Programming Project

Tea MIKELIC

December 14, 2017

1 Introdutcion

We would like to determine the trajectory of a charged particle in a vertical electric field. So, our project will approximate the solution of a system of differential equations. At the same time, we will learn how to use a scientific library, GSL, because our code will be in C++. The project is separate on several parts:

- First order differential equation
- Second order differential equation
- Git link
- References

2 First Order differential equation

We would like to solve an inital value problem:

2.1 Theoritical solution

$$\begin{cases} \frac{\partial y}{\partial t} = f(y(t), t) \\ y(t = 0) = y_0 \end{cases}$$

To find a numerical approximation of the solution of this problem, we will use a backward Euler scheme:

$$\begin{cases} y_{n+1} = y_n + hf(y_{n+1}, t = nh + h) \\ y_0 \quad given \end{cases}$$

We will do our own code and use the GSL version of this scheme.

2.2 The Algorithms

We provide here:

- Two main programs in C++ which provide the numerical approximation for the two different ways of solving the problem.
- A make file which compiles our programs
- An example input file which defines the IVE (time step h, time interval [a,b] and the initial value)
- A bash script which uses gnuplot to display the solution

2.3 The example

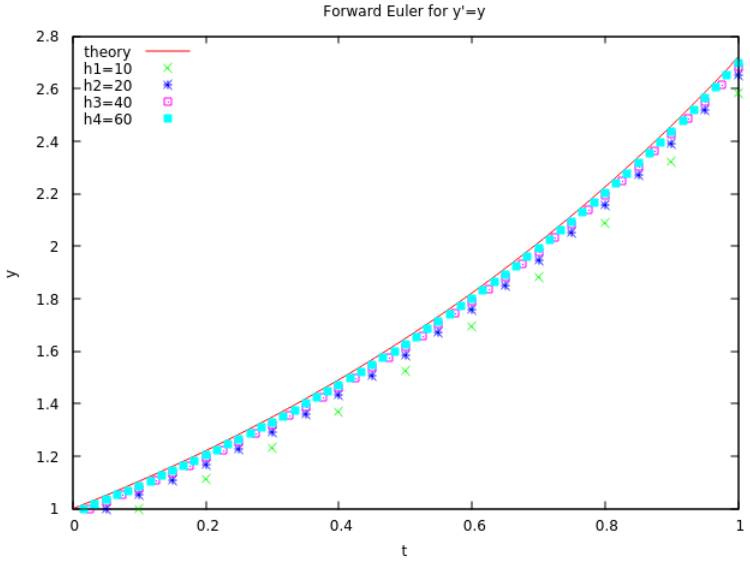
We will use the following example:

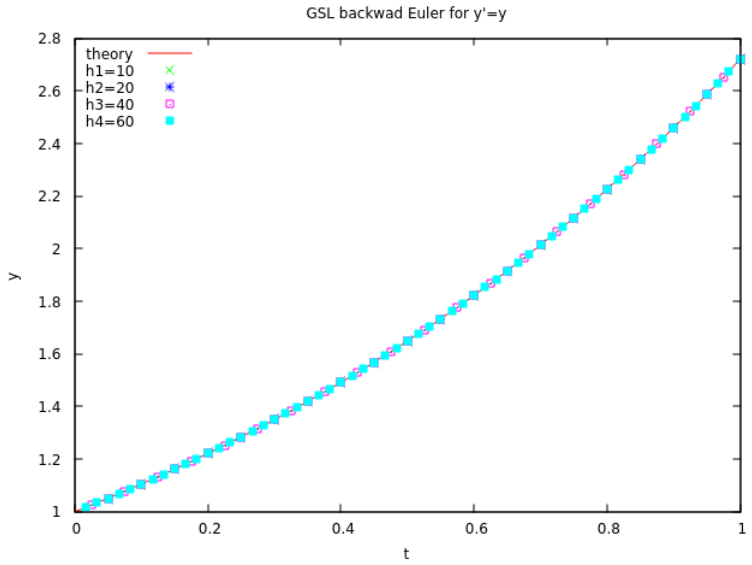
$$\begin{cases} \frac{\partial y}{\partial t} = y(t) \quad \forall y \in [0,1] \\ y_0 = 1 \end{cases}$$

