DIFFERENTIAL EQUATION

COMPUTATIONAL PRACTICUM

Done by Bekzhan Talgat (BS18-05)

Innopolis University 2019

Problem statement:

$dy/dx = (2 - y^2) / (2yx^2)$ $y(1) = 1$; $x \in [1, 6]$

Exact solution of IVP:

$dy/dx = (2 - y^2) / (2yx^2)$ **$x \neq 0$**   **$y \neq 0$**

1. $(- (y^2 - 2)\, dy) / (2y) = dx / x^2$
2. $- \ln |y^2 - 2| = c - 1/x$
3. $c + 1/x = \ln |y^2 - 2|$
4. $e^{c + 1/x} = y^2 - 2$
5. $y = \sqrt{(2 + e^{c + 1/x})}$

Here shown that: $c = \ln (y^2 - 2) - 1/x$
To solve IVP and find c : $y^2 > 2$
IVP is not solvable for y=1

But it is possible to solve this problem with numerical methods methods such as Euler's method, Improved Euler's method, and Runge-Kutta method

Implementation

For implementation, I used JavaFX programming language with Scene Builder for making GUI elements

There are 8 code files: Main.java, Controller.java, NumericalMethods.java, EulerMethod.java, ImprovedEuler.java, RungeKuttaMethod.java sample.fxml, style.css

Description of classes and interesting moments of the code

1. Main.java initializes application

2. Controller.java has the main logic of the application, defines all functionalities of every entity in application

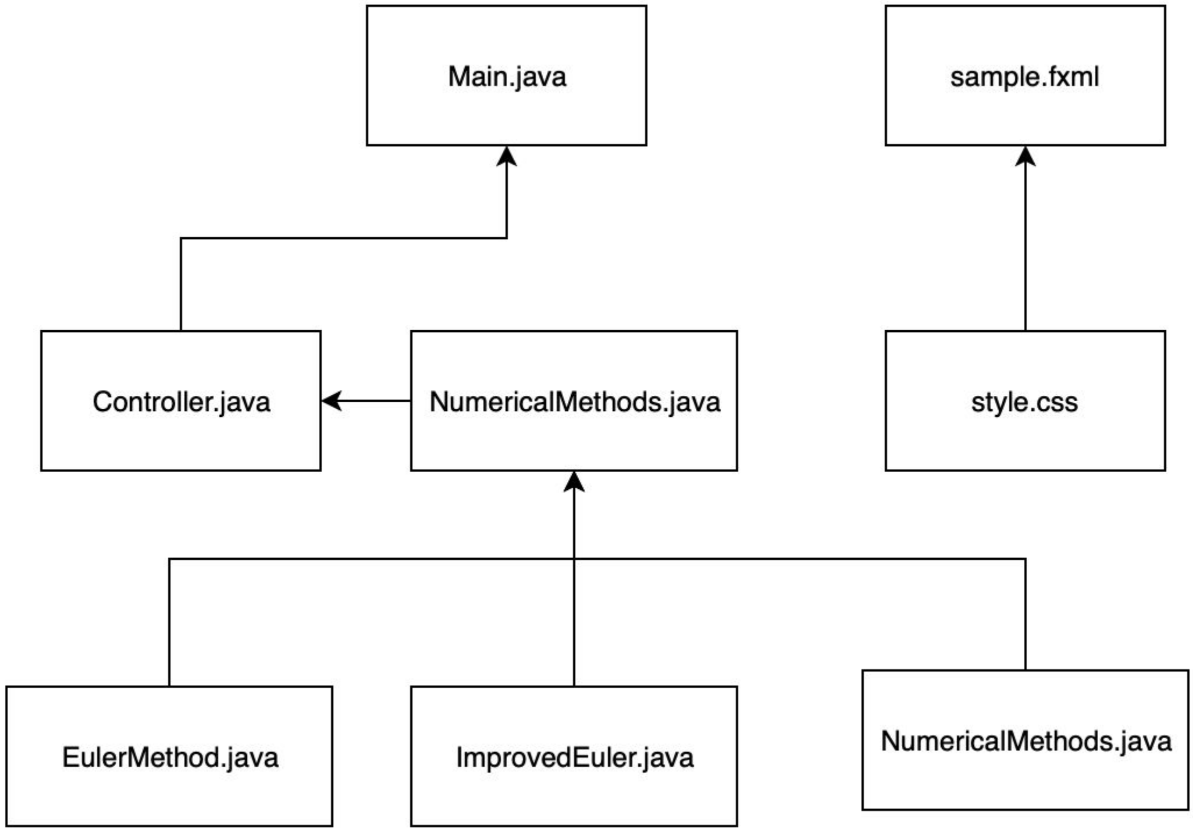
3. NumericalMethods.java is abstract class that contains exact solution and total error methods for differential equation

