Computational practicum

Dluzhinskaya Anna (Variant 5)
(a.dluzhinskaya@innopolis.university)

1) Exact solution

Given:

- $y' = \frac{y}{x} + x\cos(x) \qquad y_0 = 1 \quad x_0 = \pi$

Solution:

Remark: $x \neq 0$

$$y' - \frac{1}{x}y = x\cos(x)$$

Complementary solution :

$$y' - \frac{1}{x}y = 0$$

$$y' = \frac{1}{x}y$$

$$\int \frac{1}{y}\,dy = \int \frac{1}{x}\,dx$$

$$\ln|y| = \ln|x| + C$$

$$y = Cx \quad C \rightarrow C(x) \longrightarrow \quad y' = C'x + C$$

Substitute into initial equation:

$$C'x + C - \frac{1}{x}Cx = x\cos(x)$$

$$C'x + \cancel{C} - \cancel{C} = x\cos(x)$$

$$C'\cancel{x} = \cancel{x}\cos(x)$$

$$C' = \cos(x) \longrightarrow C = \sin(x) + C_0$$

$$\boxed{y = x\sin(x) + C_0 x}$$

Solve initial value problem:

$$\begin{cases} y = x\sin(x) + C_0 x \\ y_0 = 1 \\ x_0 = \pi \end{cases}$$

$$1 = \pi\sin(\pi) + C_0\pi \longrightarrow C_0 = \frac{y - x\sin(x)}{x}$$

$$C_0 = \frac{1 - \pi\sin(\pi)}{\pi}$$

$$C_0 = \frac{1}{\pi}$$

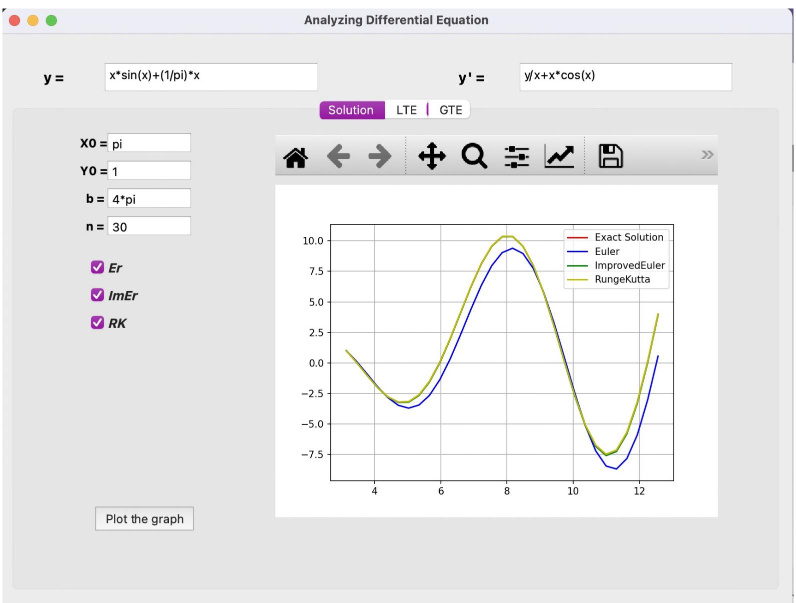
Exact solution with given initial values: $y = x\sin(x) + \frac{1}{\pi}x$

