Computational Practicum Report

– By *Parth Kalkar*

***Objective**: Implementation of Euler, Improved Euler and Runge Kutta method for the given variant and comparison of error between them plotted in graph with capability of providing desired number of grids.*

Variant 3: $y' = \frac{4}{x^2} - \frac{y}{x} - y^2$

Or, $y' + y^2 + \frac{y}{x} = \frac{4}{x^2}$

To solve this task we will use the Riccati approach of differential equations.

***General Riccati Equation:** The Riccati equation is one of the most interesting nonlinear differential equations of first order. It is written in the form:*

$$y' = a(x) * y + b(x) * y^2 + c(x),$$

Where a(x), b(x), c(x) are continuous functions of x.

To solve such kind of equation we must intuitively seek the particular solution and proceed, in our case we get the particular solution in the form:

$y = \frac{c}{x}$ *Which implies* $y' = -\frac{c}{x^2}$

Substituting this into the equation, we obtain:

$$-\frac{c}{x^2} + \frac{c}{x^2} + \frac{c^2}{x^2} = \frac{4}{x^2}$$

$c^2 = 4$, *Therefore* $c = \pm 2$

To proceed further we can take any value of c, let us choose c = 2, now as the particular solution is known we make the replacement:

$$z = \frac{1}{y - \frac{2}{x}} \quad Or,\ y = \frac{1}{z} + \frac{2}{x}$$

$$And,\ y' = -\frac{z'}{z^2} - \frac{2}{x^2}$$

Now substituting this into original Riccati equation:

$$-\frac{z'}{z^2} - \frac{2}{x^2} = \frac{4}{x^2} - \frac{\left(\frac{1}{z} + \frac{2}{x}\right)}{x} - \left(\frac{1}{z} + \frac{2}{x}\right) * \left(\frac{1}{z} + \frac{2}{x}\right)$$

Simplifying this we get:

$$-\frac{z'}{z^2} - \frac{2}{x^2} = \frac{4}{x^2} = \frac{1}{2 * x} - \frac{2}{x^2} - \frac{1}{z^2} - \frac{4}{z * x} - \frac{4}{x^2}$$

So cancelling out the terms:

$$-\frac{z'}{z^2} = \frac{5}{z * x} - \frac{1}{z^2} \quad Or,\ z' = 5 * \frac{z}{x} + 1$$

Solving this differential equation in z with respect to variable, we get:

$$z = c * x^5 - \frac{x}{4}$$

Resubstituting to find y, we get:

$$y = \frac{1}{c * x^5 - \frac{x}{4}} + \frac{2}{x}$$

Solving the initial value problem with initial condition of (x0, y0), we get:

$$y0 - \frac{2}{x0} = \frac{1}{c * x0^5 - \frac{x0}{4}}$$

Simplifying further we get:

$y0 * x0 - 2 = \frac{1}{c*x0^4 - \frac{1}{4}}$ *(Here, x0! =0)*

$(y0 * x0 - 2) * \left(c * x0^4 - \frac{1}{4}\right) = 1$ *Or* $c * x0^4 - \frac{1}{4} = \frac{1}{y0*x0-2}$

Hence, $c = \frac{\frac{1}{y0*x0-2} + \frac{1}{4}}{x0^4}$

Substituting the value of c back in our general solution of the differential equation, we get:

$$y = \frac{1}{\frac{\frac{1}{y0 * x0 - 2} + \frac{1}{4}}{x0^4}) * x^5 - \frac{x}{4}} + \frac{2}{x}$$