The solution of this first order differential equation is:

$$y(t) = e^t$$

The results are shown in the folloing figures:

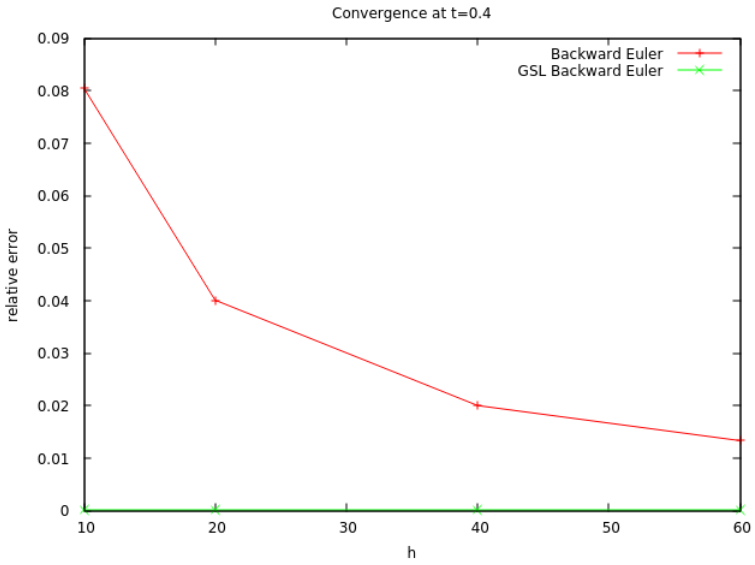




And the performances are (for h= 50 000):

Code	performance
Explicit Forward Euler	0.09 s
GSL	0.14 s

GSL need more time to solve the problem because there is an adaptive integrator used in it. My code doesn't work without it. It explain why the result is better even for small time step and why it takes more time to run the code. The following figure shows the convergence of the two methods at point t= 0.4 s



3 Charged Particule motion in a vertical electric field

3.1 The problem and Runge Kutta method

The problem to solve is:

$$\begin{cases} \frac{\partial^2 x}{\partial t^2} = \omega \frac{\partial y}{\partial t} - \frac{1}{\tau}\frac{\partial x}{\partial t} \\ \frac{\partial^2 y}{\partial t^2} = -\omega \frac{\partial x}{\partial t} - \frac{1}{\tau}\frac{\partial y}{\partial t} \\ \frac{\partial^2 z}{\partial t^2} = -\frac{1}{\tau}\frac{\partial z}{\partial t} \end{cases}$$

$$X_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} V_0 = \begin{bmatrix} u_0 \\ v_0 \\ w_0 \end{bmatrix}$$

We rewrite these equations as a set of 6 first order differential equations:

$$\begin{cases} \frac{\partial x}{\partial t} = u \\ \frac{\partial y}{\partial t} = v \\ \frac{\partial z}{\partial t} = w \\ \frac{\partial u}{\partial t} = \omega v - \frac{u}{\tau} \\ \frac{\partial v}{\partial t} = -\omega u - \frac{v}{\tau} \\ \frac{\partial w}{\partial t} = -\frac{w}{\tau} \end{cases}$$

And for numerical application we choose:

$$X_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, V_0 = \begin{bmatrix} 20 \\ 0 \\ 2 \end{bmatrix}, \omega = 5, \tau = 5$$

We use, in tis case, the GSL library with Runge kutta method. The general way to write this method is the following way (see wikipedia and ref [1]):

$$\begin{cases} y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \\ t_{n+1} = t_n + h \\ with: \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + c_2 h, y_n + h a_{21} k_1) \\ k_3 = f(t_n + c_3 h, y_n + h a_{31} k_1 + h a_{32} k_2) \\ ... \\ k_s = f(t_n + c_s h, y_n + h\sum_{1}^{s-1}(a_{si} k_i) \end{cases}$$