Exact solution with different initial values: $y = x\sin(x) + C_0 x$ where

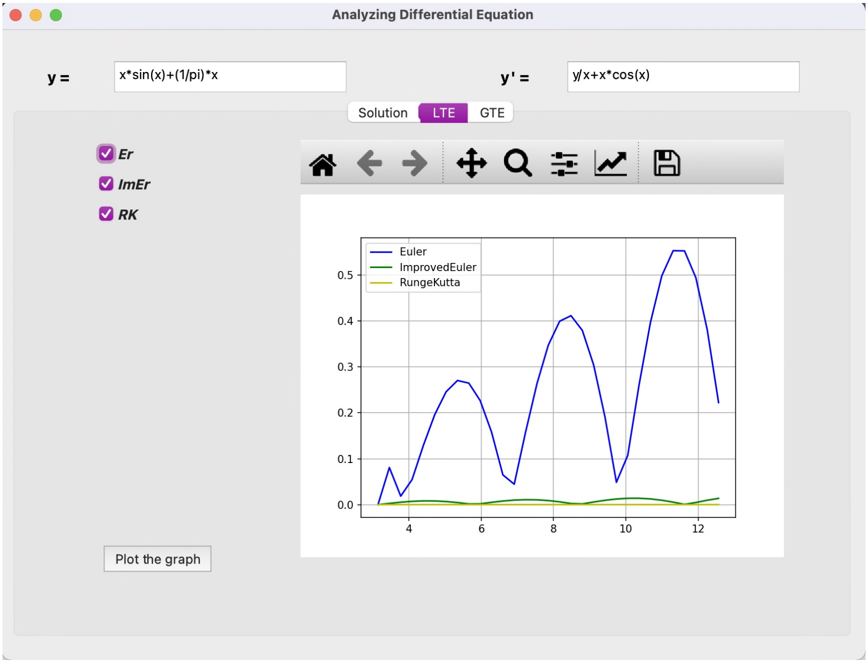
$$C_0 = \frac{y_0 - x_0\sin(x)}{x_0}$$

2) Screenshots of application:

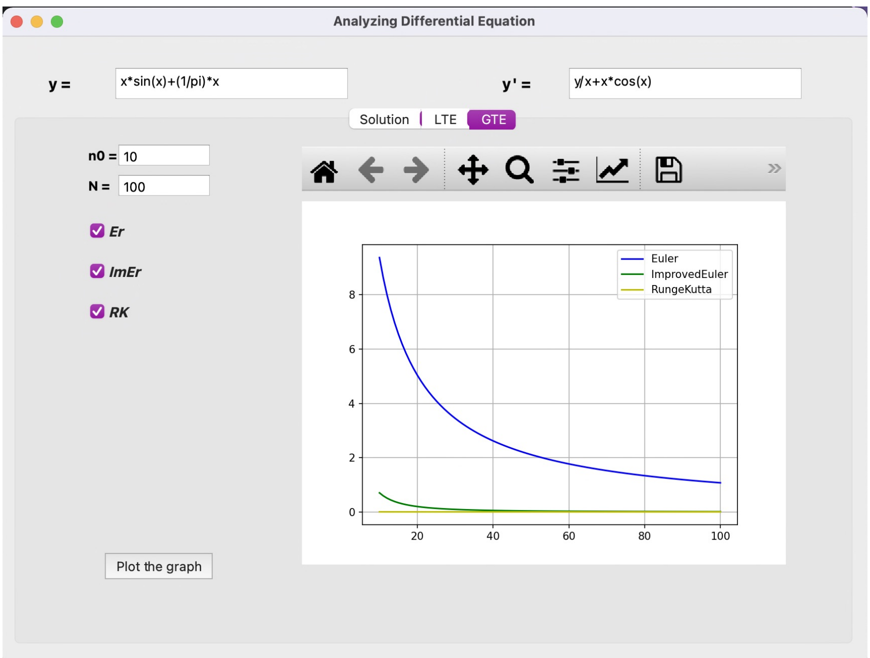
Solution tab:



LTE tab:



GTE tab:



3) Code: (https://github.com/AnnaDluzhinskaya/DifferentialEquations)