Simplifying our solution with initial values of (x0, y0) = (1, 0), we get:

c = -1/4 or -0.25

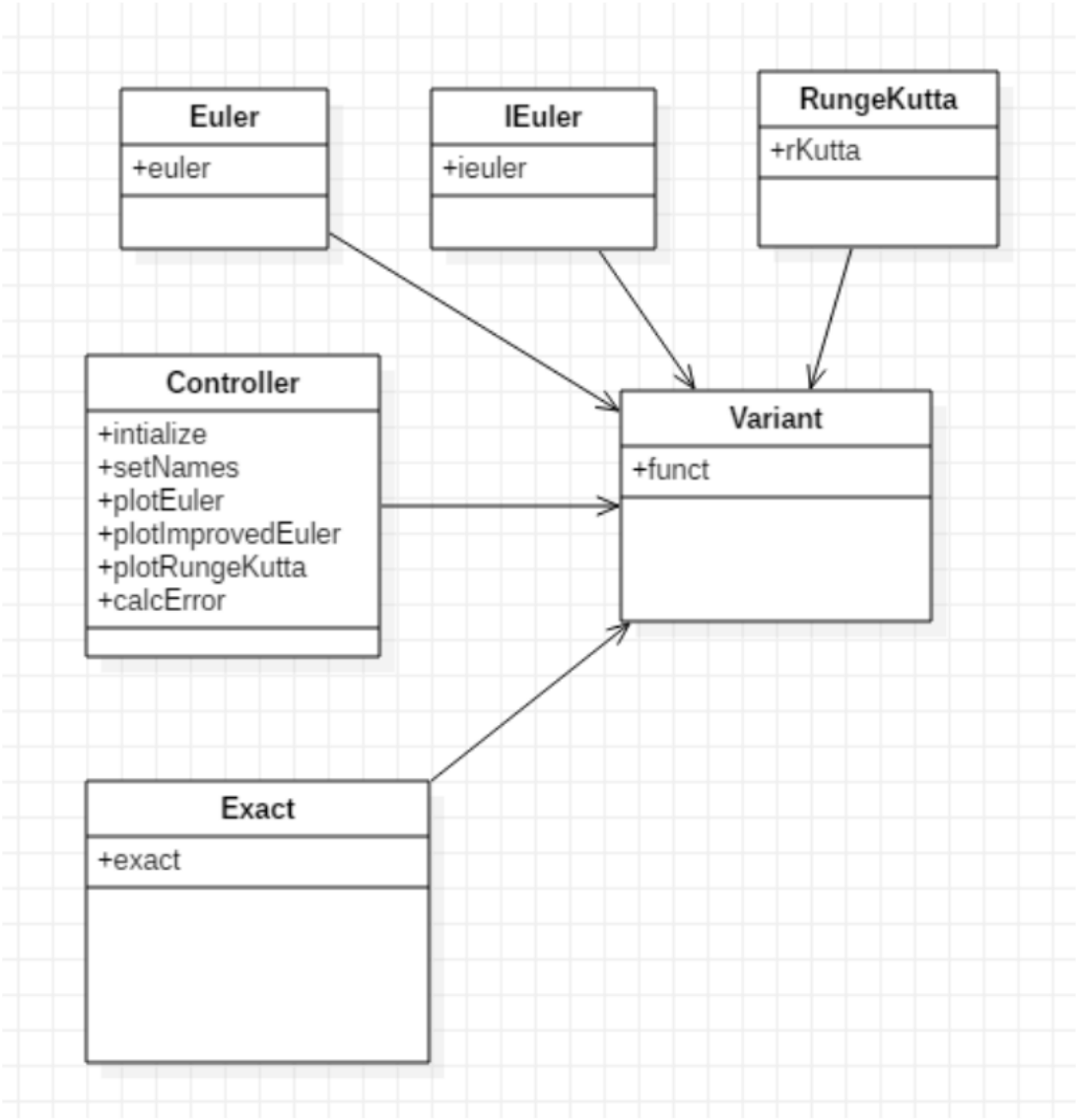
Plugging this c in our equation, we get:

$y = \frac{1}{-0.25*x^5 - \frac{x}{4}} + \frac{2}{x}$

This is the particular solution of our equation. (Here x = 0 is a discontinuity and the function has defined value for it)

The above section was the ***analytical solution*** *of our given differential equation.*

UML Diagram



Class Variant has the original function given for this task.

Class exact, euler, ieuler and rKutta have the implementation of exact solution, euler method, improved euler method and runge-kutta method to solve the differential equation respectively.

