Differential Equations - Computational Practicum

Khaled Ismaeel

November 14, 2019

1 Analytical Solution

Rearranging gives

$$y' + 2y = y^2 e^x$$

The given equation is a Bernoulli equation with $p = 2$. In order to solve it, we shall first solve the complimentary equation:

$$y_c' + 2y_c = 0 \implies y_c = e^{-2x} \ (*)$$

Next we assume $y = y_c u$ for some u. Substituting gives

$$y_c' u + y_c u' + 2y_c u = e^x y_c^2 u^2$$

Substituting $(*)$ yields

$$u' y_c = e^x y_c^2 u^2 \implies \frac{du}{u^2} = e^x y_c dx = e^{-x} dx \implies u = \frac{1}{e^x + C}$$

And so we get

$$y = y_c u = \frac{e^{-2x}}{e^x + C} = \frac{e^{-x}}{Ce^x + 1}$$

And finally, substituting $y(x_0) = y_0$ and isolating C we get

$$C = \frac{e^{-x_0}/y_0 - 1}{e^{x_0}} \implies y = \frac{e^{-x}}{\frac{e^{-x_0}/y_0 - 1}{e^{x_0}} e^x + 1}$$

with a lost solution at $u = 0 \implies y = 0$

2 Numerical Methods Implementation

In this project, we used the C++ programming language under the C++17 standard. The Qt framework was used for implementing the graphical user interface, and therefore Qt Creator was our IDE of choice.

First of all, an abstract class:

```
class ode_solver;
```

which models an abstract differential equation solver, is provided. This class specifies the non-static pure virtual method:

```
virtual std::vector<point> solve_interval(real rhs(real, real),
point p0, real x, unsigned int precision) = 0;
```

which solves the initial value problem $y' = rhs(x, y)$ with $(x_0, y_0) =$ `p0` over the interval $[x0, \mathtt{x}]$ taking `precision` $+ 1$ number of samples. This function has also been defined as a brackets operator, so that object of this class are callable (invokable).

Class `ode_solver` is derived to:

```
template <unsigned int stages>
class explicit_runge_kutta_solver_base
:public ode_solver;
```

that models a solver using the Runge-Kutta numerical method with a fixed number of stages. This class maintains private data members holding the Butcher table of the method. The class provides protected setter functions, along with an override of `solve_interval` that implements the Runge-Kutta method. Note that this class is not meant to be used directly, as its setters are protected; this class is rather meant to be further derived to give concrete solution methods.

Three Class are derived from `explicit_runge_kutta_solver_base`, namely:

```
class euler_method_solver
:public explicit_runge_kutta_solver_base<1>;

class improved_euler_method_solver
:public explicit_runge_kutta_solver_base<2>;

class RK4_solver
:public explicit_runge_kutta_solver_base<4>;
```

These classes' constructors fill the Butcher tables with the appropriate values and and do not provide any other functionality to the user (except what it inherited). The UML diagram of the mentioned classes is provided above.

3 Interface and Results

The GUI is vertically divided into two halves, one for plotting the analytical and numerical solutions (and local errors) and one to plot the global errors

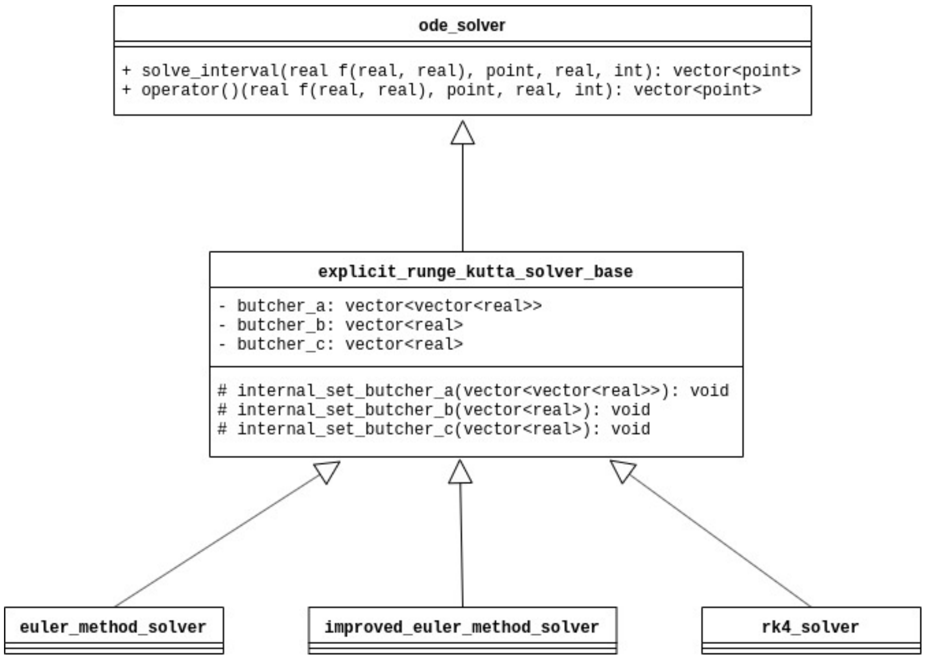

ode_solver
+ solve_interval(real f(real, real), point, real, int): vector<point>
+ operator()(real f(real, real), point, real, int): vector<point>
explicit_runge_kutta_solver_base
- butcher_a: vector<vector<real>>
- butcher_b: vector<real>
- butcher_c: vector<real>
# internal_set_butcher_a(vector<vector<real>>): void
# internal_set_butcher_b(vector<real>): void
# internal_set_butcher_c(vector<real>): void
euler_method_solver
improved_euler_method_solver
rk4_solver

