DIFFERENTIAL EQUATIONS COMPUTATIONAL PRACTICUM

Goals:

1) Implement Euler’s, improved Euler’s and Runge-Kutta methods.

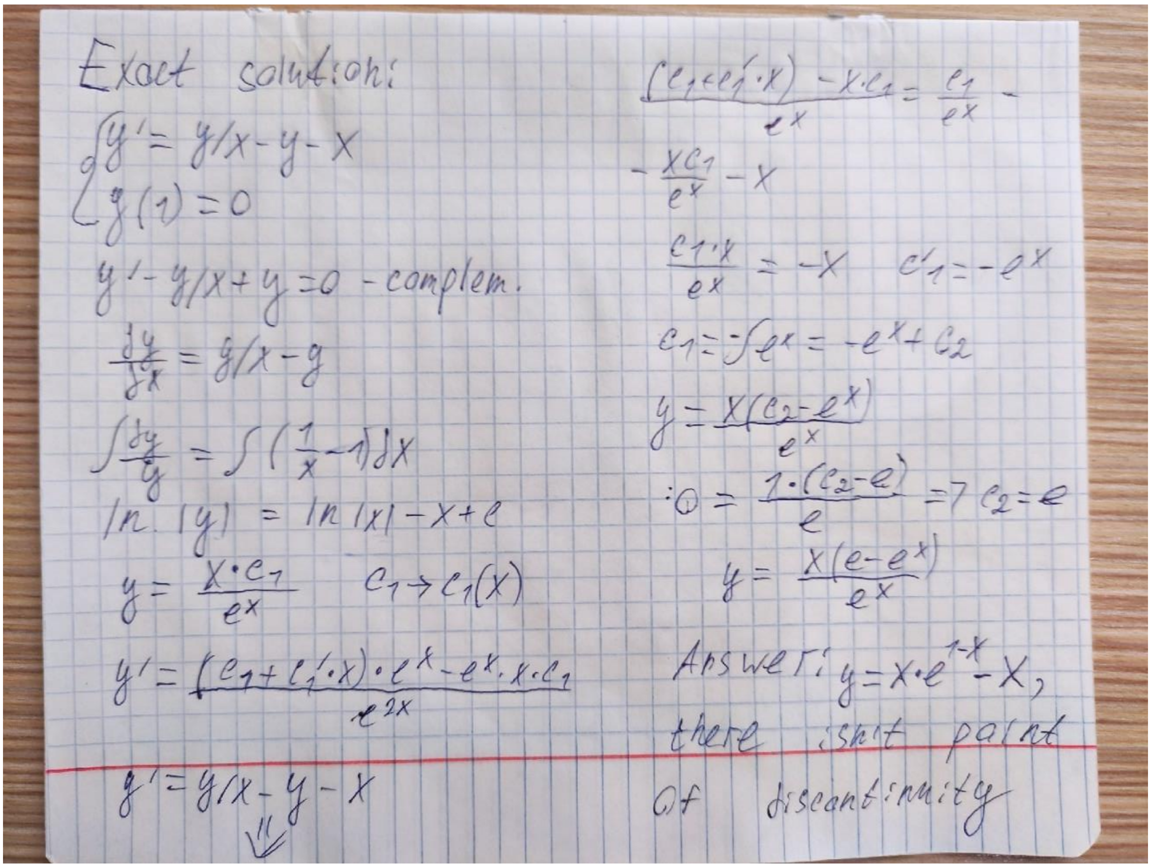
2) Implement plots of the graphs of exact and numerical solutions, graphs of local errors for each method, graphs of total approximation error depending on the number of grid cells.

3) Check the properties of each method and find differences from other methods.

Work done by:

Zakharov Mark Group-04

Part 1



Exact solution:

$$\begin{cases} y' = y/x - y - x \\ y(1) = 0 \end{cases}$$

$y' - y/x + y = 0$ - complem.