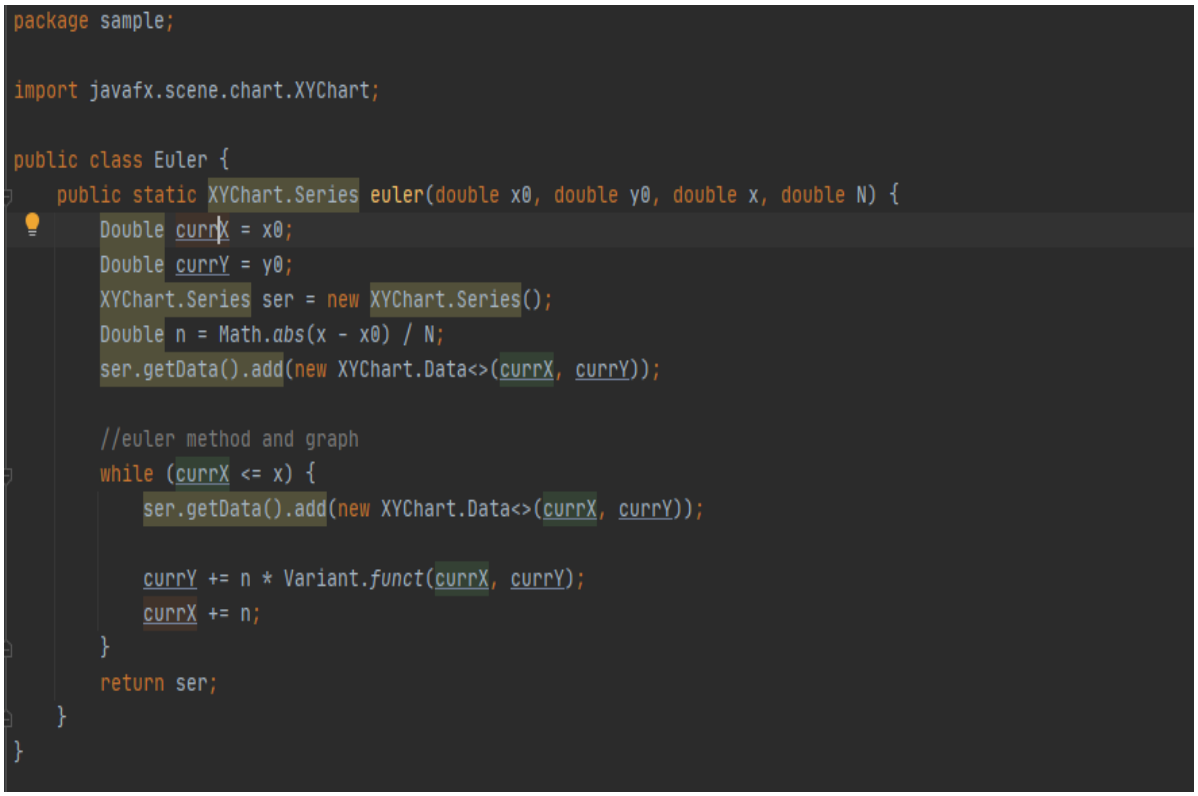
Class controller have the methods to calculate the errors (Local truncation error and Global truncation) and the methods to plot the graphs of the above-mentioned methods for the given differential equation.

Exact solution

```java
package sample;

import javafx.scene.chart.XYChart;

public class Exact {
    public static double function(double x,double c) {
        double y;
        y = (1 / ((c * x * x * x * x * x) - (x / 4))) + (2 / x);
        return y;
    }

    public static XYChart.Series exact(double x0, double y0, double x, double N, double c) {
        double currX = x0;
        double currY = y0;
        XYChart.Series ser = new XYChart.Series();
        double n = Math.abs(x - x0) / N;

        //calculation of exact solution and graph
        while (currX <= x) {
            ser.getData().add(new XYChart.Data<>(currX, currY));

            // exact solution
            currX += n;
            currY = function(currX,c);
        }
        return ser;
    }
}
```

Implementation of method to calculate exact solution and plot the graph.

