Arslanov Shamil
Report title: finding exact and approximate solution to differential equation.

Aim: solve Cauchy initial problem for given task, in practice by using any framework to calculate approximations, local and global truncation error and plot them.

Table of content:

1. Analytical solution

$y' = e^{2x} + e^{x} + y^{2} - 2ye^{x}$ (1) $y(0) = 0$, $x = 15$.

ricotti equation

Let's try $y_p = a \cdot e^{bx}$

$y_p' = ab\ e^{bx}$

(1): $abe^{bx} = e^{2x} + e^{x} + a^{2}e^{2bx} - 2ae^{(b+1)x}$

$b = 1$: $e^{2x} + a^{2}e^{2x} - 2ae^{2x} = 0 \Rightarrow a = 1$

$abe^{bx} = e^{x} \Rightarrow a = 1$

$\}\ y_p = e^{x}$ partial solution.

$\Rightarrow y = y_p + u(x) = e^{x} + u$.

(1): $u' + e^{x} = e^{2x} + e^{x} + u^{2} + e^{2x} + 2ue^{x} - 2ue^{x} - 2e^{2x}$

$u' = u^{2}$

$\frac{du}{u^{2}} = dx \Rightarrow x = -\frac{1}{u} + C$, $u = \frac{1}{c-x}$ $(x \neq c) \Rightarrow$

$y = e^{x} + \frac{1}{c-x}$ $(c \in \mathbb{R},\ x \neq c)$

$y(0) = 1 + \frac{1}{c} = 0 \Rightarrow c = -1 \Rightarrow$

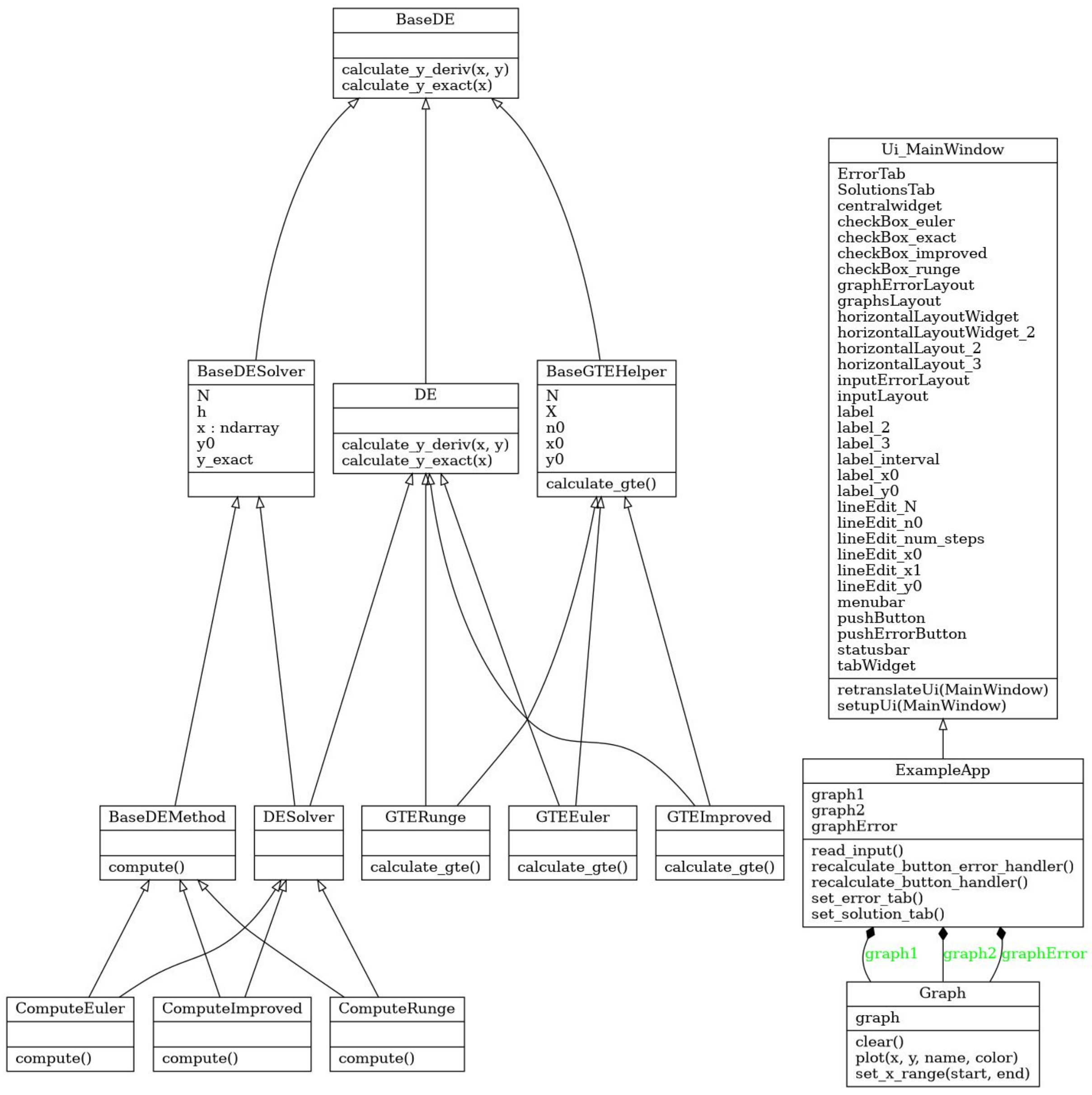
$y = e^{x} + \frac{1}{-1-x} = e^{x} - \frac{1}{x+1}$ $(x \neq -1)$ is the most general solution to (1)

