# Computational Practicum

Ayhem Bouabid BS20-07 Variant 23

October 2021

## 1 Problem and Goal

we are considering the following initial value problem:

$$y' = \frac{\sqrt{y-x}}{\sqrt{x}} + 1$$
$$y(1) = 10.0$$

We want to find the analytical solution of the IVP in question as well as write a piece of software that for each of Euler, Improved Euler and Runge-kutta numerical methods

- approximates the analytical solution
- calculates the local error at each step on interval $[x_0, X] = [1.0, 15.0]$
- analyze the step's size effect on accuracy by calculating the largest local error for different step values

The program in question should be interactive enabling plotting graphs for each of the above-mentioned calculations. Moreover, The graphical user interface should enable the user to solve the same differential equation for different parameters values: $(x_0,\ y_0,\ X,\ N,\ n_0)$

## 2 Analytical solution

we are considering the following IVP:

$$y' = \frac{\sqrt{y-x}}{\sqrt{x}} + 1 \tag{1}$$
$$y(1) = 10.0 \tag{2}$$

This problem is defined only for $x > 0$ because of having $\sqrt{x}$ in the denominator and $y \geq x$ because of $\sqrt{y-x}$.

We proceed by noting that $y = x$ is a solution for $\big(1\big)$ $\forall x > 0$. $\big(2\big)$ excludes this solution. Since $\exists\ x_0$ such that $y(x_0) \neq x_0$ and $y$ is differentiable thus continuous, $\exists$ an interval $I$ such that $\forall\ x \in I$, we have y(x) $\neq x$

The argument represented below is based on the assumption that $x \in I$. Thus $x > 0$ and $y > x$.
we have

$$y' = \frac{\sqrt{y-x}}{\sqrt{x}} + 1 = \sqrt{\frac{y-x}{x}} + 1 = \sqrt{\frac{y}{x} - 1} + 1$$

Therefore

$$y' = \sqrt{\frac{y}{x} - 1} + 1 \tag{3}$$

we consider the following substitution

$$u = \frac{y}{x} \iff y = ux$$

substituting in $\big(3\big)$, we obtain

$$\big(ux\big)' = \sqrt{u-1} + 1 \tag{4}$$

We derive the left hand side to obtain

$$u'x + u = \sqrt{u-1} + 1 \tag{5}$$

We consider the following substitution

$$z = u - 1$$

we apply it to obtain

$$z'x + z = \sqrt{z} \tag{6}$$

$\big(6\big)$ can be transformed to a separable equation since $x > 0$ and $z > 1$

$$\frac{z'}{\sqrt{z} - z} = \frac{1}{x} \tag{7}$$

From $\big(7\big)$ we derive that

$$\int \frac{1}{\sqrt{z} - z} dz = \int \frac{1}{x} dx \tag{8}$$

We manipulate the left hand side to obtain

$$2 \times \int \frac{1}{2\sqrt{z}\sqrt{z} - 1} dz = \int \frac{1}{x} dx \tag{9}$$

which implies

$$-2 \times \log(|\sqrt{z}-1|) = \log(x) + C \iff \log(|\sqrt{z}-1|) = -\frac{\log(x)}{2} + C$$

Therefore

$$|\sqrt{z}-1| = -\frac{e^C}{\sqrt{x}} \iff \sqrt{z} = 1 + \frac{K}{\sqrt{x}}$$

where $K \in \mathbb{R}$ such that $\frac{K}{\sqrt{x}} + 1 \geq 0$

$$\sqrt{z} = 1 + \frac{K}{\sqrt{x}} \implies z = \Big[\frac{K}{\sqrt{x}} + 1\Big]^2$$

with $u = z + 1$ and $y = u \cdot x$ the general solution for $\big(1\big)$ is

$$y = x\Big[1 + \Big[\frac{K}{\sqrt{x}} + 1\Big]^2\Big]$$

we have $y > x \ \forall x > 0$ thus this solution is valid $\forall x > 0$
Now we need to determine the expression of constant $K$ in terms of $x_0$ and $y_0$

$$y_0 = x_0\Big[1 + \Big[\frac{K}{\sqrt{x_0}} + 1\Big]^2\Big] \implies y_0 - x_0 = x_0\Big[\frac{K}{\sqrt{x_0}} + 1\Big]^2$$

$$\implies \frac{y_0 - x_0}{x_0} = \Big[\frac{K}{\sqrt{x_0}} + 1\Big]^2 \implies \sqrt{\frac{y_0 - x_0}{x_0}} = \Big[\frac{K}{\sqrt{x_0}} + 1\Big]$$

$$\implies \sqrt{\frac{y_0 - x_0}{x_0}} = \frac{K + \sqrt{x_0}}{\sqrt{x_0}} \implies K = \sqrt{y_0 - x_0} - \sqrt{x_0}$$