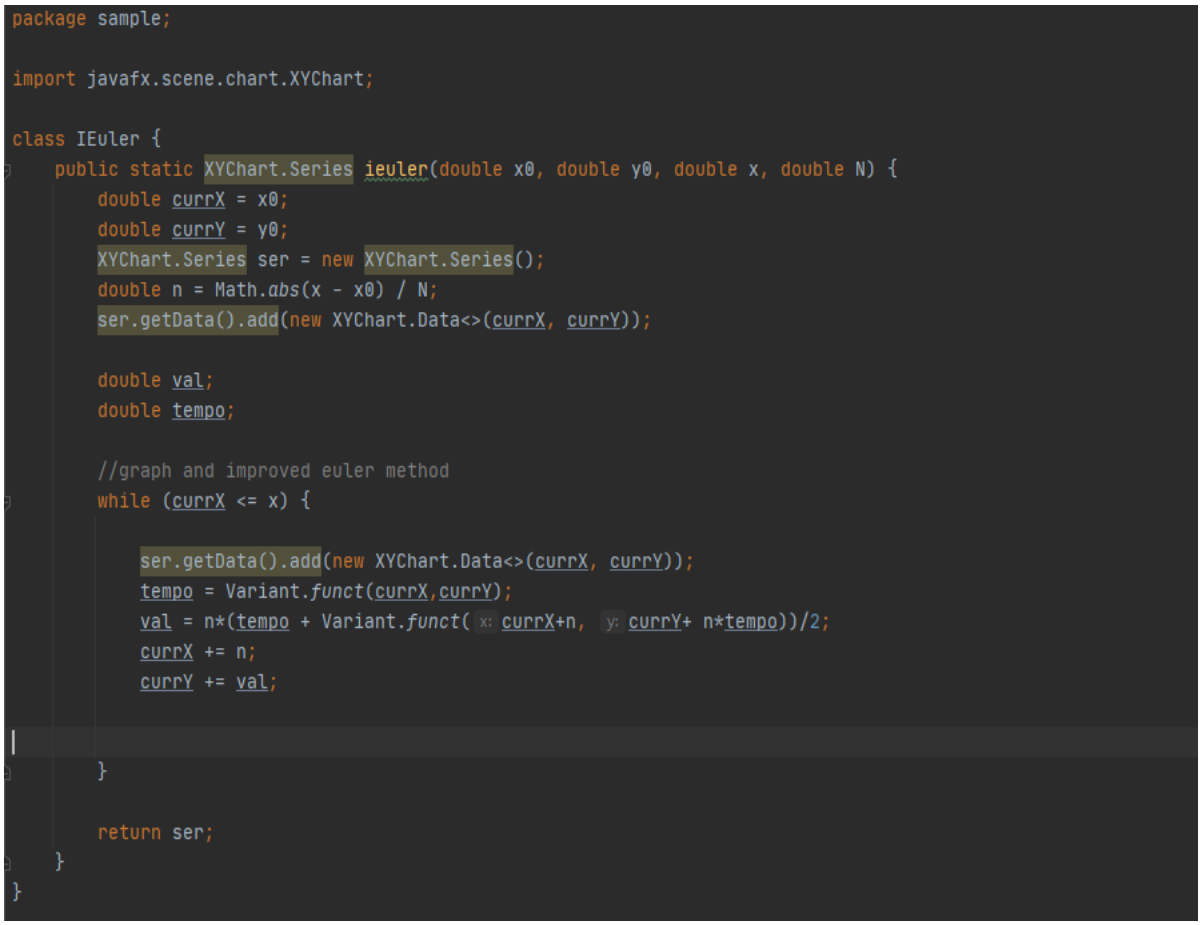
Euler method



```
package sample;

import javafx.scene.chart.XYChart;

public class Euler {
    public static XYChart.Series euler(double x0, double y0, double x, double N) {
        Double currX = x0;
        Double currY = y0;
        XYChart.Series ser = new XYChart.Series();
        Double n = Math.abs(x - x0) / N;
        ser.getData().add(new XYChart.Data<>(currX, currY));

        //euler method and graph
        while (currX <= x) {
            ser.getData().add(new XYChart.Data<>(currX, currY));

            currY += n * Variant.funct(currX, currY);
            currX += n;
        }
        return ser;
    }
}
```

Implementation of Euler method to calculate the numerical value of the differential equation and plot the graph.

