COMPUTATIONAL PRACTICUM

Nabiullin Damir B20-03

1) Exact solution:

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

1) $y' - \frac{1}{x}y = -xe^{\frac{y}{x}}$

$Remark: x \neq 0$

2) Solve complementary equation:

$$y_1' - \frac{1}{x}y_1 = 0 \quad => \quad y_1' = \frac{1}{x}y_1$$

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

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

$$y = xC, \qquad C \to C(x)$$

$$y = xC(x), \qquad y' = xC'(x) + C(x)$$

3) Substitute into initial equation:

$$xC'(x) + C(x) - \frac{1}{x}xC(x) = -xe^{\frac{xC(x)}{x}}$$

$$xC'(x) = -xe^{\frac{xC(x)}{x}}, \quad as\ x \neq 0\ we\ can\ divide\ by\ x$$

$$C'(x) = -e^{\frac{xC(x)}{x}} \quad => \quad C'(x) = -e^{C(x)}$$

$$-\int e^{-C}\,dC = \int dx \quad => \quad \frac{1}{e^C} = x + C_2$$

$$C = -\ln(x + C_2)$$

4) Substitute C:

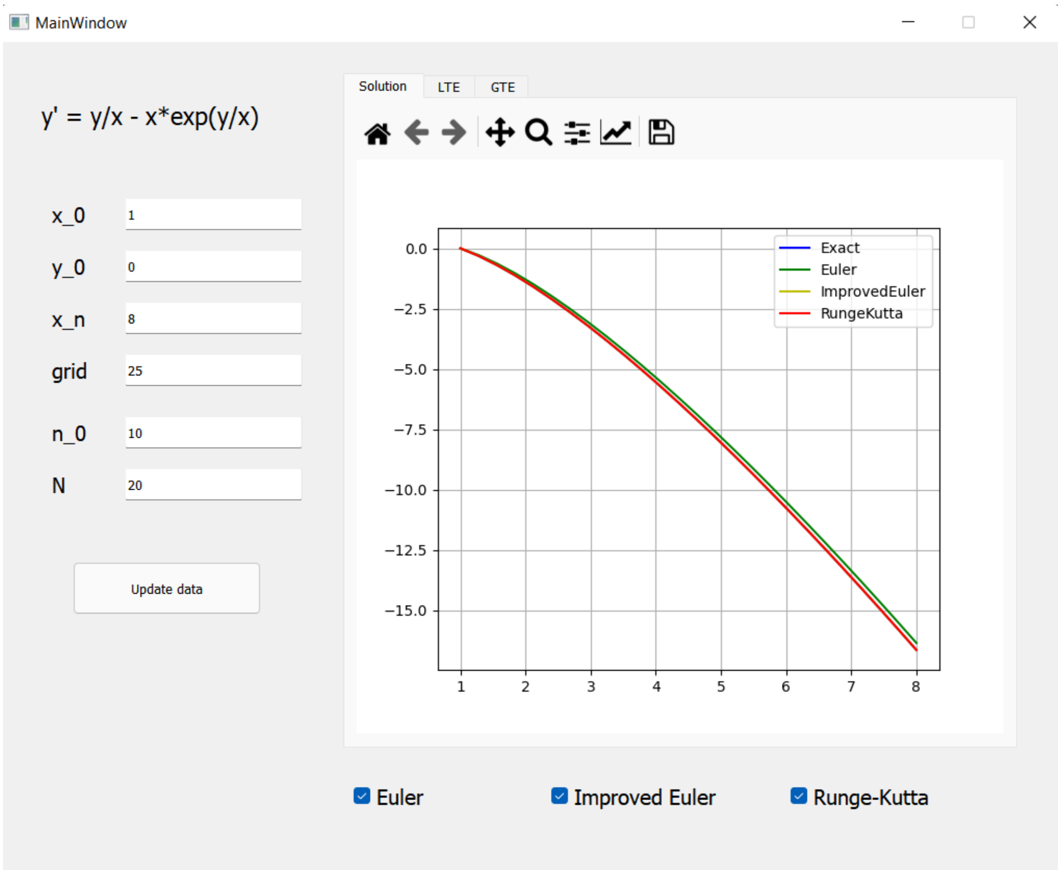
$$y = xC = -x\ln(x + C_2) \quad => \quad -\frac{y}{x} = \ln(x + C_2), \;\; x + C_2 > 0$$

$$e^{-\frac{y}{x}} = x + C_2 \quad => \quad C_2 = e^{-\frac{y}{x}} - x, \quad for\ our\ initial\ conditions: C_2 = 0$$

