Part I

Provided information :

Given equation : $$\frac{dy}{dx} = \frac{y}{x} - y - x$$

Initial conditions : $y(1) = 0$

Initial values : $x_0 = 1,\ y_0 = 0$

Domain of approximation : $x \in (x_0, 10)$

Solution :

1) This is first order linear ODE of the form :

$$y'(x) + p(x)\,y = q(x)$$

2) After rewriting in linear ODE form we get :

$$y' + \left(-\frac{1}{x} + 1\right) y = -x$$

3) After using y = uv substitution and solving for u and v, we finally can write the general solution :

$$y(x) = x(\, c_0\, e^{-x} \; - \; 1\,)$$

4) After inputing initial value of y and x, we find the value of constant :

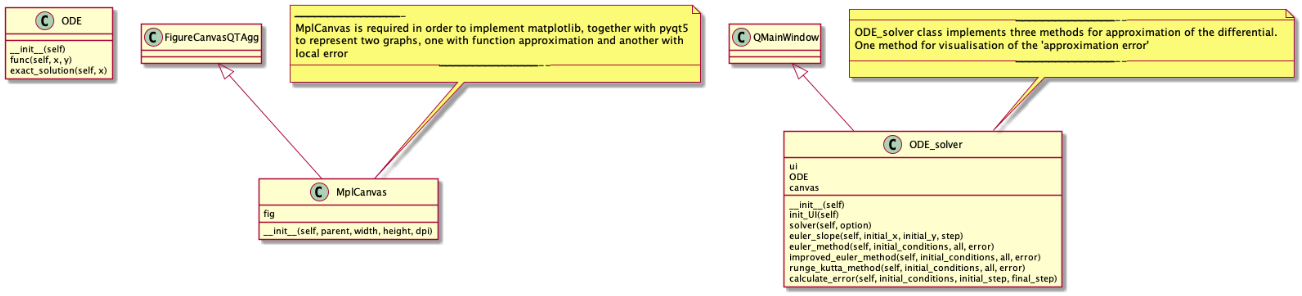
$$c_0 = e$$

5) After substituting constant we get :

$$y(x) = x\,(\, e^{-x+1} \; - \; 1\,)$$

Part II

UML Diagram :



Classes structure in the code :

```
# Helper class with exact solution of differential equation
# and a differential function.
class ODE():

# additional class for graphics
class MplCanvas(FigureCanvasQTAgg)

 # class containing all the methods of approximation
 # and methods to calculate error
class ODE_solver(QtWidgets.QMainWindow)

# class responsible for graphics
class Ui_MainWindow(object)
```

Part III

Euler’s method (most important code part) :

```
        # looping through in order to get consequetial y values => #
        while initial_x < approximation_point:

                if self.ODE.func(initial_x, initial_y) == None:
                    # check function for point of discontinuity => #
                    print("Point of discontinuit at: "+str(initial_x)+" and "+str(initial_y))
                    initial_x = initial_x + step
                    continue
                else:
                    initial_y = self.euler_slope(initial_x,initial_y,step)
                    initial_x = initial_x + step

                    x_axis.append(initial_x)
                    y_axis.append(initial_y)
                    exact_y.append(self.ODE.exact_solution(initial_x))
                    error_y.append(abs(initial_y - self.ODE.exact_solution(initial_x)))
```

Improved Euler’s method (most important code part)

```
# looping through in order to get consequetial y values => #
while initial_x < approximation_point:
    if self.ODE.func(initial_x, initial_y) == None:
            # check function for point of discontinuity => #
            print("Point of discontinuit at: "+initial_x+" and "+initial_y)
    else:
        initial_y = initial_y + (step * (1/2 * (self.ODE.func(initial_x,initial_y) + self.ODE.func(initial_x+step,self.euler_slope(initial_x,initial_y
          ,step)))))
        initial_x = initial_x + step
        x_axis.append(initial_x)
        y_axis.append(initial_y)
        exact_y.append(self.ODE.exact_solution(initial_x))
        error_y.append(abs(initial_y - self.ODE.exact_solution(initial_x)))
```

Runge-Kutta method (most important code part) :

```
# looping through in order to get consequetial y values => #
while initial_x < approximation_point:
    k_1 = self.ODE.func(initial_x, initial_y)
    k_2 = self.ODE.func(initial_x + (step/2), initial_y + ((k_1/2) * step))
    k_3 = self.ODE.func(initial_x + (step/2), initial_y + ((k_2/2) * step))
    k_4 = self.ODE.func(initial_x + step, initial_y + (step*k_3))

    initial_x = initial_x + step

    x_axis.append(initial_x)

    initial_y = initial_y + (1.0/6.0) * step * (k_1 +  2*k_2 + 2*k_3 + k_4)
    y_axis.append(initial_y)
    exact_y.append(self.ODE.exact_solution(initial_x))
    error_y.append(abs(initial_y - self.ODE.exact_solution(initial_x)))
```

Euler Slope (explanation) :

```
def euler_slope(self, initial_x, initial_y, step):

    # this function provides the euler's slope used in two methods => #
    return (initial_y + (step * self.ODE.func(initial_x, initial_y)))
```

Global Error Calculation :

```python
def calculate_error(self, initial_conditions, initial_step, final_step):
    # "unpacking" provided initial conditions => #
    initial_x, initial_y, step, approximation_point = initial_conditions

    euler_y=[]
    improved_euler_y=[]
    rk_y=[]
    axis_x=[]

    while initial_step < (final_step + 1):

        step = (float(approximation_point) - float(initial_x)) / float(initial_step)
        euler_y.append(self.euler_method((initial_x, initial_y, step, approximation_point), False, True))
        improved_euler_y.append(self.improved_euler_method((initial_x, initial_y, step, approximation_point), False, True))
        rk_y.append(self.runge_kutta_method((initial_x, initial_y, step, approximation_point), False, True))
        axis_x.append(initial_step)

        initial_step = initial_step + 1

```