PRACTICAL WORK I

You are allowed to write your code in *any* langage that suit you the best, as long as the final code is runnable and debugged. That being said, a notebook in `Python` is available on my personal webpage at `https://www.ceremade.dauphine.fr/~lelotte/`. The notebook already contains most (if not *all*) of the code needed to answer all the questions of this practical work is a (very) reasonable amount of time — your job is simply to « fill the gaps » in the code. When asked to « comment » or « explain » something, add either a comment (in the code) or a textual cell (in the notebook). Send your work at lelotte@ceremade.dauphine.fr.

This « exam » — though effectively graded — is the occasion for you to (concretely) apply some of the (abstract) notions you've learned during the lectures[1] . Feel free to roam in your notes or any supplementary material — and also feel free to ask me any questions regarding the implementation of your (and of my own) code — **I'm here to help you!**[2]

I — Stiff equations

One shall expect that, as the solution of a simple ODE of the form $y' = f(t, y)$ displays much variation in some region, a small step-size $h \ll 1$ is required in this very region when resorting to straightforward numerical schemes. It turns out that, in some peculiar problems, the step-size is required to be at an unacceptably small level even though the solution curve is extremely smooth and nicely-behaved — this is the essence of *stiffness*. We shall explore this phenomenon with the simple equation

$$y' = \lambda(y - g(t)) + g'(t) \tag{1}$$

[1] Unless you set yourself to become a professional mathematician, the `*insert here any obscure theorem*` will certainly be of no-use in the aftermath of your degree. That being said, there is a high percentage of chance that, in your future job, you will be required to code little pieces of programs here and there !

[2] Also, even though this practical work should be done individually, you are allowed to « discuss » between each others — which does not mean: copy-pasting the code of your nearest neighbor like a typing monkey !

where the general solution of Equation (1) is given by $y(t) = g(t) + ce^{\lambda t}$, where $g : \mathbb{R} \to \mathbb{R}$ is any (smooth) function, $\lambda < 0$ is some negative number and $c \in \mathbb{R}$ depends on the initial condition.

Problem 1. Consider Equation (1) with $g(t) = \tanh(t+2)$ with initial condition $y(0) = g(0)$ (*i.e.* $c = 0$) and $\lambda = -100$. Suppose that the initial measurement carries a small error ε, that is $y_0 = g(0) + \varepsilon$ with $\varepsilon = 10^{-3}$, and integrate the equation on $[0, T]$ with $T = 2$.

(a) Use the *forward Euler scheme* with $h = T/N$ where the number of steps varies in $N \in \{99, 100, 101\}$. What do you notice ?
(b) Use the *4th-order Runge-Kutta* method given at the **Example 2.21** of the lecture notes with $N \in \{70, 71, 72\}$. Similarly as before, what do you notice ?
(c) Given any numerical scheme $\mathcal{N}$, we denote by $s_{\mathcal{N}} : \mathbb{C} \to \mathbb{C}$ the function such that the *region of stability* of $\mathcal{N}$ is defined as

$$\mathcal{A}_{\mathcal{N}} = \{z \in \mathbb{C} : |s_{\mathcal{N}}(z)| < 1\} . \tag{2}$$

Compute $s_{\mathcal{N}}$ for the two preceding numerical schemes, plot the (boundary of the) regions of stability $\mathcal{A}_{\mathcal{N}}$, and explain why the *4th-order Runge-Kutta* method is performing « better ».
(d) Finally, implement the *backward Euler scheme* with much smaller N's — and explain why this scheme largely outperforms the previous ones.

II — Orders of convergence

Let $y : [0, T] \to \mathbb{R}$ be the solution to the generic ODE $y' = f(t, y)$ with $y(0) = y_0$. Recall that a numerical scheme is said to be of *order* p if the following condition is verified

$$\tau(h) := \max_{k \in \{0, \ldots, T/h\}} |y_k - y(t_k)| = \mathcal{O}(h^p), \quad \text{as } h \to 0, \tag{3}$$

where $t_k := kh$ and y_k (for $k = 0, \ldots, T/h$ with $T/h \in \mathbb{N}$) denotes the approximation of $y(t_k)$ given by the numerical scheme (that is, the k-th step of the method). The bigger is p, the « better » is the numerical scheme. Evidently, appealing to Equation (3), if a numerical scheme has order p, then we shall expect, denoting $h_q = T/2^q$ for $q \in \mathbb{N}$, that

$$\frac{\tau(h_q)}{\tau(h_{q+1})} \simeq 2^p \implies p \simeq \log_2 \left(\frac{\tau(h_q)}{\tau(h_{q+1})} \right) . \tag{4}$$

Therefore, using Equation (4), one can numericaly « retrieve » the order of convergence p.

Problem 2. Let us consider the two following dynamics, namely $f_1(t, y) = y$ (with $y(0) = 1$; so that evidently the exact solution reads $y_1(t) = e^t$) and $f_2(t, y) = 1 + \sqrt{y}$ with $y(0) = 0$. The exact solution y_2 to the second ODE is unique and given by

$$y_2(t) = \left(1 + W(-e^{-1-t/2})\right)^2, \quad \text{for all } t \in \mathbb{R}_+ \tag{5}$$

where W is an important special function coined as the *Lambert function*, defined as the reciprocal of $t \mapsto te^t$ (that is, if $s = te^t$, then $t = W(s)$).

(a) Integrate $y' = f_1(t, y)$ on $[0, T]$ with $T = 2$ using the *forward Euler scheme*, the *Heun method* and the *4th-order Runge-Kutta method* with $h = T/2^q$ for $q \in \{1, \ldots, 10\}$. Plot $q \mapsto \log_2(\frac{\tau(h_q)}{\tau(h_{q+1})})$ and determine numerically the order of convergence of each methods.

(b) Now, integrate $y' = f_2(t, y)$ on $[0, T]$ with $T = 2$ using the same methods and parameters as previously — what do you observe? Come up with an explanation.