DIFFERENTIAL EQUATIONS
COMPUTATIONAL PRACTICUM

Task Week 10

Done by Egor Osokin (B18-04)

Innopolis University 2019

Problem statement:

$y' = 5 - x^2 - y^2 + 2xy,\ y(0) = 1,\ x \in (0,\ 20)$

Exact solution of IVP:

This is the first-order nonlinear ordinary differential equation.

We can solve as Ricatti's equation:

1. Complete the square
   $\frac{dy(x)}{dx} = -(y(x) - x)^2 + 5$
2. Let $v(x) = -x + y(x),\ \Rightarrow\ y(x) = x + v(x)$ *and* $\frac{dv(x)}{dx} + 1 = -v(x)^2 + 5$
3. Substitute equations from p. 2 into p. 1:
   $\frac{dv(x)}{dx} = -v(x)^2 + 4$
4. Now let's solve for $\frac{dv(x)}{dx}$
5. By variable separation, we get
   $\int \frac{dv(x)}{-v(x)^2+4} = \int dx = x$
6. Evaluate the integral
   $\frac{1}{4} ln \frac{2+v(x)}{2-v(x)} = x + c_1,$ *where* c_1 *is an arbitrary constant*
7. Solving for v(x), we come to
   $v(x) = \frac{2(e^{4(x+c_1)}-1)}{e^{4(x+c_1)}+1} = 2 + \frac{1}{c_2 e^{4x} - 0.25}$
8. Returning to point 2, we get
   $y(x) = v(x) + x = \frac{1}{c_2 e^{4x} - 0.25} + x + 2,$ *where* c_2 *is an arbitrary constant*
9. Getting back to IVP y(0)=1, we get $c_2 = -\frac{3}{4}$, so the correct solution for our IVP is $y(x) = \frac{1}{-0.75 e^{4x} - 0.25} + x + 2$

There are no points of discontinuity in a given range.

Talking about convergence, when $h = \frac{X - x_0}{n} > 1$, the function doesn't converges. In my application, there are just no graphs for this case.

Implementation

For the implementation, I used Kotlin programming language with TornadoFX framework for plotting and GUI elements.

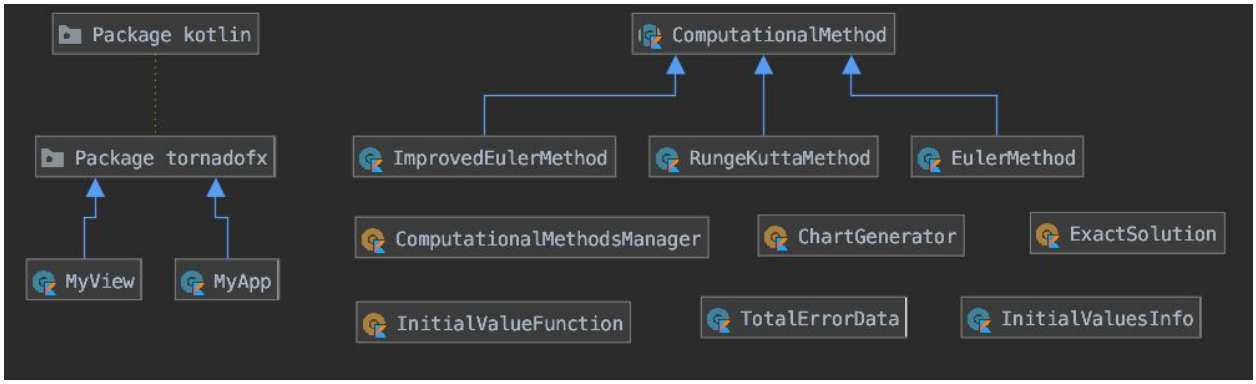
There are 3 files with source code: *main.kt, utils.kt and computational_methods.kt.*

The *main.kt* is the file with the structure of GUI, which defines the schema and objects.

The *computational_methods.kt* contains all classes which relates to the computational methods themselves, so there are basically EulerMethod, ImprovedEulerMethod, RungeKuttaMethod. Also, there are ComputationalMethod abstract class, which is the parent of every class which implements some method, and ComputationalMethodsManager, which manages all the classes (this logic will be discussed later).