When we solve u' = u^2, we also have solution u = 0 or y = e^x, but is does not satisfy initial conditions y(0)=0 (e^0 = 1)

2. Calculating approx. solutions

In this assignment I implemented numerical methods for solving DE using PyQT framework. BaseDE, BaseDESolver, BaseDEMethod, BaseGTEHelper are extendible interfaces for the euler, improved and runge methods.

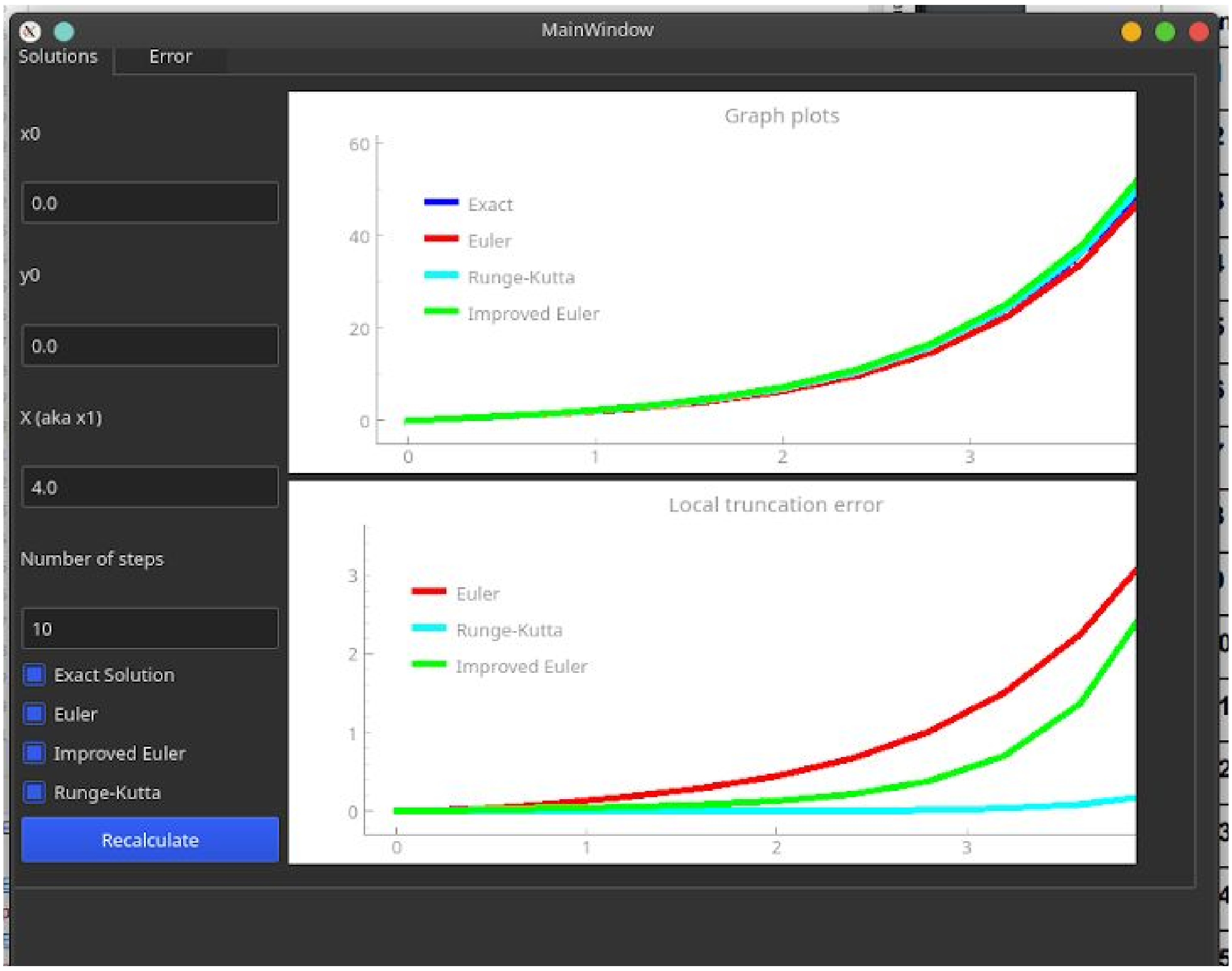
2.1 UML diagram of the program



2.2 Screenshots of the program

x0, y0, X, number of steps, n0, N are inputs, and we should mark which method we need to plot (exact, Euler, ect.)

At the tab Solutions we could plot exact and approximations, at tab Errors we could plot Max GTEs at each step.



x0

0.0

y0

0.0

X (aka x1)