From the above-mentioned argument we deduce that the general solution for the IVP in question is:

$$y = x\Big[1 + \Big[\frac{\sqrt{y_0 - x_0} - \sqrt{x_0}}{\sqrt{x}} + 1\Big]^2\Big]$$

taking $x_0 = 1$ and $y_0 = 10$, we obtain the solution to the 23-th variant

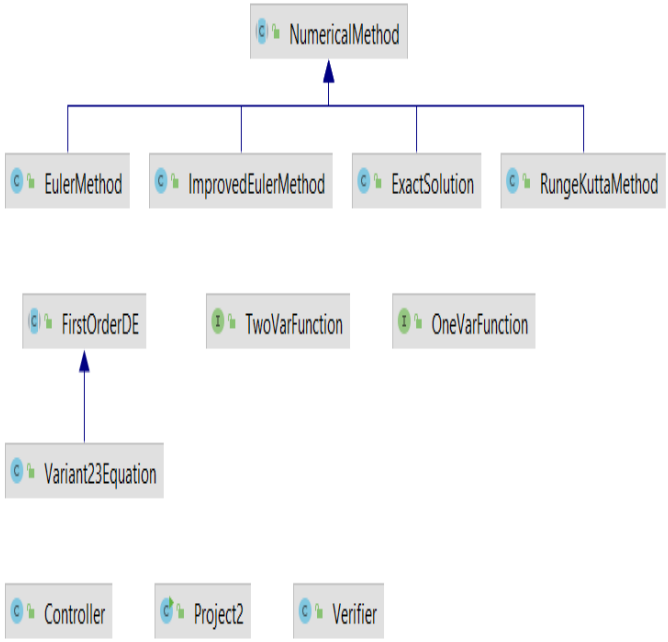
$$y = x\Big[1 + \Big[\frac{2}{\sqrt{x}} + 1\Big]^2\Big] = x\Big[1 + 1 + \frac{4}{x} + \frac{4}{\sqrt{x}}\Big] \implies y = 2x + 4 + 4\sqrt{x}$$

# 3 Coding Part

## 3.1 Technology Used

The Graphical user interface was developed using Java programming language with the JAVAFx library

## 3.2 UML Diagram




## 3.3 Application of SOLID principles

the piece of software was developed according to the SOLID principles

### 3.3.1 Single-responsibility Principle

Each class is created for a single purpose. The ***Verifier*** class makes sure that both format and mathematical restrictions are respected in the user-input. Each numerical method has its own class executing the exact calculations on data provided by a FirstOrderDE object passed as an argument. The firstOrderDE class stores the mechanism to calculate the exact solution.The ***Controller*** class user Verifier, OurEquation, and NumericalMethod classes to pass valid(Verifier), updated (OurEquation) and computed(NumericalMethod) data to the GUI.

#### 3.3.2 Open-Closed Principle

This principle is best illustrated by the ***Controller*** class. It is open for extension and closed for modification. We can extended this class to support additional feature of the user interface (such as real-time re-graphing when the user changes the input) without modifying the existing code.

#### 3.3.3 Liskov Substitution Principle

The relation between the Controller class and the different NumericalMethod subclass best illustrates this principle. The ***Controller*** classe treates **EulerMethod**,**ImprovedEulerMethod** and **RungeKuttaMethod** and even **ExactSolution** uniformly as ***NumericalMethod***

#### 3.3.4 Interface-segregation Principle

The program follows the ***MCV*** (Model, View, Control). The model is the result of the collaboration of ***OurEquation*** , ***NumericalMethod*** and ***FirstOrderDE*** classes. The View is represented as graphical user interface created in the sample.fxml file. As for the Control, it is assured by the ***Controller*** and ***Verifier*** classes.