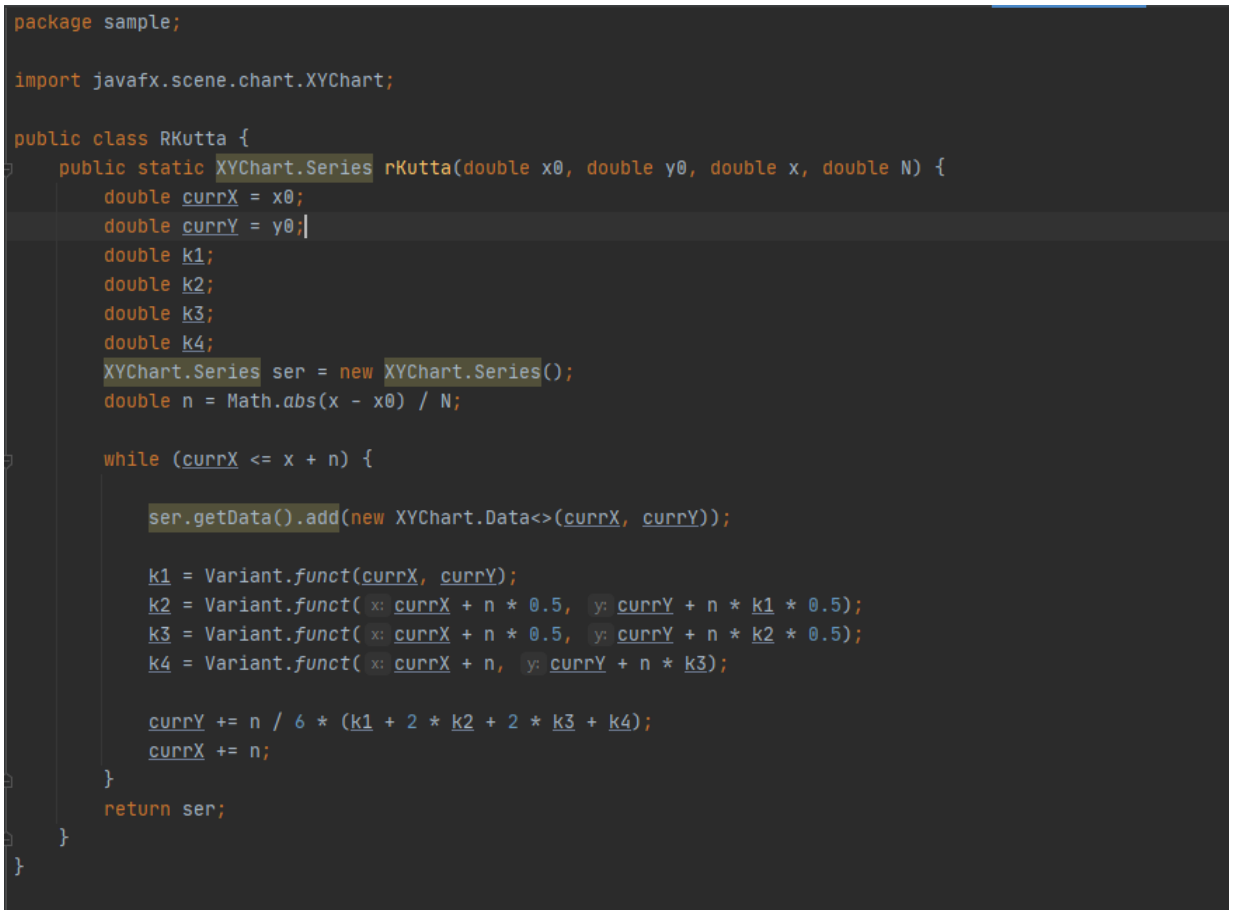
Improved Euler method



```
package sample;

import javafx.scene.chart.XYChart;

class IEuler {
    public static XYChart.Series ieuler(double x0, double y0, double x, double N) {
        double currX = x0;
        double currY = y0;
        XYChart.Series ser = new XYChart.Series();
        double n = Math.abs(x - x0) / N;
        ser.getData().add(new XYChart.Data<>(currX, currY));

        double val;
        double tempo;

        //graph and improved euler method
        while (currX <= x) {

            ser.getData().add(new XYChart.Data<>(currX, currY));
            tempo = Variant.funct(currX,currY);
            val = n*(tempo + Variant.funct( x: currX+n,  y: currY+ n*tempo))/2;
            currX += n;
            currY += val;


        }

        return ser;
    }
}
```

Implementation of Improved Euler method to calculate the numerical value of the differential equation and plot the graph.

