Challenge 1. Build a simple solver for a Cauchy problem

Luca Formaggia and Matteo Caldana

AA 22-23

We have to tackle an initial value problems of the following form: find an approximation of $y = y(t)$ solution of

$$\begin{cases} \dfrac{d}{dt}y(t) = f(t, y(t)),\ t \in (0, T], \\ y(0) = y_0, \end{cases} \tag{1}$$

where $y_0 \in \mathbb{R}$ and $f : (0, T) \times \mathbb{R} \to \mathbb{R}$ are the initial conditions and a given forcing term, respectively.
A possible numerical scheme to solve the problem is *Crank Nicolson*. Being N a positive integer and $h = T/N$ the time step, the method consists in finding $u_n \simeq y(t_n)$ for $t_n = nh$ by solving

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

for $n = 0, \ldots N - 1$, where $u_0 = y_0$.
Therefore at, each time index n we have to find the zero of a (usually non-linear) function

$$F(x) = x - \frac{h}{2}(f(t_{n+1}, x) + f(t_n, u_n)) - u_n,$$

and then set u_{n+1} equal to the found zero.

1 The challenge

Write a function that takes in input a *function wrapper* that defines f, the initial condition, the final time T, the number of steps N (and any other data you deem appropriate) and returns an array with the vectors containing t_n and u_n for all n.
You are free to choose how to organize the code, the important thing is that the user should be able to obtain the solution.
Finally, apply the code by writing a main that solves

$$\begin{cases} \dfrac{d}{dt}y(t) = -te^{-y}\ t \in (0, 1], \\ y(0) = 0. \end{cases} \tag{2}$$

and store the solution in a file.

1.1 Suggestions and available software

- For the solution of the non-linear problem you may exploit one of the methods in the file `LinearAlgebraUtil/basicZeroFun.hpp` of the repository of the Examples of the course, for instance `Newton`, where the derivative may be approximated by finite differences (in the folder `Derivatives` you have some general tools for this purpose, but you can do it by yourself). Or you may use the `secant` method.
- You are free to choose how to organize your code, try to follow what you have seen at lecture.
- You may want to use `Gnuplot` to see the result. Look at what is contained int the folder `HeatExchange`. You find in the code how to use `gnuplot—iostream`, but also (simpler) a shell script that launches gnuplot. Look at the gnuplot manual or at the video on Youtube Or you can load the file produced by your code in Matlab.
- You may want to compute the convergence rate of the algorithm with respect to the size of the discretization $\Delta t = 1/N$ to check that your implementation is correct.
- You should write some comments in your code to help us understand what you are doing...

2 General rules for the Challenges

- Challenges are meant to be a tool to help you exercise programming, put in practice what seen at the lecture, and getting ready for the project. Not really to grade you. They are graded in order to give some satisfaction to the ones of you who is making the effort, but this is not their main purpose.

  So, if you need help, ask for it. Use the Forum on `WeBeep`, so that the answer may be useful also to others.
- As I said, you are free to choose how to organize data, how to provide the input to your code, or to name variables etc. For instance, you may decide to write a class instead of a simple function. Try to put in practice what you have seen at the lectures and write a clean code.
- You should write a working Makefile that compiles your code.
- When finished, put everything in a compressed file (use zip, or tgz, or 7z, we don't care) and upload it in `WeBeep`.

3 Extras

Well, if you want to be more daring, why not implement a general θ scheme?

$$u_{n+1} - h\theta f(t_{n+1}, u_{n+1}) = u_n + h(1-\theta)f(t_n, u_n),$$

for a given $\theta \in [0, 1]$. With $\theta = 1$ you have backward Euler, $\theta = 1/2$ Crank-Nicolson, $\theta = 0$ forward Euler.
Or, why not use MuParser to read the forcing term from a getpot or json file, together with the other parameters?