CentraleSupelec **Numerical Methods, S8**

Ordinary Differential Equations

Highlights:

- Solving an ODE system
- Using explicit methods with adapted time step
- Using implicit methods, using full and linearized forms

1 Harmonic oscillator

The first problem is devoted to the study of harmonic oscillators, such as spring-mass systems:

$$\ddot{x} + \omega^2 x = 0 \tag{1}$$

where ω is the characteristic wave number of the system. The objective is to solve this equation using ODE solvers, and to compare the results in terms of errors and stability. The analytical solution is:

$$x(t) = A\cos(\omega t + \varphi) \tag{2}$$

where $A = \sqrt{x_0^2 + \frac{u_0^2}{\omega^2}}$ is the amplitude and $\varphi = \arctan(-\frac{u_0}{x_0\omega})$ is the initial phase lag, which depend on the initial position $x_0 = x(t=0)$ and the initial velocity $u_0 = \dot{x}(t=0)$.
To solve the ODE, we will use a time-marching method. To do so, the equation is classically splitted into two equations using $u(t) = \dot{x}$. Considering $y_0 = x$ and $y_1 = u$:

$$\begin{cases} \dot{y_0} = y_1 \\ \dot{y_1} = -\omega^2 y_0 \end{cases} \tag{3}$$

1.1 Explicit method

1. In a module file, implement the Forward Euler method which arguments will be the initial solution y_0, the initial time t_0, the final time t_{end}, the time step Δt, and returns two vectors: the solution and time arrays.
2. In a main program file, define the initial solution with $x_0 = 1$ and $\dot{x}_0 = 0$, and the parameter $\omega = 4$. Call the module function and plot the results.
3. Check the solution behaviour while changing the time step.
4. The purpose is to get the best estimate of $x(t = 10)$. Evaluate the global error made by the Forward Euler method as a function of the time step Δt.

1.2 Implicit methods

1. In your module file for ODE solvers, implement the Backward Euler and Trapezoidal methods.
2. For a given time step $\Delta t = 0.1\ s$, compare on the same plot the time evolution of the solution obtained by every solvers.
3. Compare the errors made by each scheme as a function of the time step, and comment on the stability limits of each method.

1.3 High order explicit methods

Here, the objective is to investigate the accuracy and the stability of high order explicit methods. The method of interest are the family of Runge-Kutta methods (RK2, RK3, RK4), the Leapfrog method, and the Adams-Bashforth method.

1. Find the formulation of these schemes on internet and implement them.
2. Comment on the accuracy and the stability limits.

2 Evolution of a population

The evolution of a population can be written as:

$$\dot{P} = -P(1-P) \tag{4}$$

where P is the population normalized by the maximum population. The analytical solution of this equation is:

$$P(t) = \frac{1}{1 - Ae^t} \tag{5}$$

where $A = 1 - 1/P_0$ and $P_0 = P(T = 0)$. Now you will solve this equation. As the equation is non-linear, we can either use Backward Euler and Trapezoidal methods by inverting a non-linear system or use the linearized version of theses schemes for which the Jacobian $\frac{\partial F}{\partial Y}$ is required as an argument.

1. Solve the equation using Forward Euler method. Analyse the stability of the method for the given equation.
2. Implement the full Backward Euler and Trapezoidal schemes.
3. Implement the linearized versions of theses two schemes (see Appendix).
4. Compare the time evolution of this equation obtained by the five solvers you have implemented.

Appendix

Non-linear solvers

We consider ODE of the general form:

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = F(t, Y(t)) \tag{6}$$

The Forward Euler's method is written in general form:

$$Y_{n+1} = Y_n + \Delta t F(t_n, Y_n)) \tag{7}$$

The Forward Euler's method is written in general form:

$$Y_{n+1} = Y_n + \Delta t F(t_{n+1}, Y_{n+1}) \tag{8}$$

while the trapezoidal scheme is written:

$$Y_{n+1} = Y_n + \Delta t \frac{F(t_n, Y_n) + F(t_{n+1}, Y_{n+1})}{2} \tag{9}$$

This two schemes end up in a system to be solved in order to find Y_{n+1}. If the RHS is linear in Y, it can be easily solved using *solve* in *numpy.linalg*. In the non-linear case, It can be done using non-linear solvers such as *fsolve* in *scipy.optimize*., a linearisation can be used:

$$F(t_{n+1}, Y_{n+1}) = F(t_{n+1}, Y_n) + \frac{\partial F}{\partial Y}(Y_{n+1} - Y_n) \tag{10}$$

where $\frac{\partial F}{\partial Y}$ is the Jacobian of F with respect to the solution vector Y. This linearisation is obviously exact when the RHS is a linear function of Y. Using this linearisation, the two schemes are then rewritten. The Linearised Forward Euler's method is written:

$$\left(\mathbb{1} - \Delta t \frac{\partial F}{\partial Y}\right) Y_{n+1} = \left(\mathbb{1} - \Delta t \frac{\partial F}{\partial Y}\right) Y_n + \Delta t F(t_{n+1}, Y_n)) \tag{11}$$

while the linearized trapezoidal scheme is written:

$$\left(\mathbb{1} - \frac{\Delta t}{2} \frac{\partial F}{\partial Y}\right) Y_{n+1} = \left(\mathbb{1} - \frac{\Delta t}{2} \frac{\partial F}{\partial Y}\right) Y_n + \Delta t \frac{F(t_n, Y_n) + F(t_{n+1}, Y_n)}{2} \tag{12}$$

Solving linear and non-linear systems: solve and fsolve

- To solve linear system of equations, the function *solve* in *numpy.linalg* can be used. The syntax is $x = \text{solve}(A, b)$ for solving $Ax = b$, where A is the $n \times n$ array and b a vector of size n.
- To solve non-linear system of equations, the function *fsolve* in *scipy.optimize* can be used. The syntax is $x = \text{fsolve}(G, x_0, args = (a, b, c))$ for solving $G(x, a, b, c) = 0$ with initial guess $x = x_0$.