$$\frac{dy}{dx} = y/x - y$$

$$\int \frac{dy}{y} = \int \left(\frac{1}{x} - 1\right) dx$$

$$\ln|y| = \ln|x| - x + c$$

$$y = \frac{x \cdot c_1}{e^x} \qquad c_1 \to c_1(x)$$

$$y' = \frac{(c_1 + c_1' \cdot x) \cdot e^x - e^x \cdot x \cdot c_1}{e^{2x}}$$

$$y' = y/x - y - x$$

$$\Downarrow$$

$$\frac{(c_1 + c_1' \cdot x)}{e^x} - \frac{x \cdot c_1}{e^x} = \frac{c_1}{e^x} - \frac{x c_1}{e^x} - x$$

$$\frac{c_1' \cdot x}{e^x} = -x \qquad c_1' = -e^x$$

$$c_1 = -\int e^x = -e^x + c_2$$

$$y = \frac{x(c_2 - e^x)}{e^x}$$

$$0 = \frac{1 \cdot (c_2 - e)}{e} \Rightarrow c_2 = e$$

$$y = \frac{x(e - e^x)}{e^x}$$

Answer: $y = x \cdot e^{1-x} - x$, there isn't point of discontinuity

After solving the equation, we derive 2 functions that are most important for us:

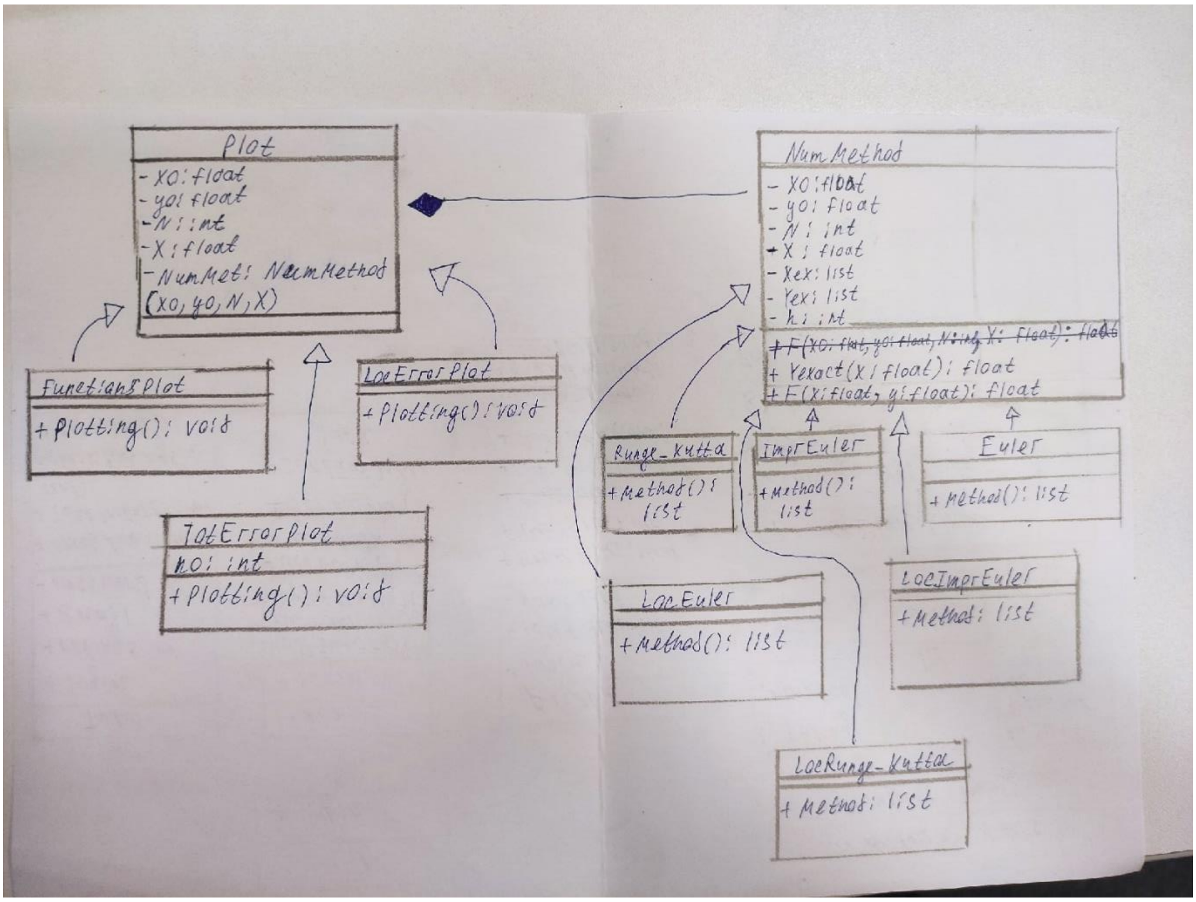
F(x, y) – function of y’

Yexact(x) – exact solution of our equation

```
def F(self, x, y):
    return y/x - y - x

def Yexact(self, x):
    return x*(e**(1-x) - 1)
```

Part 2

UML-diagram for classes in my project:



In my system Euler, ImprEuler, Runge_Kutta, LocEuler, LocImprEuler, LocRunge_Kutta classes inherited from NumMethod class. FunctionsPlot, LocErrorPlot, TotErrorPlot inherited from Plot class. Plot class uses NumMethod class for all of its children classes.