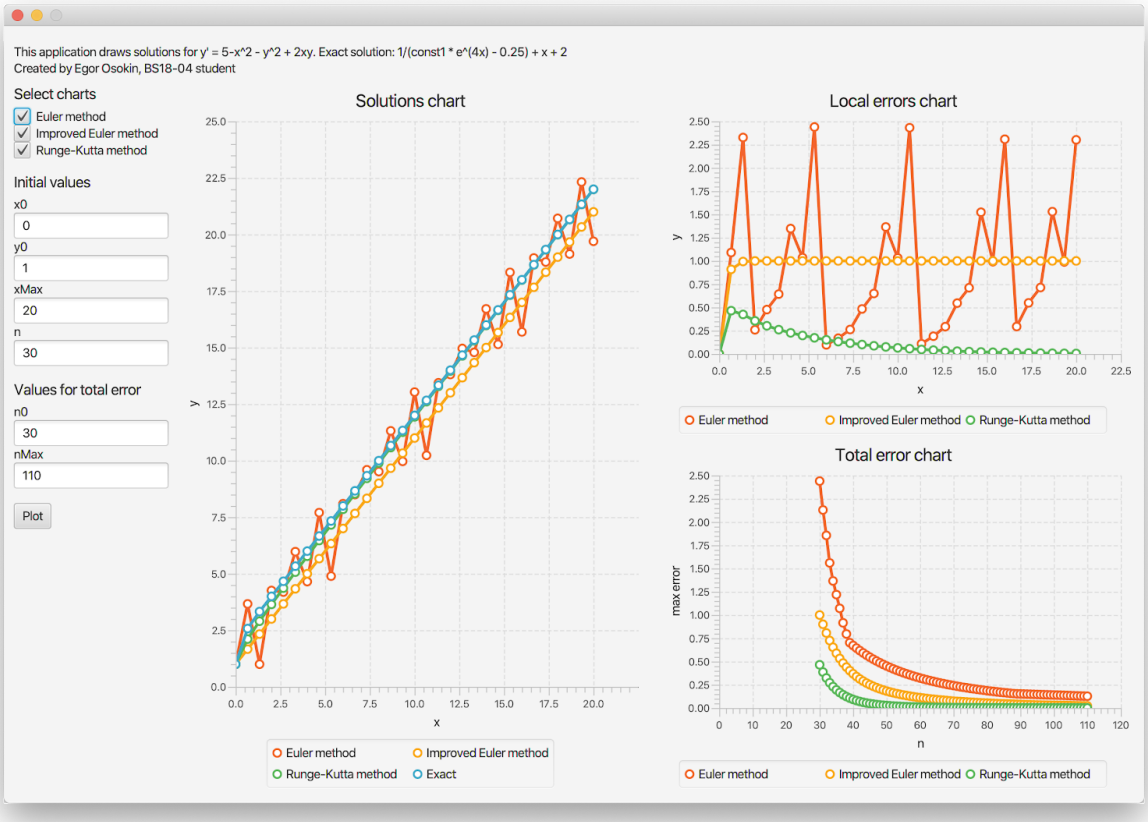
File *utils.kt* contains some useful classes which are not methods and not GUI.

Here is the UML diagram of the classes



Link to the source code: https://github.com/Birdi7/gui-ode-solver

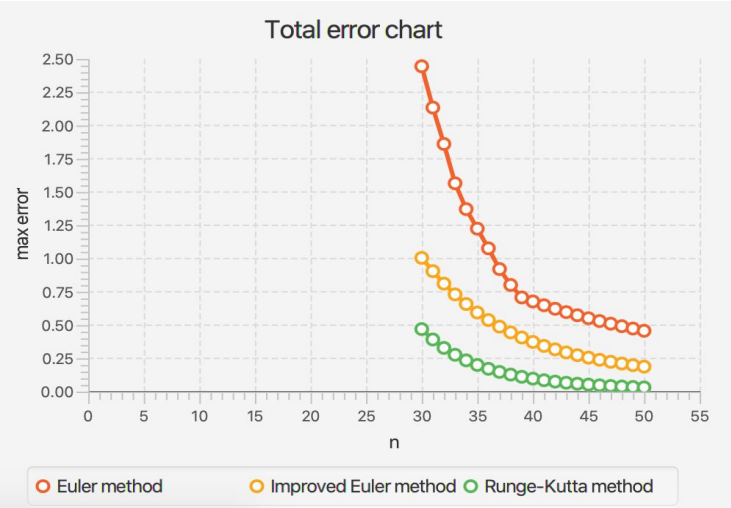
Here is the screenshot of the plots for the given IVP