Runge-Kutta method

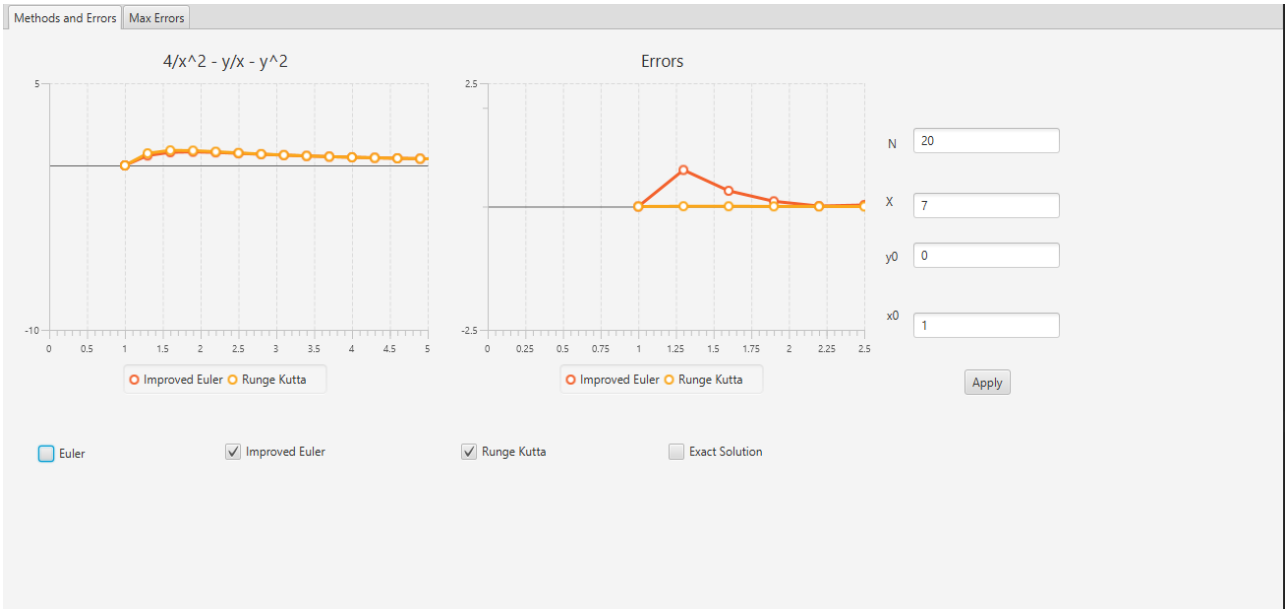


```java
package sample;

import javafx.scene.chart.XYChart;

public class RKutta {
    public static XYChart.Series rKutta(double x0, double y0, double x, double N) {
        double currX = x0;
        double currY = y0;
        double k1;
        double k2;
        double k3;
        double k4;
        XYChart.Series ser = new XYChart.Series();
        double n = Math.abs(x - x0) / N;

        while (currX <= x + n) {

            ser.getData().add(new XYChart.Data<>(currX, currY));

            k1 = Variant.funct(currX, currY);
            k2 = Variant.funct( x: currX + n * 0.5,  y: currY + n * k1 * 0.5);
            k3 = Variant.funct( x: currX + n * 0.5,  y: currY + n * k2 * 0.5);
            k4 = Variant.funct( x: currX + n,  y: currY + n * k3);

            currY += n / 6 * (k1 + 2 * k2 + 2 * k3 + k4);
            currX += n;
        }
        return ser;
    }
}
```

Implementation of Runge-Kutta method to calculate the numerical value of the differential equation and plot the graph.