All our methods are presented below:

```
class Euler(NumMethod):

    def Method(self):
        for i in range(1, self.N+1):
            self.Xex.append(self.x0 + i*self.h)
            self.Yeu.append(self.Yeu[i-1] + self.h*self.F(self.Xex[i-1], self.Yeu[i-1]))
        return self.Yeu
```

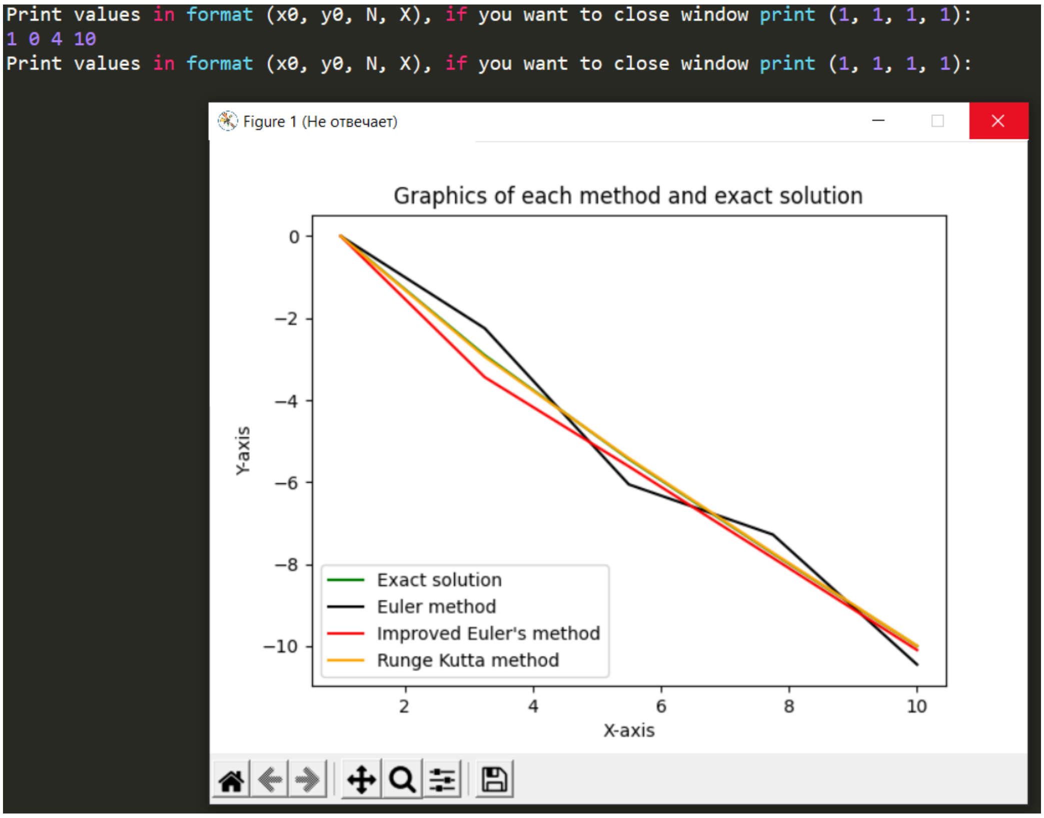
```python
class ImprEuler(NumMethod):

    def Method(self):
        for i in range(1, self.N+1):
            self.Xex.append(self.x0 + i*self.h)
            self.Yeu.append(self.Yeu[i-1] + self.h*self.F(self.Xex[i-1] + self.h/2, self.Yeu[i-1] + self.h/2*self.F(self.Xex[i-1], self.Yeu[i-1])))
        return self.Yeu
```

```python
class Runge_Kutta(NumMethod):

    def Method(self):
        for i in range(1, self.N+1):
            self.Xex.append(self.x0 + i*self.h)
            k1 = self.F(self.Xex[i-1], self.Yeu[i-1])
            k2 = self.F(self.Xex[i-1] + self.h/2, self.Yeu[i-1] + self.h*k1/2)
            k3 = self.F(self.Xex[i-1] + self.h/2, self.Yeu[i-1] + self.h*k2/2)
            k4 = self.F(self.Xex[i-1] + self.h, self.Yeu[i-1] + self.h*k3)
            self.Yeu.append(self.Yeu[i-1] + (self.h/6)*(k1+2*k2+2*k3+k4))
        return self.Yeu
```

