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**: $y' = 4/x^2 - y/x - y^2$

We will seek for a particular solution in the form:

$$y = \frac{c}{x} \qquad y' = -\frac{c}{x^2}$$

Substituting this into the equation, we obtain:

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

$$or, \quad c^2 = 4$$

$$\therefore \quad c = \pm 2$$

We can take any value of c*. For example, let* $c = 2$*. Now, when the particular solution is known, we make the replacement:*

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

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

Now substitute this into the original Riccati equation:

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

$$or, \quad -\frac{z'}{z^2} - \frac{2}{x^2} = \frac{4}{x^2} - \frac{1}{2x} - \frac{2}{x^2} - \frac{1}{z^2} - \frac{4}{zx} - \frac{4}{x^2}$$

So cancelling out the terms,

$$or, \quad -\frac{z'}{z^2} = \frac{5}{zx} - \frac{1}{z^2}$$

$$or, \quad z' = \frac{5z}{x} + 1$$

Integrating for x,

$$\therefore \quad z = c_1 x^5 - \frac{x}{4}$$

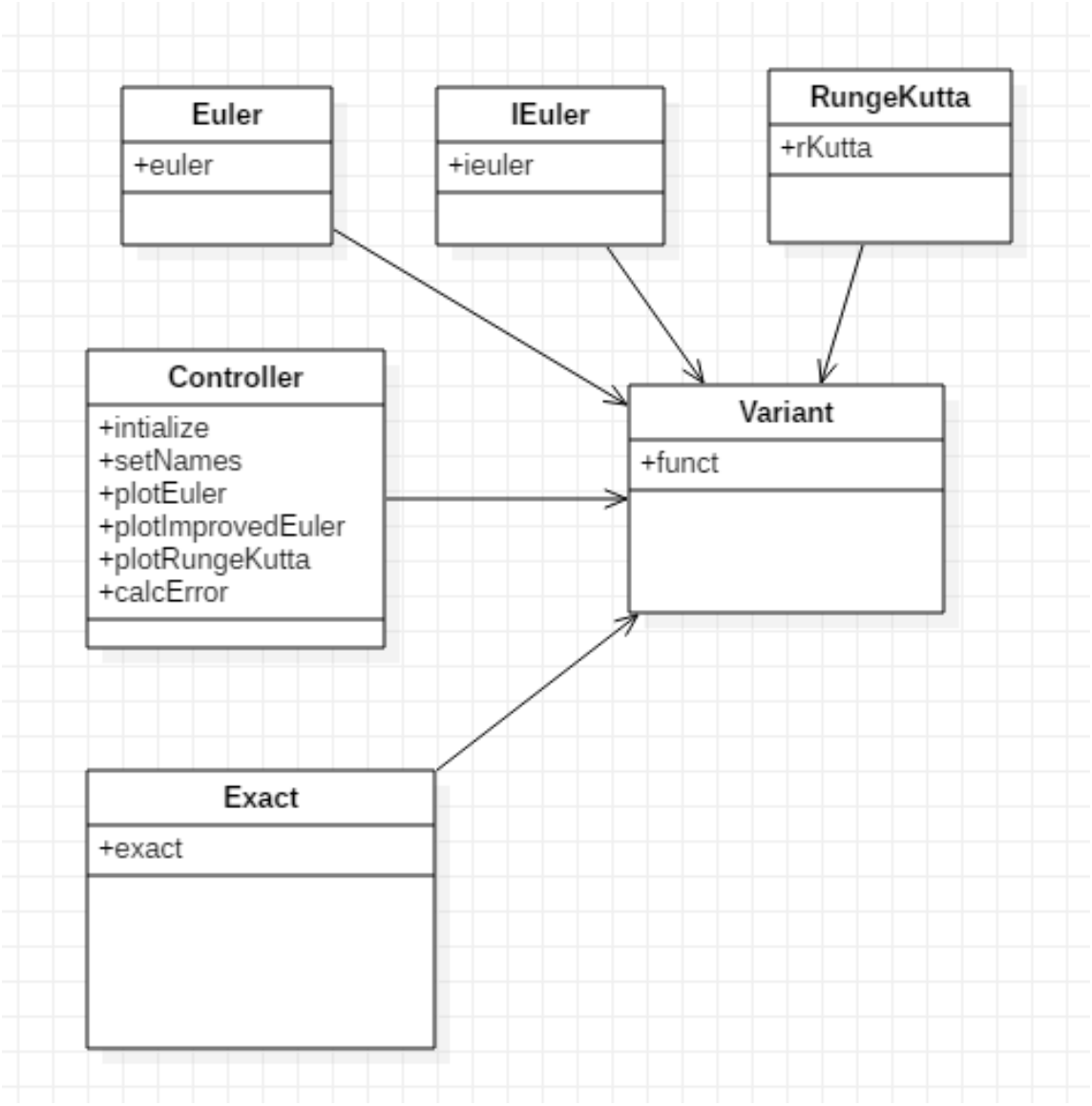
$$y = \frac{1}{c_1 x^5 - \frac{x}{4}} + \frac{2}{x} \Bigg\} \rightarrow (1,3)$$

