in comparison to the fixed-time-stepping case?

5 Animation (Bonus-Task)

(a) 📈 Create an animation of your double-pendulum.[6]

---

[5]It is sufficient to pick one chaotic and one non-chaotic case. You should nonetheless calculate $\delta(t)$ for at least three different sets of slightly different initial conditions for both cases.

[6]These references may be useful: Julia `https://docs.juliaplots.org/latest/animations/`, MAT-LAB: getframe and movie, Python: `https://matplotlib.org/stable/api/animation_api.html`.