4. EulerMethod.java, ImprovedMethod.java, and RungeKuttaMethod.java are extend NumericalMethods.java and each class has own way of implementation of the abstract methods getGraph() and TotalError();

5. sample.fxml is contains code for GUI

6. style.css is like constraint for sample.fxml and responsible for design

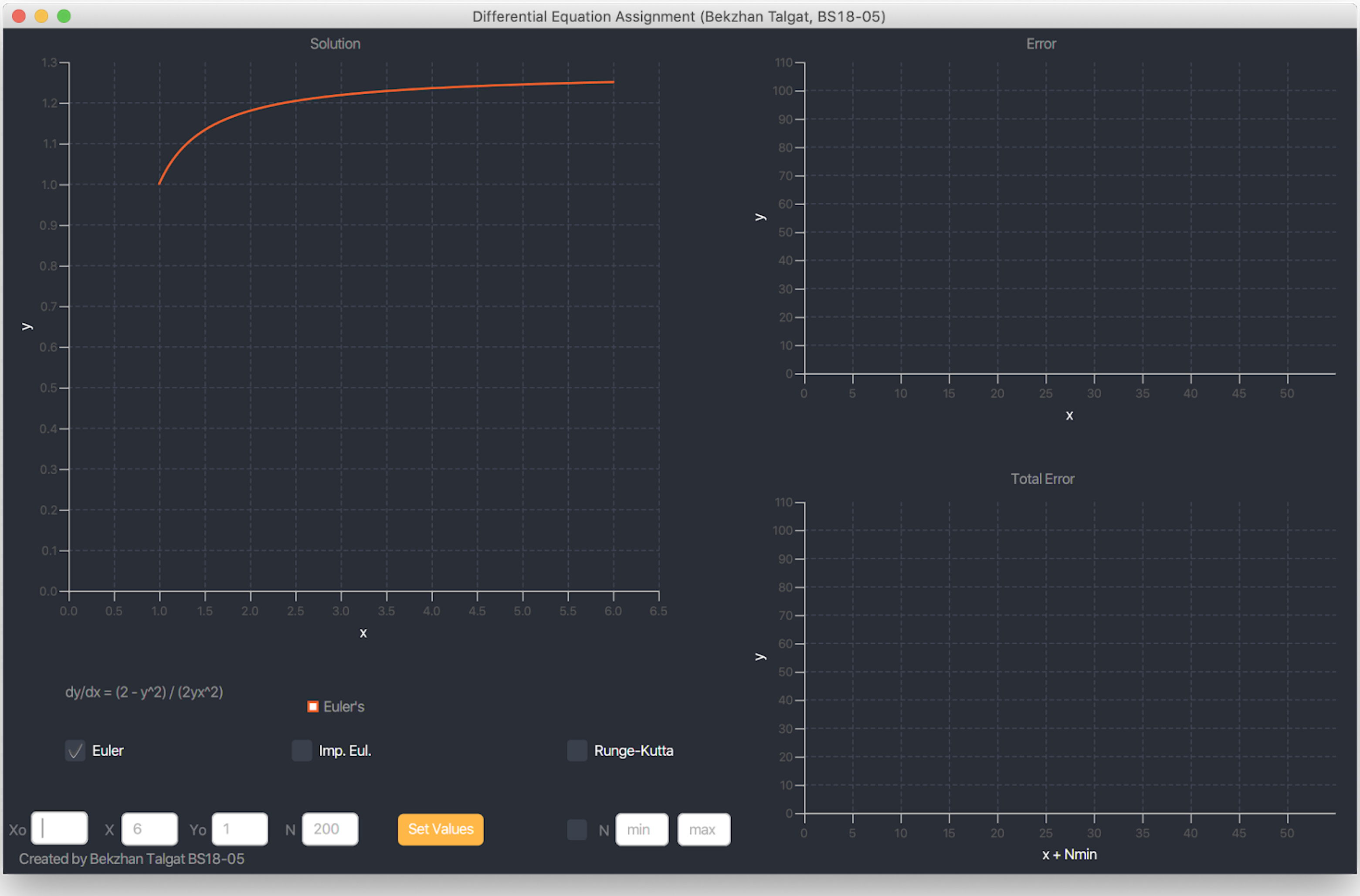
UML of code files:



Link to the source code:
https://github.com/Beka-13/DifferentialEquation/tree/master/DifferentialEq

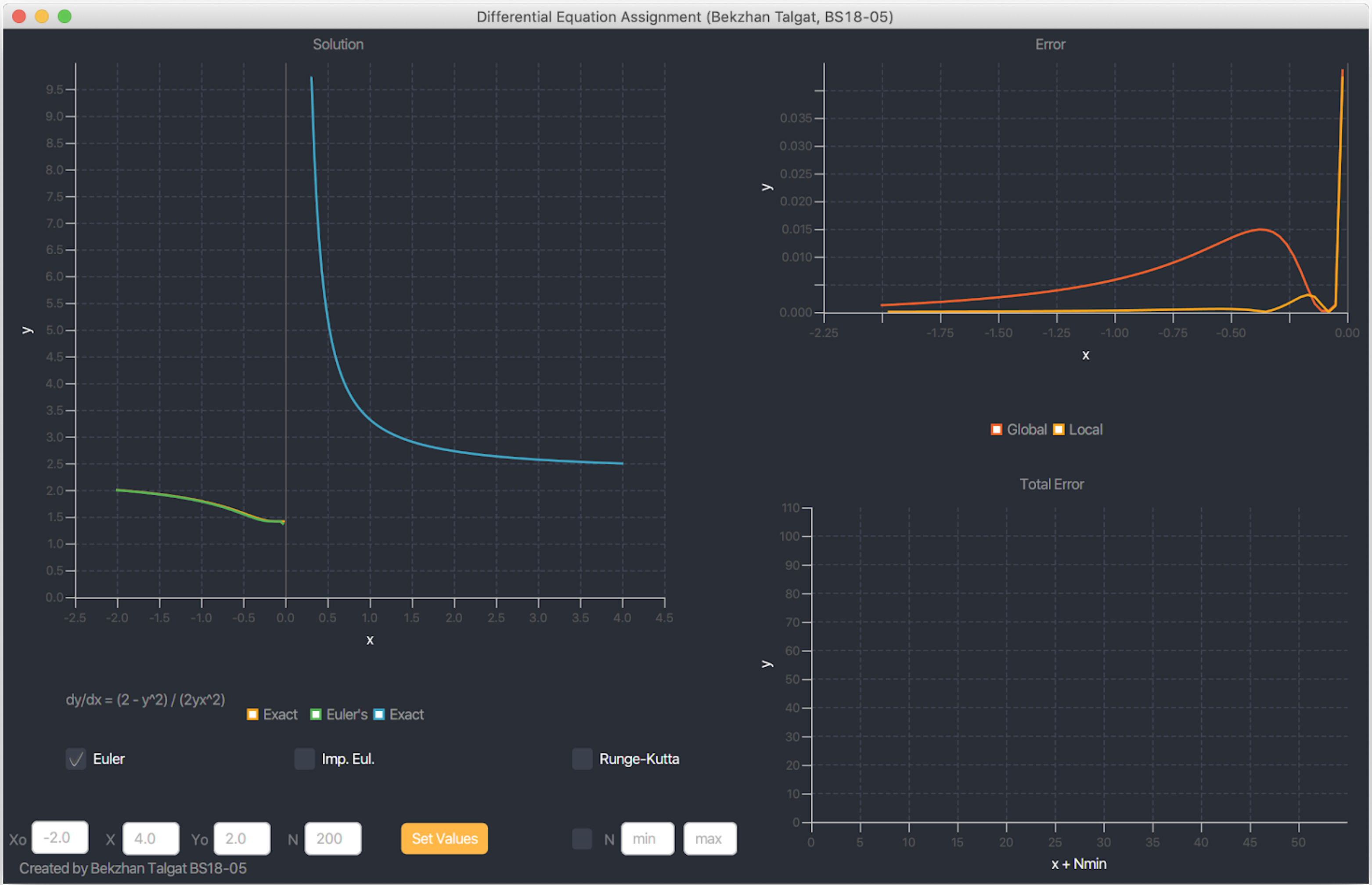
Here some screenshots of the application
On the right side set Solution graph, where plot graph of Exact solution with one numerical method solution. Which numerical method to illustrate you can choose with three checkboxes under the Solution graph: **Euler, Imp. Eul.**(Improved Euler's method)**, Runge-Kutta**
Under the checkboxes located six Text fields with Set value button.
Order of Text fields is following: x_0, X, y_0, N, N_{min}, N_{max}.
Near N_{min}, N_{max} is set N-checkbox to plot Total error graph, which is located on the right-bottom side of the application. Last thing to mention