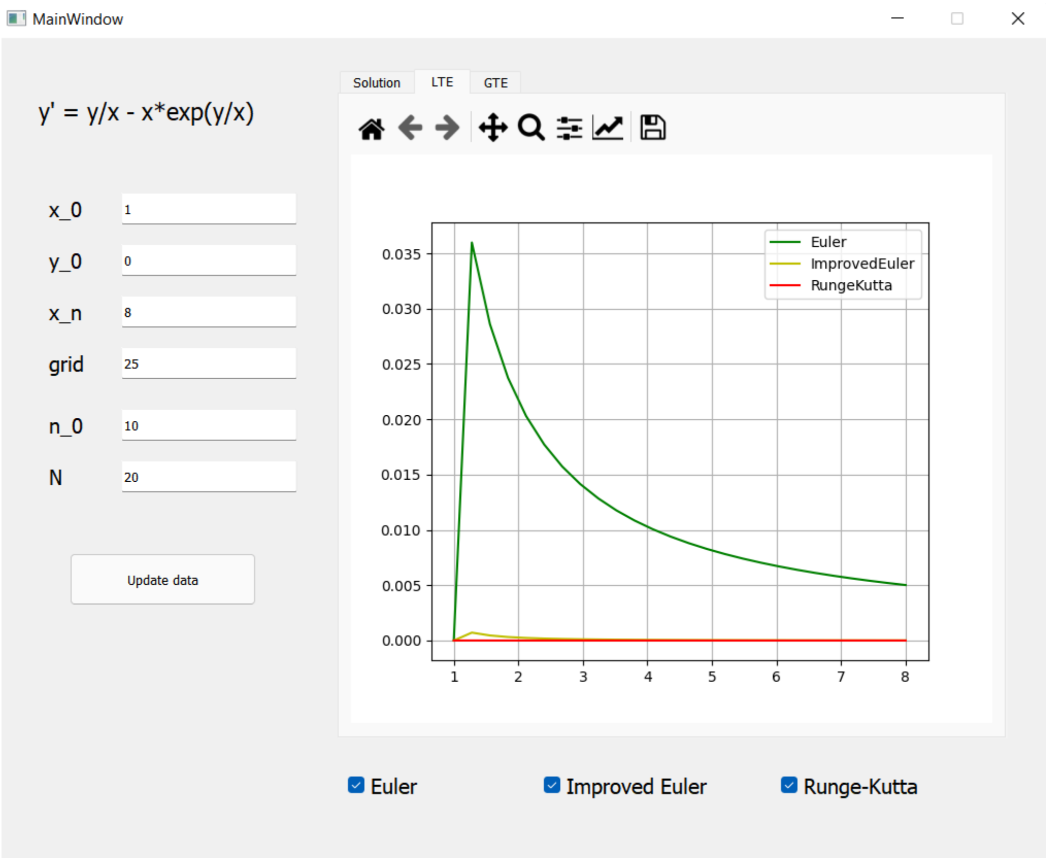
5) Exact solution: $y = -x\ln(x)$, $where\ x > 0$

2) Graphs from app:

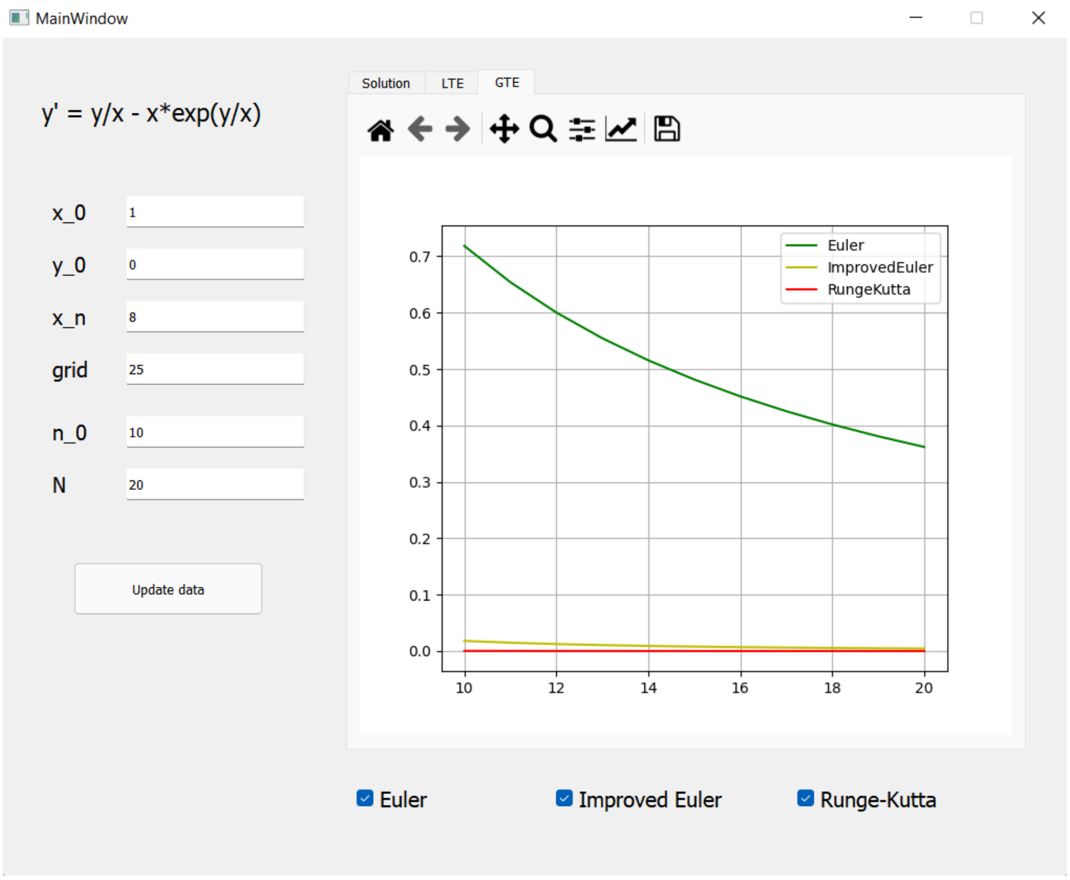
Solutions:



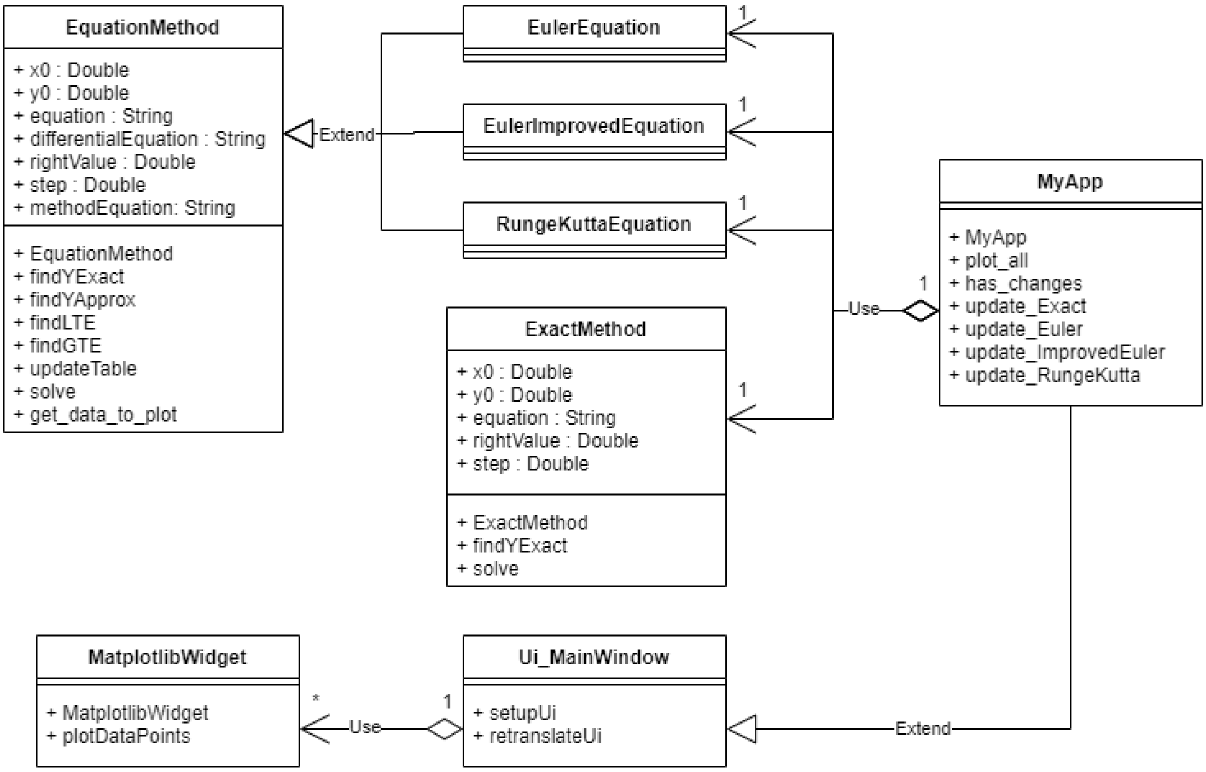
LTE:



GTE:



3) UML diagram



4) Code description

My project has settings.py, where you can put your own equations, initial app values, and title for app. As our C in equation depends on initial values, I created *find_const* function to evaluate constant for any initial values.

```python
y_0 = 0
x_0 = 1
X = 8
grid = 10
n_0 = 10
N = 20
title = 'y\' = y/x - x*exp(y/x)'
equation = '(-x*ln(x+C))'
general_equation = '(-x*ln(x+C))'
diff_equation = '((y/x)-(x*ep(y/x)))'
constant = 'ep(-y/x)-x'
```

```python
def find_const(x, y):
    global equation, general_equation, constant
    temp_eq = constant[:]
    temp_eq = temp_eq.replace('y', str(y))
    if x == 0.0:
        x = 0.1
    temp_eq = temp_eq.replace('x', str(x))
    temp_eq = temp_eq.replace('ep', 'exp')
    c = parse_expr(temp_eq).evalf()
    equation = general_equation.replace('C', str(c))
```

In my project you can find NumericalMethods.py with *EquationMethod*, *EulerEquation*, *EulerImprovedEquation*, and *RungeKuttaEquation* classes. *EquationMethod* class – the main class for all numerical methods in my program. It has all methods and fields that needed for numerical solutions.

EulerEquation, *EulerImprovedEquation*, and *RungeKuttaEquation* classes extends from *EquationMethod* class. But each of them has their own *methodEquation* – complete equation for approximate solution.

```python
class EulerEquation(EquationMethod):
    def __init__(self, equation, differentialEquation, color='g', leftValue=0, rightValue=0, step=0.1, x0=0, y0=0, roundNum=8):
        EquationMethod.__init__(self, equation, differentialEquation, color, leftValue, rightValue, step, x0, y0, roundNum)
        self.methodEquation = 'y + h*(' + self.differentialEquation + ')'


class EulerImprovedEquation(EquationMethod):
    def __init__(self, equation, differentialEquation, color='y', leftValue=0, rightValue=0, step=0.1, x0=0, y0=0, roundNum=8):
        EquationMethod.__init__(self, equation, differentialEquation, color, leftValue, rightValue, step, x0, y0, roundNum)
        self.methodEquation = differentialEquation
        self.methodEquation = self.methodEquation.replace('x', '(x + h/2)')
        self.methodEquation = self.methodEquation.replace('y', '(y + (h*' + self.differentialEquation + ')/2)')
        self.methodEquation = 'y + h*(' + self.methodEquation + ')'


class RungeKuttaEquation(EquationMethod):
    def __init__(self, equation, differentialEquation, color='r', leftValue=0, rightValue=0, step=0.1, x0=0, y0=0, roundNum=8):
        EquationMethod.__init__(self, equation, differentialEquation, color, leftValue, rightValue, step, x0, y0, roundNum)

        self.k1 = '(' + differentialEquation + ')'

        self.k2 = differentialEquation.replace('x', '(x + h/2)')
        self.k3 = self.k2[:]
        self.k2 = self.k2.replace('y', '(y + h*' + self.k1 + '/2)')
        self.k3 = self.k3.replace('y', '(y + h*' + self.k2 + '/2)')

        self.k4 = differentialEquation.replace('x', '(x + h)')
        self.k4 = self.k4.replace('y', '(y + h*' + self.k3 + ')')

        self.methodEquation = 'y + h*(' + self.k1 + ' + 2*' + self.k2 + ' + 2*' + self.k3 + ' + ' + self.k4 + ')/6'
```