Our class with GUI, that allows the user to change x0, y0, X, N and plot the graphs of exact and numerical solutions:

```python
class GeneralPlot(Plot):

    def Plotting(self):

        plt.ion()

        while (self.y0 != 1 or self.x0 != 1 or self.X != 1 or self.N != 1):

            print("Print values in format (x0, y0, N, X), if you want to close window print (1, 1, 1, 1): ")
            self.x0, self.y0, self.N, self.X = map(float, input().split())
            self.N = int(self.N)
            self.h = (self.X-self.x0)/self.N
            plt.clf()

            x = []
            y = []
            for i in range(int(self.N)+1):
                x.append(self.x0 + i*self.h)
                y.append(self.NumMet.Yexact(x[i]))

            eul = Euler(self.x0, self.y0, self.N, self.X).Method()
            i_eul = ImprEuler(self.x0, self.y0, self.N, self.X).Method()
            run_k = Runge_Kutta(self.x0, self.y0, self.N, self.X).Method()

            plt.title("Graphics of each method and exact solution")
            plt.xlabel("X-axis")
            plt.ylabel("Y-axis")

            plt.plot(x, y, label = u'Exact solution', color = 'green')
            plt.plot(x, eul, label = u'Euler method', color = 'black')
            plt.plot(x, i_eul, label = u"Improved Euler's method", color = 'red')
            plt.plot(x, run_k, label = u'Runge Kutta method', color = 'orange')
            plt.legend(loc= 3)
            plt.draw()
            plt.pause(0.3)

        plt.ioff()
```

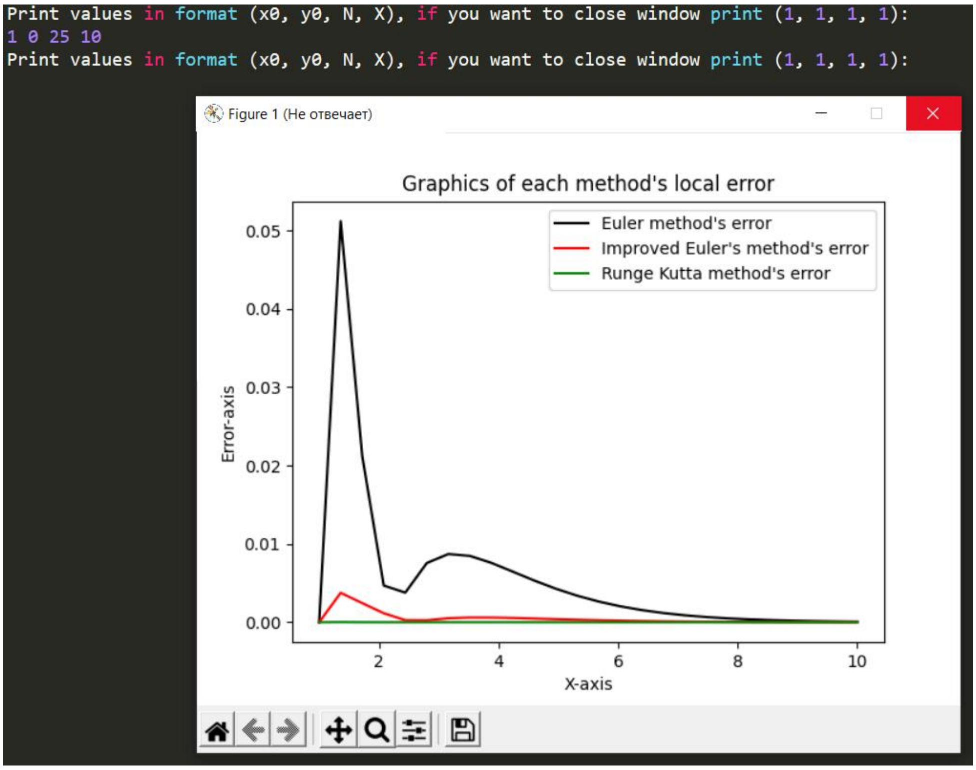
Example of work of FunctionsPlot class:



Our class with GUI, that allows the user to change x0, y0, X, N and plot graph of local errors for each method:

```python
class LocErrorPlot(Plot):

    def Plotting(self):
        plt.ion()
        while (self.y0 != 1 or self.x0 != 1 or self.X != 1 or self.N != 1):

            print("Print values in format (x0, y0, N, X), if you want to close window print (1, 1, 1, 1): ")
            self.x0, self.y0, self.N, self.X = map(float, input().split())
            self.N = int(self.N)
            self.h = (self.X-self.x0)/self.N
            plt.clf()

            eul = LocEuler(self.x0, self.y0, self.N, self.X).Method()
            i_eul = LocImprEuler(self.x0, self.y0, self.N, self.X).Method()
            run_k = LocRunge_Kutta(self.x0, self.y0, self.N, self.X).Method()
            x, y = [], []
            for i in range(int(self.N)+1):
                x.append(self.x0 + i*self.h)
                y.append(self.NumMet.Yexact(x[i]))
                eul[i] -= y[i]
                i_eul[i] -= y[i]
                run_k[i] -= y[i]
                if eul[i] < 0:
                    eul[i] *= -1
                if i_eul[i] < 0:
                    i_eul[i] *= -1
                if run_k[i] < 0:
                    run_k[i] *= -1

            plt.title("Graphics of each method's local error")
            plt.xlabel("X-axis")
            plt.ylabel("Error-axis")
            plt.plot(x, eul, label = u"Euler method's error", color = 'black')
            plt.plot(x, i_eul, label = u"Improved Euler's method's error", color = 'red')
            plt.plot(x, run_k, label = u"Runge Kutta method's error", color = 'green')
            plt.legend(loc= 1)
            plt.draw()
            plt.pause(0.3)
        plt.ioff()
```

Example of work of LocErrorPlot class:

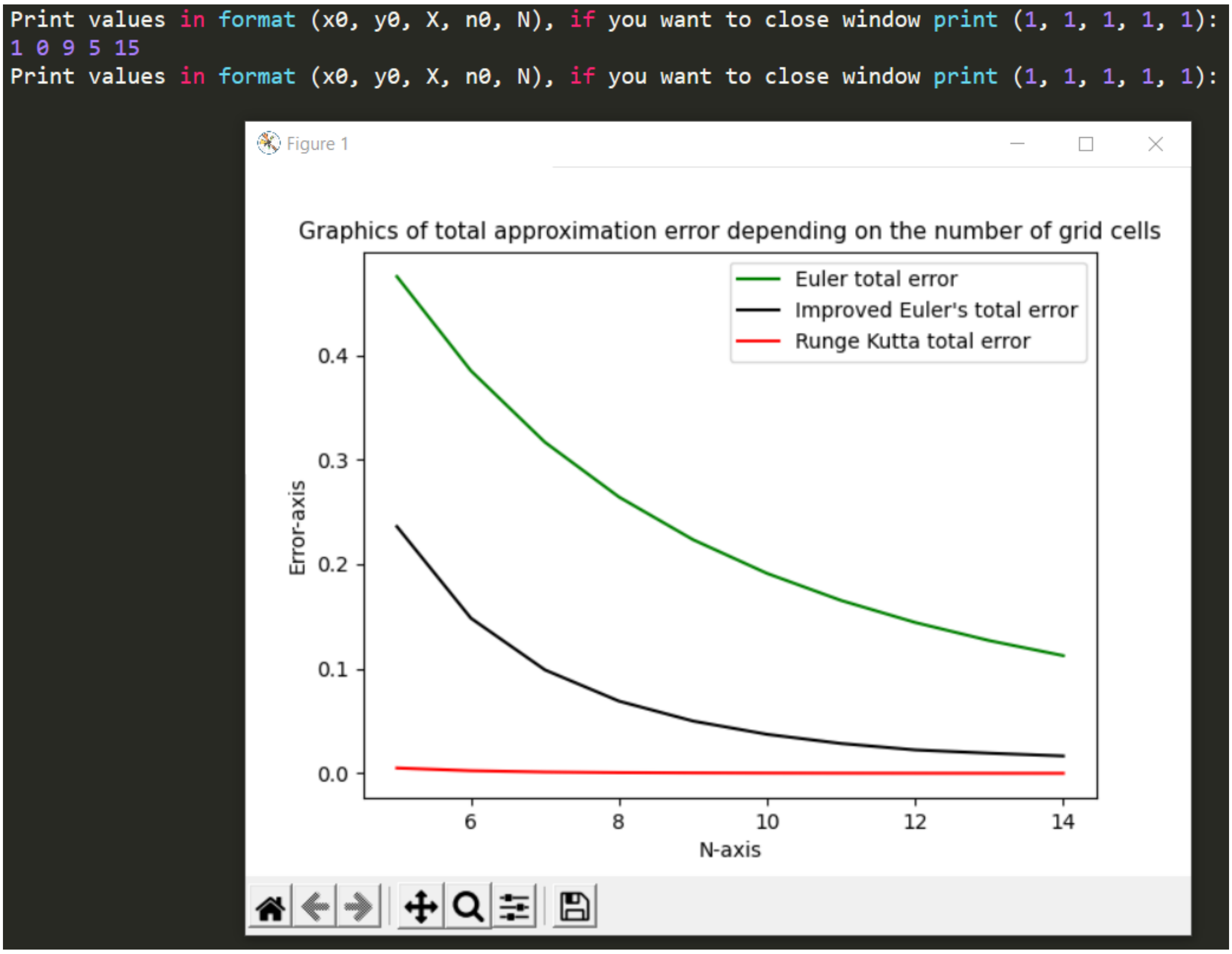


Part 3

Our class with GUI, that allow user input starting and finishing values of the number of grid cells and provide the graph of total errors for each method in a given range:

```python
class TotErrorPlot(Plot):

    n0 = 0
    def Plotting(self):

        plt.ion()
        while (self.y0 != 1 or self.x0 != 1 or self.X != 1 or n0 != 1 or self.N != 1):

            print("Print values in format (x0, y0, X, n0, N), if you want to close window print (1, 1, 1, 1, 1): ")
            self.x0, self.y0, self.X, n0, self.N = map(float, input().split())
            n0 = int(n0)
            self.N = int(self.N)
            eul_er = []
            i_eul_er = []
            run_k_er = []
            Xn = []
            plt.clf()

            for i in range(n0, self.N):
                self.h = (self.X-self.x0)/i
                eul = Euler(self.x0, self.y0, i, self.X).Method()
                i_eul = ImprEuler(self.x0, self.y0, i, self.X).Method()
                run_k = Runge_Kutta(self.x0, self.y0, i, self.X).Method()
                x, y = [], []
                for j in range(i+1):
                    x.append(self.x0 + j*self.h)
                    y.append(self.NumMet.Yexact(x[j]))
                    eul[j] -= y[j]
                    i_eul[j] -= y[j]
                    run_k[j] -= y[j]
                    if eul[j] < 0:
                        eul[j] *= -1
                    if i_eul[j] < 0:
                        i_eul[j] *= -1
                    if run_k[j] < 0:
                        run_k[j] *= -1
                eul_er.append(max(eul))
                i_eul_er.append(max(i_eul))
                run_k_er.append(max(run_k))
                Xn.append(i)

            plt.title("Graphics of total approximation error depending on the number of grid cells", size = 11)
            plt.xlabel("N-axis")
            plt.ylabel("Error-axis")
            plt.plot(Xn, eul_er, label = u'Euler total error', color='green')
            plt.plot(Xn, i_eul_er, label = u"Improved Euler's total error", color='black')
            plt.plot(Xn, run_k_er, label = u'Runge Kutta total error', color='red')
            plt.legend(loc= 1)
            plt.draw()
            plt.pause(0.3)
        plt.ioff()
```

Example of work of TotErrorPlot class:



Conclusions:

1) I implement all methods and plots.

2) In FunctionsPlot I found out that Euler’s method most inaccurate to the exact solution, Runge-Kutta method is the most accurate to the exact solution, improved Euler's method is slightly accurate to the exact solution.

3) In LocErrorPlot I found out that all methods have the largest local error for the first few steps.

4) In TotErrorPlot I found out that the largest global error of each method decreases with an increase in the number of steps.