In my project I have ExactSolution.py, where C is calculating depending on initial values. This constant is substituted into the equation and due to this in the upper left corner of the application, you will be able to see the update expression. Also in this part of program if initial value of x is equal to zero then the value is replaced by one that is closed to zero.

```
class ExactSolution:
    def __init__(self, eq, y0, x0):
        self.exact_eq = eq
        if x0 != 0:
            self.x = x0
            if x0 == float(pi) and y0 == 1.0:
                self.x = "pi"
                self.exact_eq = eq.replace("c", '(1/pi)', eq.count("c"))
            elif x0 != 0:
                c = "(y-x*sin(x))/x"
                c = c.replace("x", str(x0), c.count("x"))
                c = c.replace("y", str(y0), c.count("y"))
                c = 1 * ne.evaluate(c)
                self.x = round(self.x, 4)
                self.exact_eq = eq.replace("c", "("+str(round(c, 4))+")", eq.count("c"))
        elif x0 == 0:
            x0 = 0.0000001
            self.x = x0
            c = "(y-x*sin(x))/x"
            c = c.replace("x", str(x0), c.count("x"))
            c = c.replace("y", str(y0), c.count("y"))
            c = 1 * ne.evaluate(c)
            self.x = round(self.x, 7)
            self.exact_eq = eq.replace("c", str(round(c, 4)), eq.count("c"))
```

Also in my project has class NumericalMethods, which contains all the necessary fields for solving the differential education. For example, methods substituteEq() and substituteDiff() which calculate the value of expression. And also methods calculateLTE() and calculateGTE() which are the same for every type of numerical methods.

```
class NumericalMethods:
    def __init__(self, eS, eq, y_0, a, b, n):
        self.y_exact = []
        self.y_approx1 = []
        self.y_approx2 = []
        self.LTE = []
        self.GTE = []
        self.diff_equation = eq
        self.exact_eq = eS.exact_eq
        self.step = (b - a) / n
        self.numberOfSteps = n
        self.initialValueY = y_0

        self.x = []
        temp = a
        i = 0
        while temp < b:
            temp = a + self.step * i
            i = i + 1
            if temp == 0:
                temp = 0.0000001
            self.x.append(temp)

        for i in range(0, self.numberOfSteps + 1):
            if i == 0:
                self.y_exact.append(self.initialValueY)
            else:
                self.y_exact.append(self.substituteEq(self.x[i]))
```

```
def substituteEq(self, x):
    temp_eq = self.exact_eq
    temp_eq = temp_eq.replace('x', str(x), temp_eq.count('x'))
    return 1 * ne.evaluate(temp_eq)

def substituteDiff(self, x, y):
    temp_diff = self.diff_equation
    temp_diff = temp_diff.replace('x', str(x), temp_diff.count('x'))
    temp_diff = temp_diff.replace('y', str(y), temp_diff.count('y'))
    return 1 * ne.evaluate(temp_diff)
```

```
def calculateLTE(self):
    for i in range(0, self.numberOfSteps + 1):
        if i == 0:
            self.LTE.append(0)
        else:
            self.LTE.append(abs(self.y_exact[i] - self.y_approx2[i]))
    return self.LTE

def calculateGTE(self):
    for i in range(0, self.numberOfSteps + 1):
        if i == 0:
            self.GTE.append(0)
        else:
            self.GTE.append(abs(self.y_exact[i] - self.y_approx1[i]))
    return self.GTE
```

There are classes Euler, ImprovedEuler and RungeKutta in the protect that extend the class NumericalMethods and contain only one method solveTask() which calculate approximate values depends on method

```python
class Euler(NumericalMethods):
    def solveTask(self):
        for i in range(0, self.numberOfSteps + 1):
            if i == 0:
                self.y_approx1.append(self.initialValueY)
                self.y_approx2.append(self.initialValueY)
            else:
                self.y_approx1.append(self.y_approx1[i - 1] + self.step * self.substituteDiff(self.x[i - 1], self.y_approx1[i - 1]))
                self.y_approx2.append(self.y_exact[i - 1] + self.step * self.substituteDiff(self.x[i - 1], self.y_exact[i - 1]))