#### 3.3.5 Dependency Inversion Principle

This principle is illustrated by the ***NumericalMethod*** abstract class, since it represents a high-level abstraction for all numerical methods without considering the concrete implementation of the method.

# 4 Code Snippets

The full code used in this program is beyond the scope of this report. For further verification, please feel free to use the following link:

## 4.1 The NumericalMethod abstract class

This class stores all the common fields and methods which avoids duplicated code. All resulting calculations are stored in the *XYCHART.Series<* BigDecimal,BigDecimal > fields. They are passed to the *Controller* class for display. The *BigDecimal* class ensures an extreme degree of accuracy to decrease round-off errors.

```java
public abstract class NumericalMethod {
    protected  XYChart.Series<BigDecimal,BigDecimal> approximations;
    protected  XYChart.Series<BigDecimal,BigDecimal> localErrors;
    protected  XYChart.Series<BigDecimal,BigDecimal> globalErrors;
    protected BigDecimal maxLocalError;

    public NumericalMethod(){
        approximations = new XYChart.Series<>();
        localErrors = new XYChart.Series<>();
        globalErrors = new XYChart.Series<>();
        maxLocalError = new BigDecimal( val: "0");
    }

    public XYChart.Series<BigDecimal, BigDecimal> getApproximations(FirstOrderDE equation) {
        makeCalculations(equation);
        approximations.setName(getMethodName());
        return approximations;
    }

    public XYChart.Series<BigDecimal, BigDecimal> getLocalErrors(FirstOrderDE equation) {
        makeCalculations(equation);
        localErrors.setName(getMethodName());
        return localErrors;
```

```java
public XYChart.Series<BigDecimal, BigDecimal> getGlobalErrors(FirstOrderDE equation, String n0, String N) {

    // convert the string parameters to integers
    int N0 = Integer.parseInt(n0);
    int NF = Integer.parseInt(N);
    // initialize the series
    globalErrors = new XYChart.Series<>();
    // for each step store the pair: grid size, the largest local error
    for(int i = N0; i <= NF;i++) {
        // set the grid size to the i
        equation.updateGridSize(String.valueOf(i));
        // make calculations
        makeCalculations(equation);

        // makeCalculations errors will store the largest local error in the maxLocalError field
        // the latter will be added to the globalErrors series
        globalErrors.getData().add(new XYChart.Data<>(new BigDecimal(i),maxLocalError));
    }
    // set the name to be displayed on the user interface
    globalErrors.setName(getMethodName());
    return globalErrors;
}

public abstract void makeCalculations(FirstOrderDE equation);

public abstract String getMethodName();
```

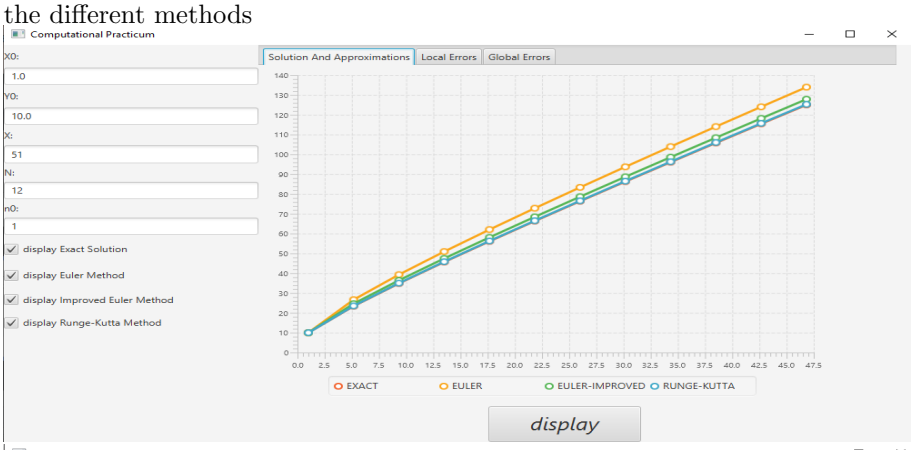
## 4.2 EulerMethod Class

```java
public void makeCalculations(FirstOrderDE equation) {
    //initialize the maxLocalError field to 0
    maxLocalError = new BigDecimal( val: "0");

    approximations = new XYChart.Series<>();
    localErrors = new XYChart.Series<>();
    // the intermediateValue
    BigDecimal currentX = equation.x0;
    // the approximation of the intermediate step
    BigDecimal currentY = equation.y0;
    // the difference between two consecutive intermediate values
    BigDecimal step = (equation.xMax.subtract(equation.x0)).divide(new BigDecimal(equation.N),SCALE, RoundingMode.CEILING);

    BigDecimal localError,exactImage;

    // while currentX is less than xMax
    while(currentX.compareTo(equation.xMax)<=0){
        // get the image of the intermediate value
        exactImage = equation.getExactValue(currentX);

        // calculate the local error
        localError = (currentY.subtract(exactImage)).abs();

        //updating the value of the maximum local error
        maxLocalError = maxLocalError.max(localError);
```

```
            // add it to the corresponding series
            localErrors.getData().add(new XYChart.Data<>(currentX,localError));

            // add the pair of intermediate value and approximation to the series
            approximations.getData().add(new XYChart.Data<>(currentX,currentY));

            // temp1  =  f(xi,yi)
            BigDecimal temp1 = equation.getRightHandSideValue(currentX,currentY);

            // assign currentY according to : y(i+1) = y(i) + (step) h * f(xi,yi)
            currentY = currentY.add(step.multiply(temp1));

            // increment intermediate value
            currentX = currentX.add(step);
        }
    }

    @Override
    public String getMethodName() { return METHOD_NAME; }
```