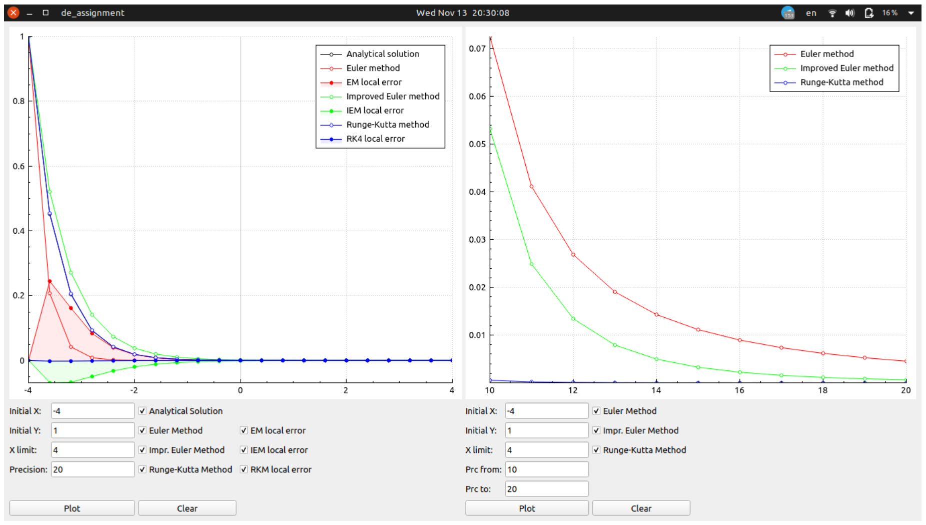
Figure 1: UML Class Diagram of the Numerical Solutions Classes

(deviation) against precision.

The solution part of the interface asks the user to input the initial value problem parameters, along with the desired solution range and precision. The user also chooses the desired graphs to plot (solutions and local errors) and the graphs are shown upon pressing ”Plot”.

As for the global error part of the interface, it asks the user for the initial value problem parameters and solution range, along with the desired range of precision to plot the deviation for. Again, the user chooses the desired graphs and will be shown upon pressing ”Plot”.

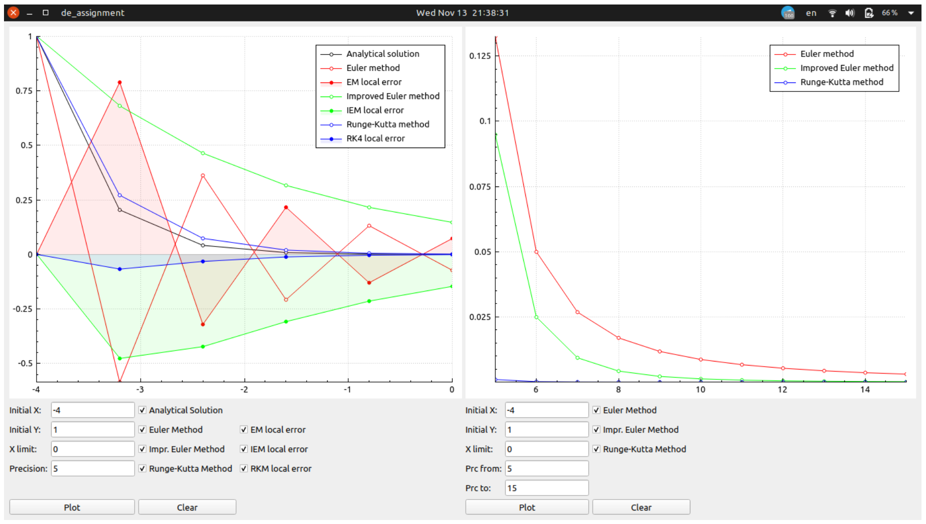
Project and report are available on github here.



Initial X: -4 ✓ Analytical Solution
Initial Y: 1 ✓ Euler Method ✓ EM local error
X limit: 4 ✓ Impr. Euler Method ✓ IEM local error
Precision: 20 ✓ Runge-Kutta Method ✓ RKM local error

Initial X: -4 ✓ Euler Method
Initial Y: 1 ✓ Impr. Euler Method
X limit: 4 ✓ Runge-Kutta Method
Prc from: 10
Prc to: 20

Figure 2: Interface screenshot with the input as required by the assignment



Initial X: -4 ✓ Analytical Solution
Initial Y: 1 ✓ Euler Method ✓ EM local error
X limit: 0 ✓ Impr. Euler Method ✓ IEM local error
Precision: 5 ✓ Runge-Kutta Method ✓ RKM local error

Initial X: -4 ✓ Euler Method
Initial Y: 1 ✓ Impr. Euler Method
X limit: 0 ✓ Runge-Kutta Method
Prc from: 5
Prc to: 15

Figure 3: Interface screenshot with low precision indicating the inaccuracy of Euler and improved Euler methods