is Error graph, which is located on the right-top side, on what global and local errors are plotted



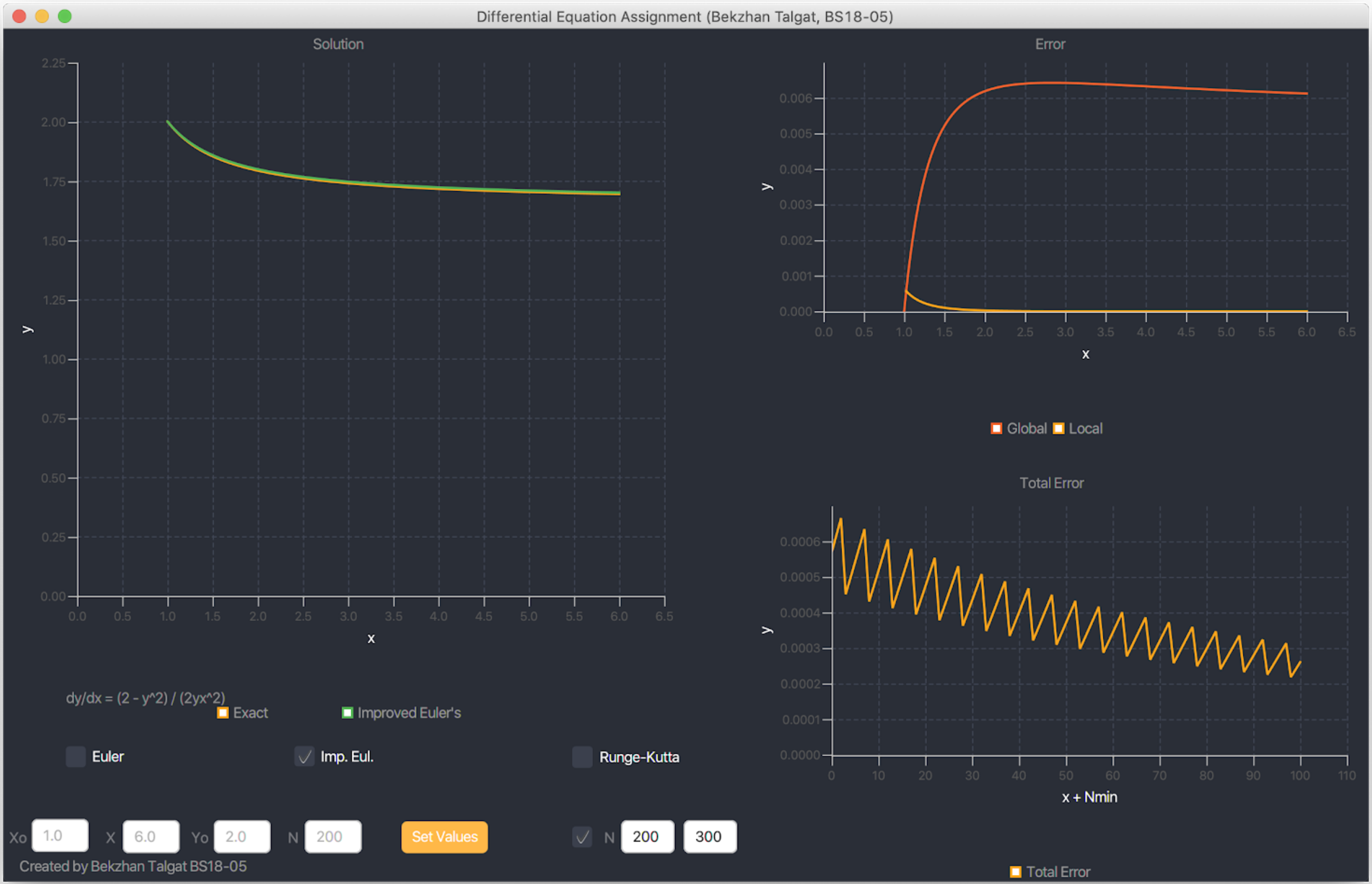
Here shown only graphic of a Euler's method, because initial value of **y = 1**. To find exact solution **y > √2**. As exact solution is abcent, there are no global and local errors. $x_0 = 1$, $y_0 = 1$, X = 6, N = 200

Under the graph of solutions there are three checkboxes: Euler, Imp. Eul.(Improved Euler), and Runge-Kutta; to display graph of these methods with Exact solution