```

```python
class ImprovedEuler(NumericalMethods):
    def solveTask(self):
        for i in range(0, self.numberOfSteps + 1):
            if i == 0:
                self.y_approx1.append(self.initialValueY)
                self.y_approx2.append(self.initialValueY)
            else:
                self.y_approx1.append(self.y_approx1[i - 1] + self.step * self.substituteDiff((self.x[i - 1] + self.step / 2),
                                                (self.y_approx1[i - 1] + self.step / 2 * self.substituteDiff(self.x[i - 1], self.y_approx1[i - 1]))))
                self.y_approx2.append(self.y_exact[i - 1] + self.step * self.substituteDiff((self.x[i - 1] + self.step / 2),
                                                (self.y_exact[i - 1] + self.step / 2 * self.substituteDiff(self.x[i-1],self.y_exact[i - 1]))))
```

```python
class RungeKutta(NumericalMethods):
    def solveTask(self):
        for i in range(0, self.numberOfSteps + 1):
            if i == 0:
                self.y_approx1.append(self.initialValueY)
                self.y_approx2.append(self.initialValueY)
            else:
                k1 = self.substituteDiff(self.x[i-1], self.y_approx1[i - 1])
                k2 = self.substituteDiff((self.x[i-1] + self.step / 2), (self.y_approx1[i - 1] + self.step * k1 / 2))
                k3 = self.substituteDiff((self.x[i-1] + self.step / 2), (self.y_approx1[i - 1] + self.step * k2 / 2))
                k4 = self.substituteDiff((self.x[i-1] + self.step), (self.y_approx1[i - 1] + self.step * k3))

                self.y_approx1.append(self.y_approx1[i - 1] + self.step * (k1 + 2 * k2 + 2 * k3 + k4) / 6)

                k1 = self.substituteDiff(self.x[i-1], self.y_exact[i - 1])
                k2 = self.substituteDiff((self.x[i-1] + self.step / 2), (self.y_exact[i - 1] + self.step * k1 / 2))
                k3 = self.substituteDiff((self.x[i-1] + self.step / 2), (self.y_exact[i - 1] + self.step * k2 / 2))
                k4 = self.substituteDiff((self.x[i-1] + self.step), (self.y_exact[i - 1] + self.step * k3))

                self.y_approx2.append(self.y_exact[i - 1] + self.step * (k1 + 2 * k2 + 2 * k3 + k4) / 6)
```

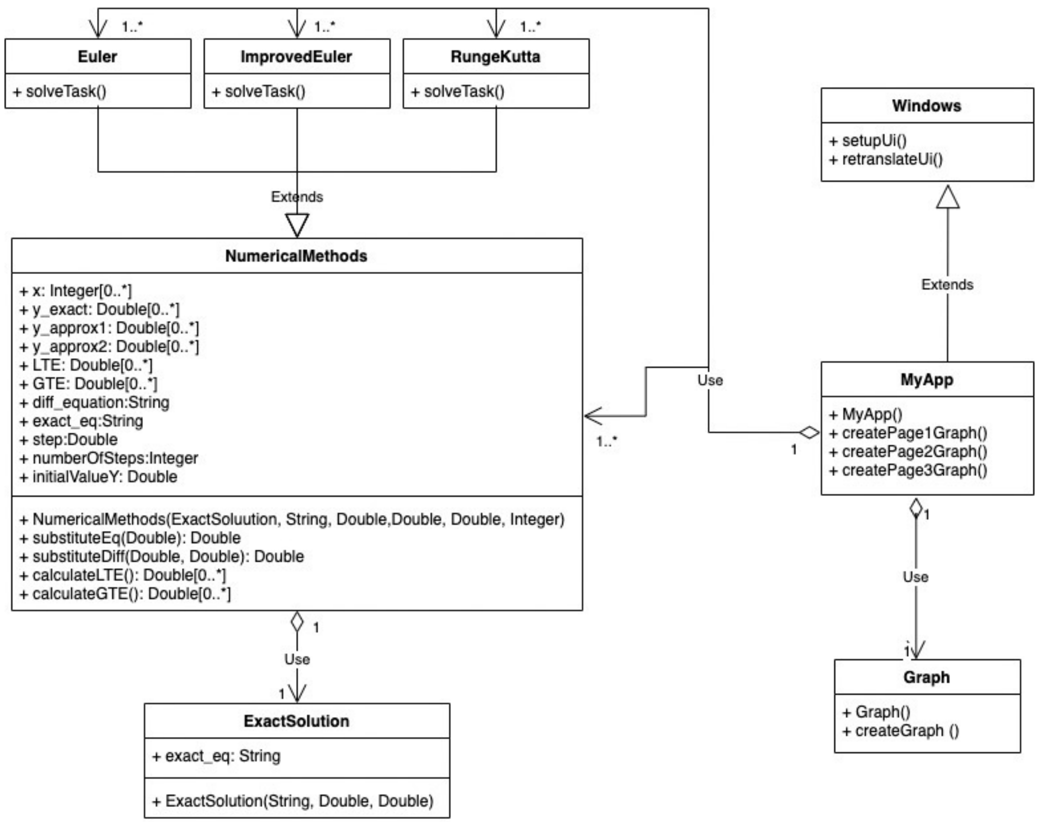
All updates controls by MyApp in Controller.py. When user clicks on buttons, then equations are calculated and displayed on the graph. Depending on the widget and button, the user calls the createPage1Graph (), createPage2Graph () and createPage3Graph () functions. Each function contains code that plots x_array and y_array and then creates a graph.

```python
class MyApp(QMainWindow, Window):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.setWindowTitle("Analyzing Differential Equation")

        self.graph_page1 = Graph(self.graph_page1)
        self.graph_page1.createGraph([], [], [])

        self.graph_page2 = Graph(self.graph_page2)
        self.graph_page2.createGraph([], [], [])

        self.graph_page3 = Graph(self.graph_page3)
        self.graph_page3.createGraph([], [], [])

        self.button_page1.clicked.connect(self.createPage1Graph)
        self.button_page2.clicked.connect(self.createPage2Graph)
        self.button_page3.clicked.connect(self.createPage3Graph)

        self.save_exact_eq = self.exact_equation.toPlainText()
```