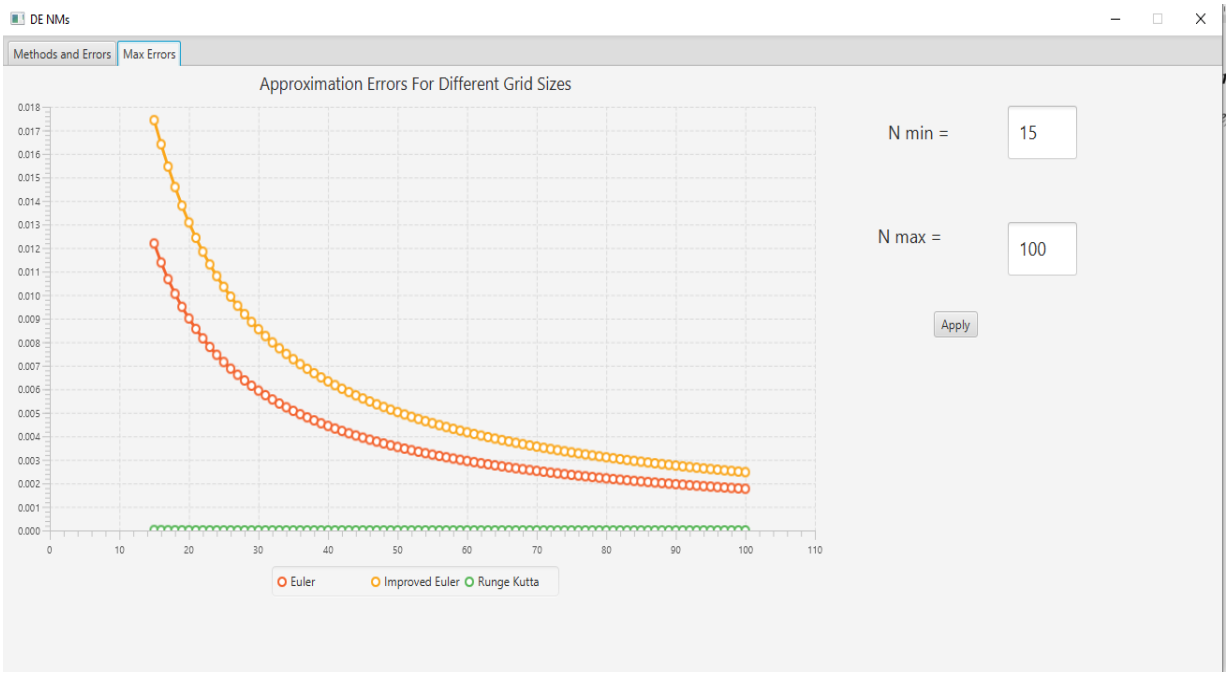
Plots

(For the methods and their respective errors)



Approximation of errors

(For different grid size)



Why Runge-Kutta is better?

(In general)

1. The Euler method numerically gets the first derivative correct. RK4 gets the first four derivatives correct.

2. From the Taylor series, this means that RK4's first error term is from the 5th derivative term, which from the Taylor series has a $(x - x0)^5 = h^5$ *or if you prefer* dx^5*. Therefore, its error is like the fifth power of h and is called 4th order. The Euler method is incorrect at the second derivative, which from the Taylor series has*$(x - x0^2) = h^2$*. This is why it scales as*

h^2*, and is called 1st order.*

3. For more clear understanding, think about scaling our solution. You have a solution at h, and now you re-solve at h/2. How much did you change our error? For the Euler method your error is $\left(\frac{h}{2}\right)^2 = \frac{h^2}{4}$*, so you have a quarter of the error from before. For the RK4 method you have* $\left(\frac{h}{2}\right)^5 = \frac{h^5}{32}$ *and so you have* $\frac{1}{32}$ *the error from before!*

4. Thus, we can see that as h gets smaller the higher order method gets better and better.

Conclusion

Thus, three different methods are implemented, compared, and their errors are plotted. This report is based on the requirements given on Moodle.