We use three different Runge Kutta methods, rk2, rk4 and rk8pd. So for the

Classical Runge Kutta (rk4);

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ t_{n+1} = t_n + h \\ with: \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{h}{2}, y_n \frac{k_1}{2}) \\ k_3 = f(t_n + \frac{h}{2}, y_n \frac{k_2}{2}) \\ k_4 = f(t_n + h, y_n + hk_3) \end{cases}$$

For the Runge Kutta 2:

$$\begin{cases} y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2) \\ t_{n+1} = t_n + h \\ with: \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + h, y_n + hk_1) \end{cases}$$

And finally for the Runge Kutta 8, Dormand Prince method:

$$\begin{cases} y_{n+1} = y_n + h * (\frac{35}{384}k_1 + 0 * k_2 + \frac{500}{1113}k_3 + \frac{125}{192}k_4 - \frac{2187}{6784}k_5 + \frac{11}{84}k_6) \\ t_{n+1} = t_n + h \\ with: \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{h}{5}, y_n + \frac{h}{5}k_1) \\ k_3 = f(t_n + \frac{3h}{10}, y_n + \frac{3h}{40}(k_1 + 3k_2)) \\ k_4 = f(t_n + \frac{4h}{5}, y_n + h(\frac{44}{45}k_1 - \frac{-56}{15}k_2 + \frac{32}{9}k_3)) \\ k_5 = f(t_n + \frac{8h}{9}, y_n + h(\frac{19372}{6561}k_1 - \frac{-25360}{2187}k_2 + \frac{64448}{6561}k_3 - \frac{212}{729}k_4)) \\ k_6 = f(t_n + h, y_n + h(\frac{9017}{3168}k_1 - \frac{-355}{33}k_2 + \frac{46732}{5247}k_3 + \frac{49}{176}k_4 - \frac{5103}{18656}k_5)) \\ k_7 = f(t_n + h, y_n + h(\frac{35}{384}k_1 + 0 * k_2 + \frac{500}{1113}k_3 + \frac{125}{192}k_4 - \frac{2187}{6784}k_5 + \frac{11}{84}k_6)) \end{cases}$$

