



1. We implement the following model for a spring pendulum:

$$\begin{aligned} x''(t) + 0.01x'(t) + x(t) &= 100w(t), & x(0) = x_0, \quad x'(0) = v_0, \quad w(0) = w_0. \\ w'(t) + 1000w(t) &= x'(t) \end{aligned}$$

$x(t)$ is the **position**, $v(t) := x'(t)$ the **velocity** of the pendulum and $w(t)$ is a model for the **thermal energy** of the pendulum that affects the damping.

a) (theoretical exercise) Show that the second order system can be written as the first order system

$$\mathbf{x} : \mathbb{R} \rightarrow \mathbb{R}^3 \quad \mathbf{x}'(t) = \begin{pmatrix} 0 & 1 & 0 \\ -1 & -0.01 & 100 \\ 0 & 1 & -1000 \end{pmatrix} \mathbf{x}(t), \quad \mathbf{x}(0) = \begin{pmatrix} x_0 \\ v_0 \\ w_0 \end{pmatrix}$$

b) Implement the first order system with the explicit forward Euler method, Heun's method or with Runge-Kutta (you can choose). Use the initial value

$$\mathbf{x}(0) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

- (i) Print $\mathbf{x}(T)$ for $T = 10$.
- (ii) Plot the solution in the time interval $I = [0, 10]$.

Answer the following questions (by numerical experiments)

- (iii) What is the maximum time step $h > 0$ that gives a stable solution?
- (iv) What is the order of convergence of your method?

hint1.py implements Heun's second order methods
solution1.py gives the complete solution

2. **(Optional)** We write a program to plot the *region of absolute stability* for single-step methods. We solve

$$u'(t) = \lambda u(t), \quad u(0) = 1,$$

for $\lambda \in \mathbb{C}$.

Write a program that performs one step of a single step method (for example forward Euler)

$$u_1 = u_0 + \underbrace{h\lambda}_{=:z} u_0 = u_0 + zu_0.$$

Compute $|u_1|$ and decide if the method is stable for $z = \lambda h$.

Hint: Choose an interval for real and imaginary part of $z = \lambda h$, for instance $\operatorname{Re}(z) \in [-4, 4]$ and $\operatorname{Im}(z) \in [-4, 4]$. Create a mesh of points in this interval $z_{ij} = x_i + jy_j$ and for each points compute the resulting $|u_1|$. Plot, if the absolute value is larger than one or not.

hint2.py shows how to work with complex numbers and implements the forward Euler method. Also it shows how to create a grid of points and how to plot values.

solution2.py gives the complete solution.

3. We solve the system from exercise 1

$$\mathbf{x}'(t) = \begin{pmatrix} 0 & 1 & 0 \\ -1 & -0.01 & -100 \\ 0 & 1 & -1000 \end{pmatrix} \mathbf{x}(t), \quad \mathbf{x}(0) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

with the implicit methods.

a) Implement the implicit backward Euler method

$$u_n = u_{n-1} + hf(u_n)$$

and plot the solution on $T = [0, 10]$ using the large step size $h = 1$.

Note: The solution looks very different from the solution in Exercise 1. This is due to the large step size h . If you decrease the step size, the solution will get more similar.

b) Determine the order of convergence of the implicit Euler method by simulating with decreasing step sizes $h = 1, h = 0.5, \dots$ and observing the solution $\mathbf{x}(10)$.

c) Now, implement the trapezoidal rule

$$u_n = u_{n-1} + \frac{h}{2}f(u_{n-1}) + \frac{h}{2}f(u_n)$$

and plot the solution on $I = [0, 10]$ for $h = 1$ and $h = 0.5$. Compare the solution to the Euler solution.

Note: The Euler method is strongly damping. The trapezoidal rule conserves the energy of the pendulum.

d) Determine the order of convergence of the trapezoidal rule (like **b**)

Hints: In each time step you must solve a linear system of equations. In numpy you can invert a matrix A with `np.linalg.inv(A)`. If you need the 3×3 unit matrix, it is `np.eye(3)`.

*hint3.py implements the backward Euler method
solution3.py gives the complete solution*

4. We now solve the nonlinear spring pendulum:

$$\left. \begin{aligned} x_1'(t) &= x_2(t) \\ x_2'(t) &= -x_1(t) - 0.01x_2(t) - 100x_2(t)^2x_3(t) \\ x_3'(t) &= \exp(2x_1(t)) - 1000x_3(t) \end{aligned} \right\} \quad \mathbf{x}(0) = \begin{pmatrix} x_0 \\ v_0 \\ w_0 \end{pmatrix}$$

a) Solve the problem on $I = [0, 10]$ using an explicit method and very small time steps (e.g. $h = 0.002$) and plot the solution.

b) Implement the implicit Euler method and the trapezoidal rule using Newton's method. Use a large time step $h = 0.2$ and plot both solutions.

c) Experimentally determine the order of convergence of both implicit methods using large time steps $h = 1, h = 0.5, \dots$

hint4.py implements with implicit Euler method with Newton
solution4.py gives the complete solution