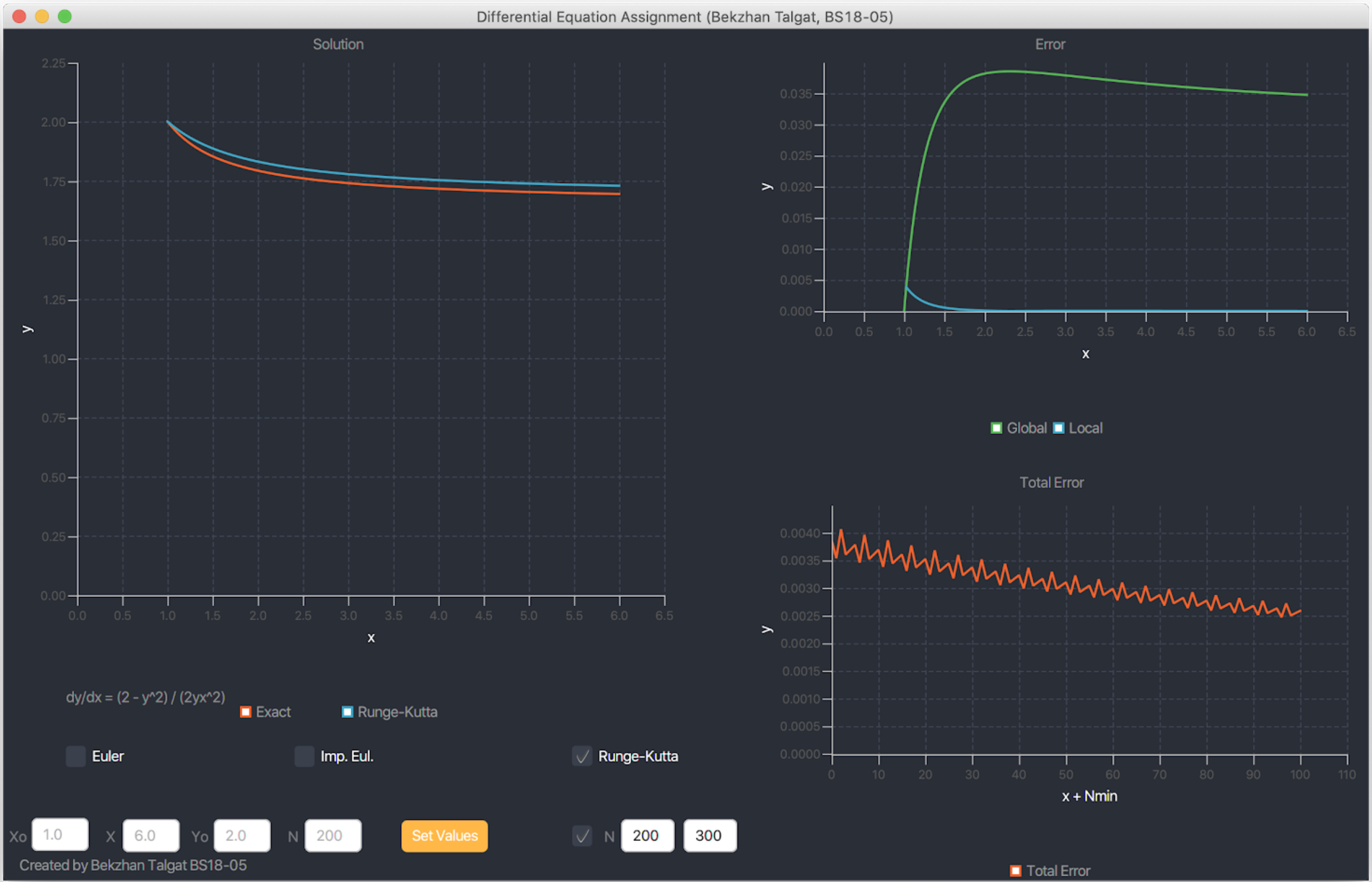


This example shows graph with disjoint point **x=0**

$x_0 = -2$, $y_0 = 2$, $X = 4$, $N = 200$



This example shows graph of Improved Euler method with Exact solution. x_0 = 1, y_0 = 2, X = 6, N = 200
On the right-top side illustrated global and local errors. Also, as N checkbox is turned on, on the right-bottom side displayed total error from N_{min}=200 and N_{max}=300



In this example on the solution graph demonstrated Runge-Kutta method with Exact solution. $x_0 = 1$, $y_0 = 1$, $X = 6$, $N = 200$

Also, shown global error with local error, and total error with $N_{min}=200$ and $N_{max}=300$

Here are the main computational methods of each numerical method
Euler's method, Improved Euler's method, and Runge-Kutta method

```
private double EulerEq(double x, double h, double y) {
    if (x== 0 || y == 0) { return CONST; }
    else { return (y + h*dydx(x,y)); }
}
```

```
private double ImEulerEq(double x, double h, double y) {
    double k1 = dydx(x, y);
    double k2 = dydx((x+h), (y+(h*k1)));

    if (k1 == CONST || k2 == CONST){ return CONST; }
    else { return (y + (h/2)*(k1+k2)); }
}
```

```
private double RungeKuttaEq(double x, double h, double y) {
    double k1 = dydx(x,y);
    double k2 = dydx((x + h/2),(y + k1/2));
    double k3 = dydx((x + h/2),(y + k2/2));
    double k4 = dydx((x + h),(y + k3));

    if (k1 == CONST || k2 == CONST || k3 == CONST || k4 == CONST) { return CONST; }
    else { return (y + (h/6) * (k1 + 2*k2 + 2*k3 + k4)); }
}
```