LM3331 INDIVIDUAL PROJECT II: Ordinary differential equations • REPORT

Raquel Romão, 5629608

The goal of this project was to simulate the progress of the concentrations of all the chemical components during a *Belousov-Zhabotinskii* reaction. Basically, to do so, it's needed to solve an initial value problem. Taking this into account, I started to build a system of differential equations for each component that translates the problem. Then, I solved it using the built-in python method solve_ivp, as well as by using a first-order scheme coded on my own.

In this problem, some of the reactions are slow and others fast, so in this way we can say we are in the presence of a stiff problem. Consequently, not every method will work. When obtaining the solutions with solve_ivp using different solvers, that was exactly what I was able to observe. I studied 4 different solvers considering that I would need a small tolerance, since it implies the errors are smaller, while the order of stability remains. Therefore, I simulated the problem with 3 solvers that deal with stiffness (`**LSODA**´, ` **Radau**´ and `**BDF**´) and one that doesn't (`**RK45**´), for comparison.

For Runge-Kutta 45 (`**RK45**´, the default solve_ivp method), while using the default tolerance, the method clearly diverges and presents a short simulation, as it was to expect since this solver doesn't deal with stiffness well. After adjusting the tolerance to a smaller one, in order to try to solve this, the function entered a never-ending loop. The solution could be eventually reached; however, it would take ages.

Using `**BDF**´ solver with a relative tolerance of 10^{-4} and an (default) absolute tolerance of 10^{-6}, the simulation obtained was incomplete, showing only the correct version of it in the beginning. Nevertheless, setting the absolute tolerance to 10^{-8} solved it! Something similar happened using the `**LSODA**´ solver, with the default tolerance the method diverged not so long after the beginning of the simulation but when lowering the absolute tolerance, the simulation was performed correctly. At last, using `**Radau**´ solver the simulation was obtained without the need to adjust the default tolerance. This last solver would be the one I would go for when solving this problem, since it deals with stiffness and has the higher accuracy between all three.

To obtain this type of simulation, i.e. solve this type of non-linear equations, we don't necessarily need a higher order method like the ones used, we need a method that can deal with stiffness (event though we have a higher accuracy with bigger step sizes when using higher order methods). The region of stability of Backward-Euler (implicit) is quite big (one of the more stable methods), it only fails in the complementary region of Forward-Euler (explicit), that is one of the reasons why implementing an implicit Backward-Euler-based function is a good option.

To implement this first-order scheme function, I first used Forward-Euler as an initial guess for the next step and then, to check the prediction of the step was done correctly, I used Newthon-Raphson method. This translates to a more robust method, minimizing the gap between the approximation and the function, making it possible to simulate this problem.

It's to note that when we use an approximation to evaluate the function, we lose stability, that is why we use a method that finds a root such as Newthon-Raphson. Moreover, since Newthon-Raphson doesn't reassure conversion, we set a small h to reassure our solution is within that step, what I consider exactly to be one of the limitations of my function; if a set a bigger tolerance the simulation isn't done in its correct timing (like shown in the Jupyter notebook), what could be misleading. Furthermore, the function it's slow to give the solution.