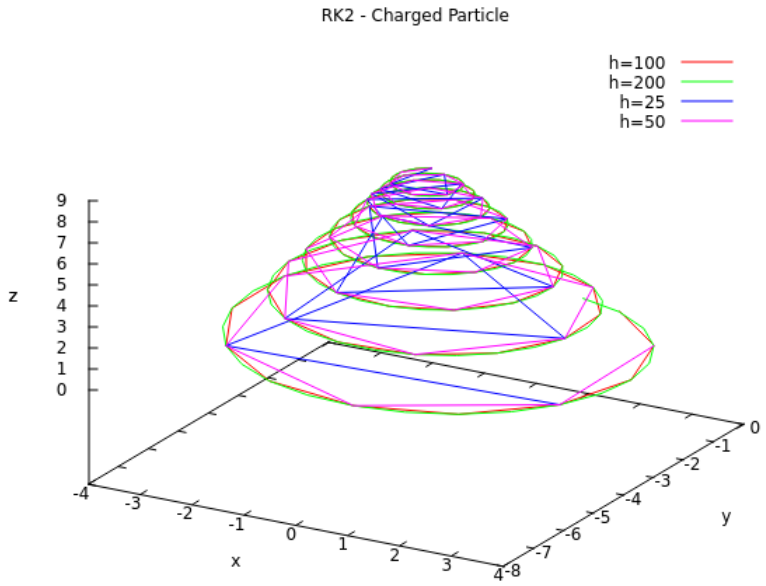
3.2 The Algorithm

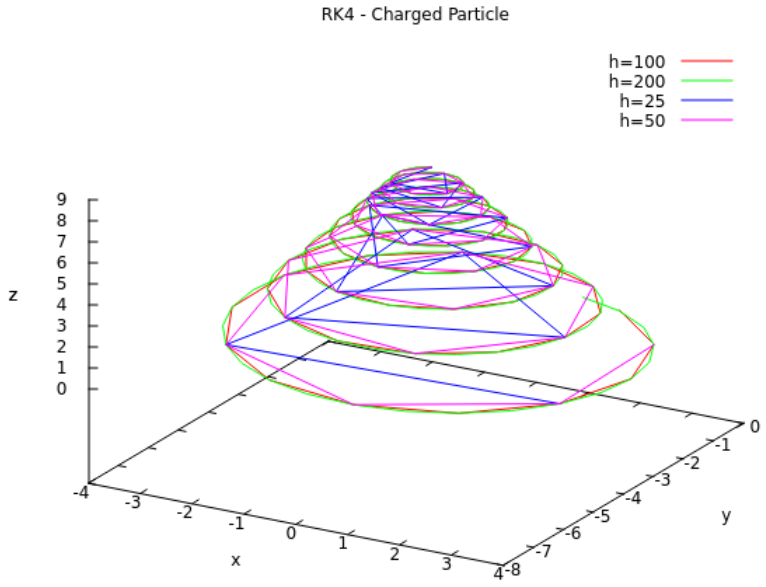
We split the problem in two parts, because as, we can see, w depends only on t and z, and u,v depend on t, x, y. We choose to solve the ODE in the interval [0, 10]

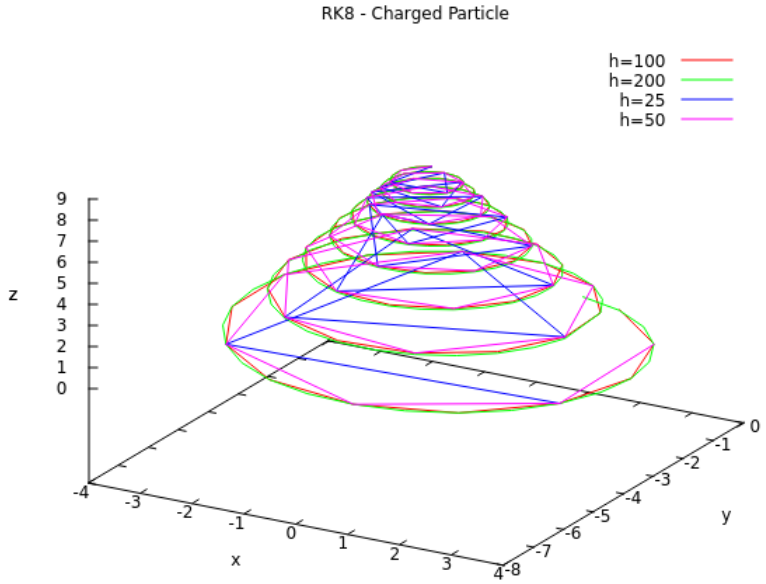
We provide for every methods:

- Two main programs in C++. One computes the z trajectory and the other one the x-y trajectory
- A make file which compiles our programs
- An example input file which defines the IVE (time step h, time interval [a,b] and the initial value)
- A bash script which uses gnuplot to display the trajectory

3.3 The Results







3.4 The performance

For h= 50 000, we run the three versions of the code:

Method	performance
rk2	2.83 s
rk4	15.36 s
rk8	10.88 s

As for the section 2, there is an adaptive integrator used in GSL. My code doesn't work without it. So I am enable to give the convergence of the method, because the error is exactly the same at every time step I choose to use.

3.5 Example of an imput file

```
0
0.0
10.0
50.0
0.0
0.0
20.0
0.0
```

debug mode 1 / standard mode 0

time interval [a=0, b=10]

time step h

Initial value here a vector

4 Git

The link to the Github project: https://github.com/tm33322/project

References

[1] Numerical Mathematics, Quarteroni, Alfio, Sacco, Riccardo, Saleri, Fausto. Copyright ©Springer; 2nd edition (November 10, 2006).

[2] GNU Free Documentation License, 27 Ordinary Differential Equations. Copyright ©2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

[3] Wikipedia, the free encyclopedia, RungeKutta methods.

[4] Wikipedia, the free encyclopedia, DormandPrince method.