4.0

Number of steps

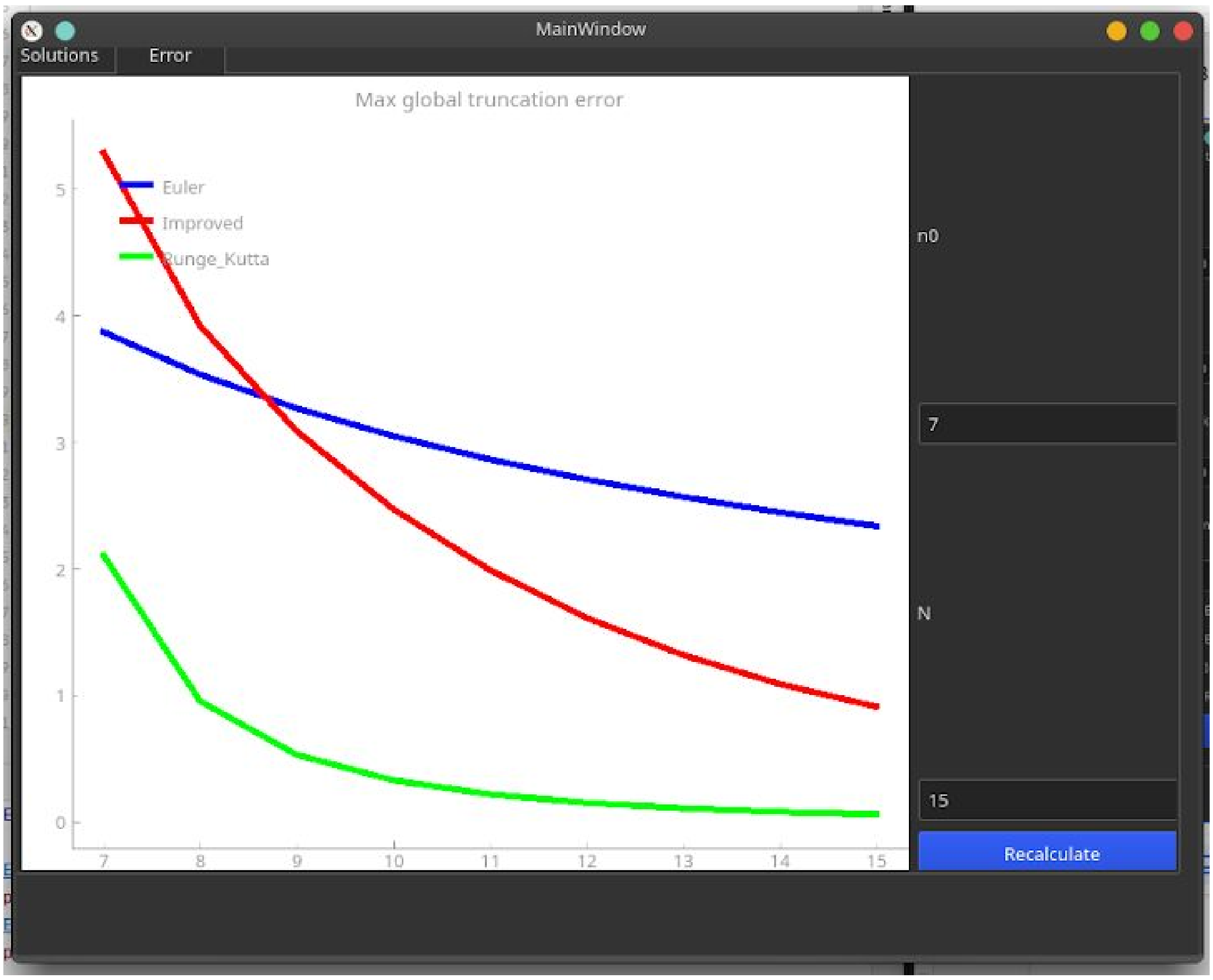
10

☑ Exact Solution

☑ Euler

☑ Improved Euler

☑ Runge-Kutta



2.3 Interesting parts of code

```
class BaseDE:

    @abstractmethod
    def calculate_y_deriv(self, x, y):
        pass

    @abstractmethod
    def calculate_y_exact(self, x):
        pass
```

```
class BaseDESolver(BaseDE, ABC):
    def __init__(self, x0, y0, N, x1):
        super().__init__()
        self.x = np.linspace(x0, x1, N + 1, dtype=np.float64)  # May be N
        self.N = N + 1
        self.y0 = y0
        self.h = (x1 - x0) / N
        self.y_exact = self.calculate_y_exact(self.x)


class BaseDEMethod(BaseDESolver):

    @abstractmethod
    def compute(self):
        pass
```

```python
class ComputeEuler(BaseDEMethod, DESolver):
   def __init__(self, x0, y0, N, x1):
       super().__init__(x0, y0, N, x1)

   def compute(self):
       y_appr = np.empty(shape=[self.N], dtype=np.float64)
       lte = np.zeros(shape=[self.N], dtype=np.float64)
       y_appr[0] = self.y0
       lte[0] = self.y_exact[0]

       for i in range(self.N - 1):
           k1 = self.calculate_y_deriv(self.x[i], y_appr[i])

           y_appr[i + 1] = y_appr[i] + self.h * k1

           k1 = self.calculate_y_deriv(self.x[i], self.y_exact[i])

           lte[i + 1] = self.y_exact[i] + self.h * k1

       lte = np.abs(lte - self.y_exact)

       return self.x, y_appr, lte
```

```python
class Graph:
   def __init__(self, title):
       self.graph = pg.PlotWidget()
       self.graph.setBackground('w')
       self.graph.setTitle(title)
       self.graph.addLegend()

   def set_x_range(self, start, end):
       self.graph.setXRange(start, end)

   def plot(self, x, y, name, color):
       self.graph.plot(x, y, name=name, pen=pg.mkPen(color, width=5))

   def clear(self):
       self.graph.clear()
```

```python
class ExampleApp(QtWidgets.QMainWindow, Ui_MainWindow):
   ...