Simpliying with $x = 1$ *and* $y = 3$ *we will get* $c_1 = \frac{5}{4}$

Putting back the value of c_1 *we will get,*

$$\boldsymbol{y = \frac{4 + 2(5x^4 - 1)}{x(5x^4 - 1)}} \ (\boldsymbol{analytical\ solution})$$

UML DIAGRAM



Exact Solution

```
public class Exact {
    public static double function(double x) {
        double y;
        y = (4/(5*x*x*x*x*x - x)) + 2/x;
        return y;
    }

    public static XYChart.Series exact(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;

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

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

Euler Method

```
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) {

            currY += n * Variant.funct(currX, currY);
            currX += n;

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

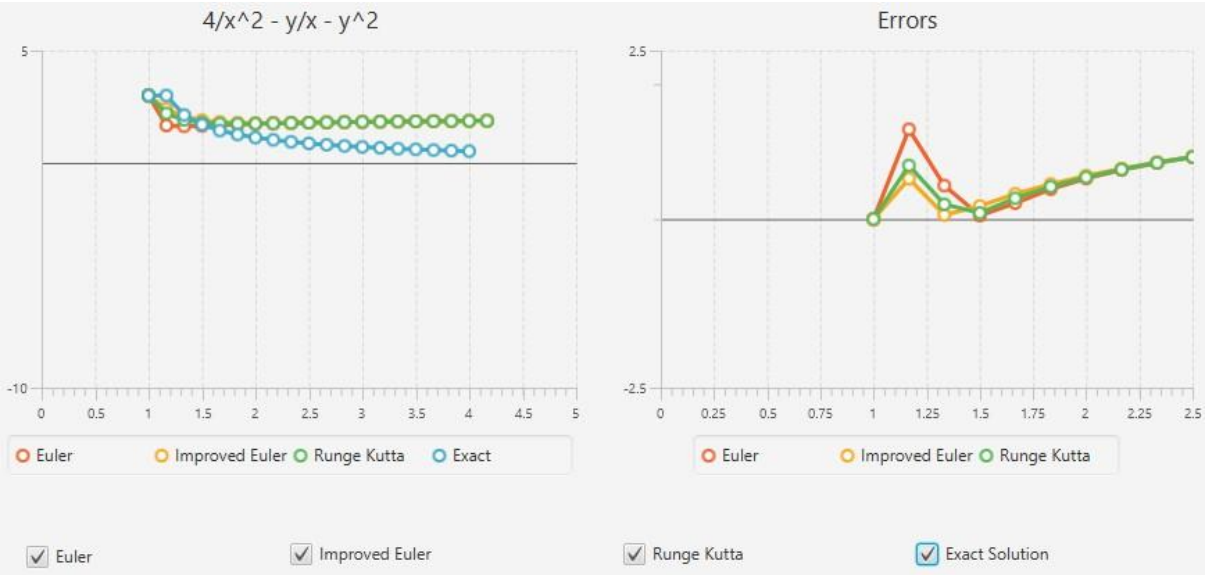
Improved Euler

```
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) {

            tempo = Variant.funct(currX,currY);
            val = n*Variant.funct( x: currX+n/2,  y: currY+ n*tempo/2);
            currX += n;
            currY += val;

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

        return ser;
    }
}
```

Runge Kutta

```
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;
    }
}
```

Plot and Error



Why Runge Kutta is better in general?

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

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-x_0)^5. = h^5 or if you prefer dx^5. So 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 a (x-x_0)^2 = h^2. This is why it scales as h^2 and is called 1st order.

To understand what this really means, 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 (h/2)^2 = h^2/4, so you have a quarter of the error from before. For the RK4 method you have (h/2)^5 = h^2/32 and so you have 1/32 the error from before! Thus we can see that as h gets smaller the higher order method gets better and better.

Git hub Link: https://github.com/donrast41/DEs

Conclusion: *Thus, three different methods are implemented and they are compared and their errors are plotted. This report is based on the requirements given on moodle.*