# 5 Resulting Graphs

The value $X = 51$ was chosen to illustrate the difference in accuracy between the different methods



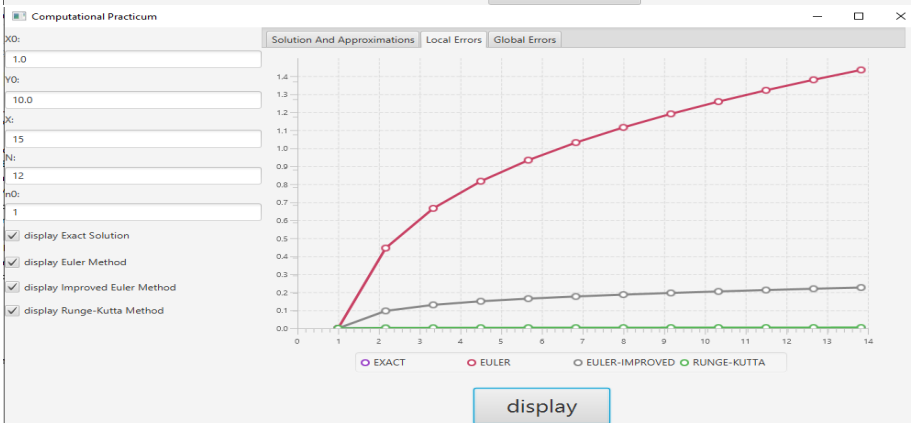




Figure 1: The effect of the grid size with $N = 100$, $n_0 = 1$ and $n_0 = 80$

# 6 Analysis And Conclusion

For a fixed $N$, the differences between the approximations are determined by the value of parameter $X$. For small values $X \leq 15$ the different graphs are

almost superposed. The gaps increase significantly as the value of $X$ increases.Yet, the gap between Runge-Kutta's approximation and the exact solution is almost indistinguishable as illustrated in the first figure in previous section. Such remarks are supported by the **LocalError** graphs. The values of LTE increase for each method along the interval $\left[x_0,\, X\right]$ with significantly different rates: Euler method can exceed a LTE of 1.4, while the Runge-Kutta method remains too small to be displayed in the same scope as Euler method as illustrated in the second figure.

As for the Global errors the last two illustrations demonstrate that, with large values of grid size, all the methods converge to the exact solution. The last figure provide more accurate estimation where the Euler method reaches a difference less than 0.175 while both improved Euler and Runge-kutta methods decrease the approximation error to less than 0.2. The full potential of the user interface is put forth when displaying each method individually, which is beyond the scope of this report. Yet, this does not prevent us from sharing the results. as The Improved method reaches an accuracy of less than 0.04 while Runge-Kutta lands on a difference of 0.00009

In the light of the above-mentioned results, we conclude that each numerical method, with sufficiently small step size, converges to the exact solution. On the other hand, for fixed parameters $\left(x_0,\, y_0,\, X,N\right)$, Runga-kutta, Improved Euler and then Euler methods are sorted in decreasing order of precision.