   def set_solution_tab(self):

       self.graphsLayout.addWidget(self.graph1.graph)
       self.graphsLayout.addWidget(self.graph2.graph)

       self.tabWidget.setTabText(0, "Solutions")

       self.pushButton.clicked.connect(self.recalculate_button_handler)
```

```python
def recalculate_button_handler(self):
    x0, y0, N, x1 = self.read_input()

    self.graph1.clear()
    self.graph2.clear()
    self.graph1.set_x_range(x0, x1)
    self.graph2.set_x_range(x0, x1)

    if self.checkBox_exact.isChecked():
        x_ex = np.linspace(x0, x1)
        y_ex = DE().calculate_y_exact(x_ex)

        self.graph1.plot(x_ex, y_ex, name="Exact", color='b')

    if self.checkBox_euler.isChecked():
        x_appr, y_appr, lte = ComputeEuler(x0, y0, N, x1).compute()

        self.graph1.plot(x_appr, y_appr, name="Euler", color='r')
        self.graph2.plot(x_appr, lte, name="Euler", color='r')

    if self.checkBox_runge.isChecked():
        x_appr, y_appr, lte = ComputeRunge(x0, y0, N, x1).compute()

        self.graph1.plot(x_appr, y_appr, name="Runge-Kutta", color='c')
        self.graph2.plot(x_appr, lte, name="Runge-Kutta", color='c')

    if self.checkBox_improved.isChecked():
        x_appr, y_appr, lte = ComputeImproved(x0, y0, N, x1).compute()

        self.graph1.plot(x_appr, y_appr, name="Improved Euler", color='g')
        self.graph2.plot(x_appr, lte, name="Improved Euler", color='g')
```

3. Conclusion

During this computational assignment I have learned a new framework PyQT for working with GUI, applied mathematical knowledge in calculating approximations and errors (local and global), applied OOP principles of SOLID for solving given tasks. Substituting different values of interval, I found that errors of approximation are not high if interval is not big (from 0 to 4-5). Also I found that Runge-Kutta method converges more quickly to exact solution compared with other approximations.