Graph creation implementation - Graph.py:

```python
class Graph(QWidget):
    def __init__(self, parent=None, dpi=75):
        super(Graph, self).__init__(parent)
        self.labels = {'r': "Exact Solution", 'b': "Euler", 'g': "ImprovedEuler", 'y': "RungeKutta"}
        self.figure = Figure(dpi=dpi)
        self.canvas = Canvas(self.figure)
        self.toolbar = NavigationToolbar(self.canvas, self)

        layout = QVBoxLayout()
        layout.addWidget(self.toolbar)
        layout.addWidget(self.canvas)

        self.setLayout(layout)

    def createGraph(self, x, y, color):
        self.figure.clear()
        ax = self.figure.add_subplot(1, 1, 1)

        ax.grid(which='minor')
        ax.grid(which='major')

        for i in range(len(y)):
            ax.plot(x, y[i], color[i], label=self.labels[color[i]])
        if len(y) != 0:
            ax.legend()

        self.canvas.draw()
```

4) UML:



Euler
+ solveTask()
ImprovedEuler
+ solveTask()
RungeKutta
+ solveTask()
1..*
1..*
1..*
Extends
NumericalMethods
+ x: Integer[0..*]
+ y_exact: Double[0..*]
+ y_approx1: Double[0..*]
+ y_approx2: Double[0..*]
+ LTE: Double[0..*]
+ GTE: Double[0..*]
+ diff_equation:String
+ exact_eq:String
+ step:Double
+ numberOfSteps:Integer
+ initialValueY: Double
+ NumericalMethods(ExactSoluution, String, Double,Double, Double, Integer)
+ substituteEq(Double): Double
+ substituteDiff(Double, Double): Double
+ calculateLTE(): Double[0..*]
+ calculateGTE(): Double[0..*]
1
Use
1
ExactSolution
+ exact_eq: String
+ ExactSolution(String, Double, Double)
Windows
+ setupUi()
+ retranslateUi()
Extends
Use
1..*
1
MyApp
+ MyApp()
+ createPage1Graph()
+ createPage2Graph()
+ createPage3Graph()
1
Use
1
Graph
+ Graph()
+ createGraph ()