Charged Particle motion in a vertical field

Tea MIKELIC

CSE 380 - Final Project

12th december 2017

Table of contents

1. Introduction

The goals is to provide the trajectory of charged particle which is ruled by a second order differential equation system using several numerical methods. Work done in several parts :

- Learn how to use GSL and compare it to a simple way of solving an ODE Forward Euler
- Provide an analytical example to test our code
- Use GSL to provide the trajectory

Our work as been done in C++ using GSL scientific library

2. Simple case - First order differential equation

Definition of the ODE

We would like to solve a problem with the following form :

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

We choose :

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

The solution of this first order differential equation is:

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

Forward Euler

The forward Euler scheme is define as:

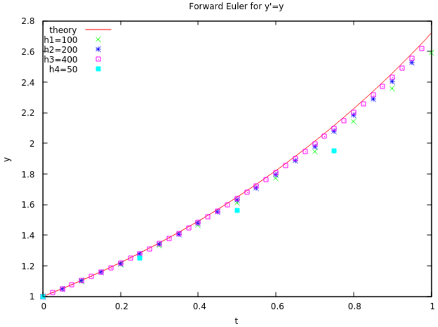
$$\begin{cases} y_{n+1} = y_n + hf(y_n, t = nh) \\ y_0 \quad given \end{cases}$$

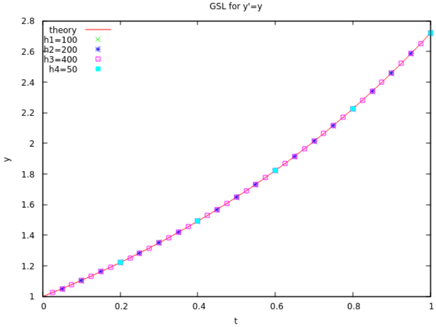
GSL - Runge Kutta 4

The GSL librabry uses several methods to solve an ODE, here we decide to use the Classical Runge Kutta (rk4);

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

Results


Forward Euler for y'=y
theory
h1=100
h2=200
h3=400
h4=50
y
t
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
0.2
0.4
0.6
0.8
1


GSL for y'=y
theory
h1=100
h2=200
h3=400
h4=50
y
t
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
0.2
0.4
0.6
0.8
1

Performance

And the performances are (for h= 1 000 000):

Code	performance
Explicit Forward Euler	14.35 s
GSL	15 s

3. Charged Particle

Definition of the problem

The problem to solve is:

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

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

Definition of the problem

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:

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

Runge kutta methods

General definition

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

We will use rk2, rk4 and rk8

Runge kutta methods

For the Runge Kutta 2:

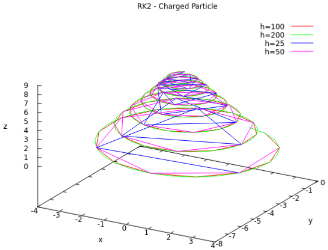
$$\begin{cases} y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2) \\ t_{n+1} = t_n + h \\ with : \\ k_1 = f(t_n, y_n) \\ k_2 = f(t_n + h, y_n + hk_1) \end{cases}$$

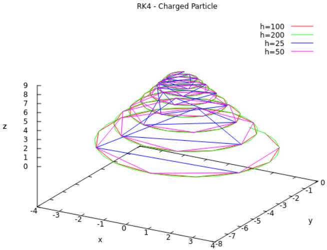
Runge kutta methods

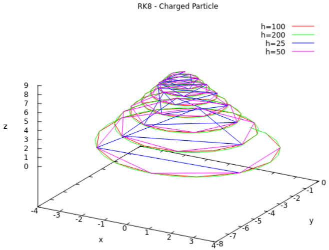
And finally for the Runge Kutta 8, Dormand Prince method:

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

Results


RK2 - Charged Particle
h=100
h=200
h=25
h=50
z
x
y


RK4 - Charged Particle
h=100
h=200
h=25
h=50
z
x
y


RK8 - Charged Particle
h=100
h=200
h=25
h=50
z
x
y

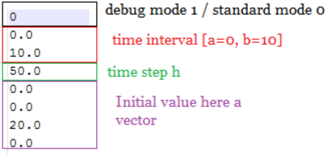
Performance

h= 50 000

Method	performance
rk2	2.83 s
rk4	15.36 s
rk8	10.88 s

4. Code's Presentation

Input file



For Euler and ODE GSL the inital value is a point, for xy trajectory and z trajectory it is a vector

make file: g++ compiler and use of gprof

```
#!bin/sh
# files

EXEC    := xy_trajectory
SRC     := $(wildcard *.cpp)
OBJ     := $(patsubst %.cpp, %.o, $(SRC))

# Options
CC              := g++
GSL_INCLUDE     := -I $$TACC_GSL_INC -I $$TACC_GSL_INC/gsl
LDFLAGS         := -L $$TACC_GSL_LIB
LDLIBS          := -lgsl -lgslcblas -limf

# Rules
$(EXEC) : $(OBJ)
	$(CC) $(LDFLAGS) $(LDLIBS) -g -pg -o $@ $^
%.o: %.cpp
	$(CC) $(GSL_INCLUDE) -g -pg -c $<

main.o mytools.o: mytools.h

.PHONY: clean neat echo
clean: neat
	$(RM) $(OBJ) $(EXEC)
neat:
	$(RM) $~ .*~
```

Output file

Outputfile	utility
outdebug	Display different loop values, GSL success etc ...
output	give numerical solution / data for our plots

5. Conclusion

Conclusion

GSL is a good tool to solve second order differential equation.
This project helped me to learn:

- How to use GSL
- How to use gprof
- Another use of gnuplot
- Be more confortable with makefiles