Moreover, *EquationMethod* has function *findGTE.* Program use this function to find GTE for each number of grid cells from n_0 to N.

```python
def findGTE(self, n_0, N):
    self.gte_arr = []
    for i in range(n_0, N+1):
        self.step = (self.rightValue-self.x0)/i
        x, y = self.x0, self.y0
        gte = 0
        j = round(self.leftValue + self.step, self.roundNum)
        while j < self.rightValue + self.step / 2:
            if (abs(x+self.step) < 0.05) or (abs(x+self.step/2) < 0.05):
                x += 1.5*self.step

            if abs(round(x, self.roundNum)) <= 0.1:
                x = 0.1

            self.findYExact(x)
            self.findYApprox(x, y)

            if (abs(x+self.step) < 0.05) or (abs(x+self.step/2) < 0.05):
                x += 1.5*self.step
            else:
                x += self.step

            y = self.Y
            gte = abs(self.y - self.Y)
            j = round(j + self.step, self.roundNum)
        self.gte_arr.append(gte)
    return [self.gte_arr, self.color]
```

In App.py MyApp class generate GUI, check change in fields of GUI, and update data for graphs (if needed).

```python
def has_changes(self):
    if (self.x0 == float(self.lineEdit.text()) and
            self.y0 == float(self.lineEdit_2.text()) and
            self.rightValue == float(self.lineEdit_3.text()) and
            self.grid == float(self.lineEdit_6.text()) and
            self.step == (self.rightValue - self.x0) / self.grid and
            self.n_0 == int(self.lineEdit_4.text()) and
            self.N == int(self.lineEdit_5.text())):
        return False
    else:
        self.x0 = float(self.lineEdit.text())
        self.y0 = float(self.lineEdit_2.text())
        self.rightValue = float(self.lineEdit_3.text())
        self.grid = float(self.lineEdit_6.text())
        self.step = (self.rightValue - self.x0) / self.grid
        self.n_0 = int(self.lineEdit_4.text())
        self.N = int(self.lineEdit_5.text())
        return True

def update_Exact(self):
    self.Exact = ExactSolution.ExactMethod(settings.equation, x0=self.x0, y0=self.y0,
                                           rightValue=self.rightValue, step=self.step)
    self.x, self.y = self.Exact.solve()
    self.start = False
    self.solutions[0] = [self.x, self.y, 'b', 'Exact']
```

MatplotlibWidget class in PlotDesign.py – create widget of matplotlib plot for PyQt5. This class has function, that plot all needed graphs.

```python
class MatplotlibWidget(QtWidgets.QWidget):
    def __init__(self, parent=None, dpi=100):
        super(MatplotlibWidget, self).__init__(parent)

        self.figure = Figure(dpi=dpi)
        self.canvas = Canvas(self.figure)
        self.toolbar = NavigationToolbar(self.canvas, self)

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

        self.setLayout(layout)

    def plotDataPoints(self, data=None):
        self.figure.clear()
        ax = self.figure.add_subplot(1, 1, 1)

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

        for i in data:
            if i is not None:
                x = i[0]
                y = i[1]
                color = i[2]
                ax.plot(x, y, color, label=f'{i[3]}')
        ax.legend()
        self.canvas.draw()
```