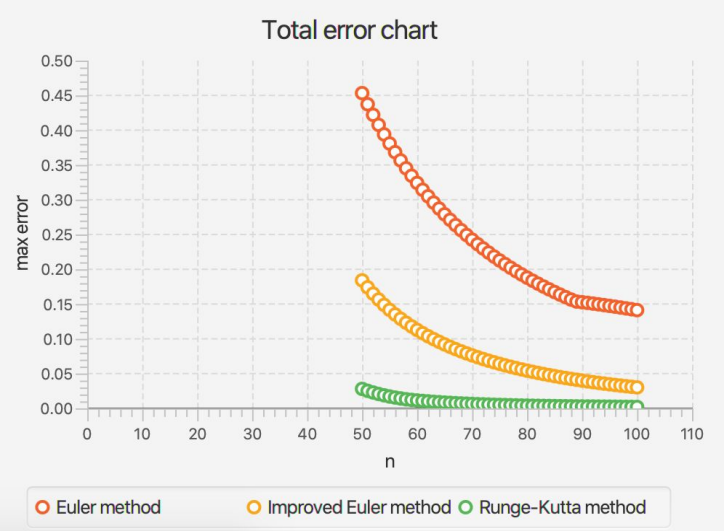


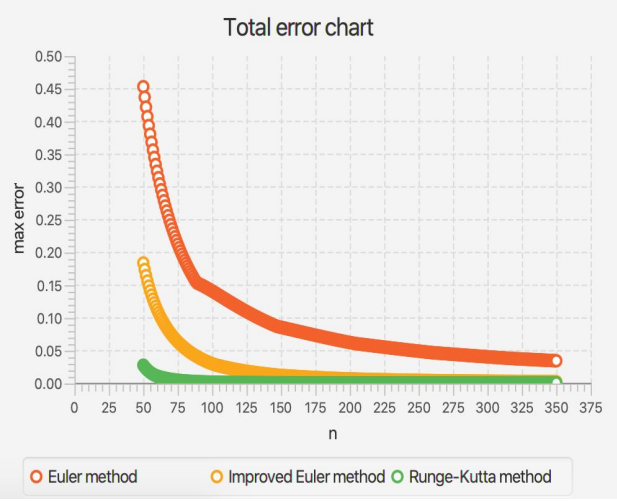
On the left part, there are checkboxes for choosing which graphs should be plotted, and input textboxes with labels for user to type the input values. The upper ones change the parameters for the Solutions and Local errors chart, and the last section, Values

Right part consists of 3 graphs: Solutions chart, which plots the graphs of exact and numerical solutions, Local error chart plotting the corresponding values for every method on every step of calculation, and the last one, Total error chart, prints the maximum error which occurred during the calculations for all N's in given range [n0, nMax].

Here are the Total error graph with different input ranges. As we can see, the Runge-Kutta method is the most accurate:







Here are the functions, which computes the actual values for methods,
In the following order:
Euler method:

Improved Euler method:

```
override fun computeFor(initialValues: InitialValuesInfo): MutableMap<Double, Double> {
    val result = mutableMapOf<Double, Double>()
    var currentX = initialValues.x0
    var currentY = initialValues.y0
    result[currentX] = currentY

    for (i in 0 until initialValues.numberOfSteps) {
        val h = initialValues.h
        val k1 = initialValues.initialFunction.computeFor(currentX, currentY)
        val deltaY = initialValues.initialFunction.computeFor( x: currentX + h / 2.0,  y: currentY + h / 2.0 * k1)
        currentY += h * deltaY
        currentX += initialValues.h
        result[currentX] = currentY
    }
    return result
}
```

Runge-Kutta:

```
override fun computeFor(initialValues: InitialValuesInfo): MutableMap<Double, Double> {
    val result = mutableMapOf<Double, Double>()
    var currentX = initialValues.x0
    var currentY = initialValues.y0
    result[currentX] = currentY

    for (i in 0 until initialValues.numberOfSteps) {
        val h = initialValues.h

        val k1 = initialValues.initialFunction.computeFor(currentX, currentY)
        val k2 = initialValues.initialFunction.computeFor( x: currentX + h / 2,  y: currentY + h / 2 * k1)
        val k3 = initialValues.initialFunction.computeFor( x: currentX + h / 2,  y: currentY + h / 2 * k2)
        val k4 = initialValues.initialFunction.computeFor( x: currentX + h,  y: currentY + h * k3)

        currentY += h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
        currentX += initialValues.h
        result[currentX] = currentY
    }
    return result
}
```

Let's discuss the classes and the most interesting moments of the code
I used few auxiliary classes:

- MyApp, MyView - classes which are children of App, View classes form TornadoFX which is used for the GUI
- InitialValueFunction - represent the the IVP statement and is used for calculating value of y' depending on values of x & y.
- InitialValuesInfo and TotalError - data classes with all information about initial values and input for total error
- ExactSolution - represents the function which is the exact solution.
- ChartGenerator - generate all charts which are used in the app

The main classes are abstract ComputationalMethod, its children and ComputationalMethodsManager. In details, the manager classes is designed to use the classes which are children of ComputationalMethod, without explicit reference to them. In particular, it initializes the children, and based on the name of method returns the values of either x's and y'x, local errors or total errors.