BS19-F20-DE Computational Practicum

Danila Danko (d.danko@innopolis.university)

October 16, 2020

1. Analytical solution

Problem statement

Given I.V.P. $\begin{cases} y'(x) = \dfrac{y}{x} - xe^{\frac{y}{x}} \\ y(1) = 0 \\ x \in (1, 8) \end{cases}$, 1. Find its exact solution

2. Analyse points of discontinuity, if they exist

1. Exact Solution

Notice that x $\neq 0$ on $(1, 8)$

Let $t = \dfrac{y}{x}$, so that $t' = \dfrac{y'x - y}{x^2} = \left(y' - \dfrac{y}{x}\right)\dfrac{1}{x}$ and $\left(y' - \dfrac{y}{x}\right) = t'x$

Substitute t, t' into the original equation $y' - \dfrac{y}{x} = -xe^{\frac{y}{x}}$ to get $t'x = -xe^t$, $t' = -e^t$, $\dfrac{dt}{dx} = -e^t$

By separating variables, obtain $e^{-t}dt = -dx$.

Hence, $\displaystyle\int e^{-t}dt = \int -dx$, $-e^{-t} = -x + C$, $e^{-t} = x + C$, $-t = \ln(x + C)$, and $t = -\ln(x + C)$

Finally, $\dfrac{y}{x} = -\ln(x + C)$ which means that $y = -x\ln(x + C)$

Now, to determine the value of C, substitute the given $x = 1$ and $y(1) = 0$ into the equation.

$0 = -ln(1 + C) \quad \Rightarrow \quad C = 0$

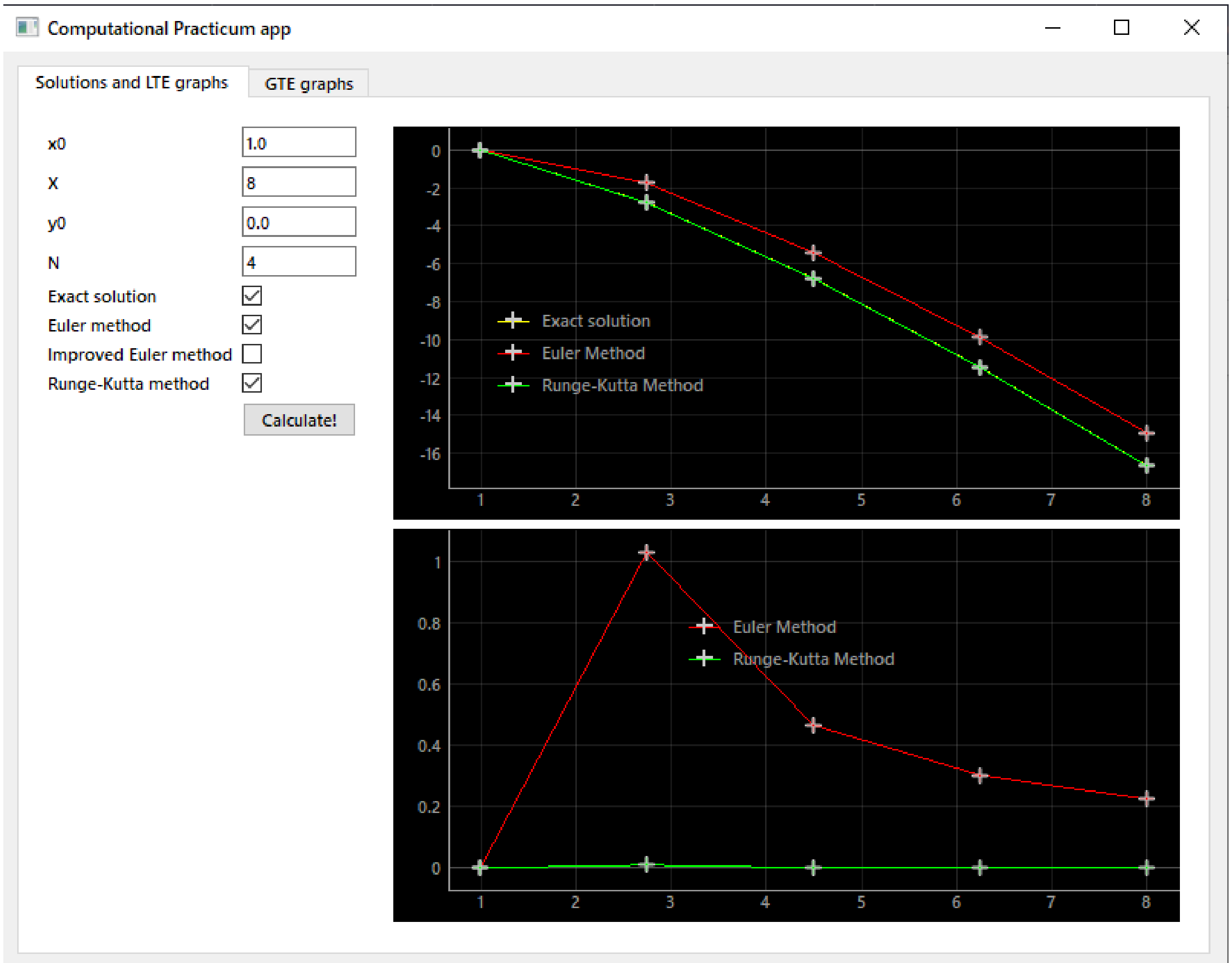
We can now write the exact solution to this **I.V.P**: $y = -x\ln(x)$

2. Analysis of discontinuity points

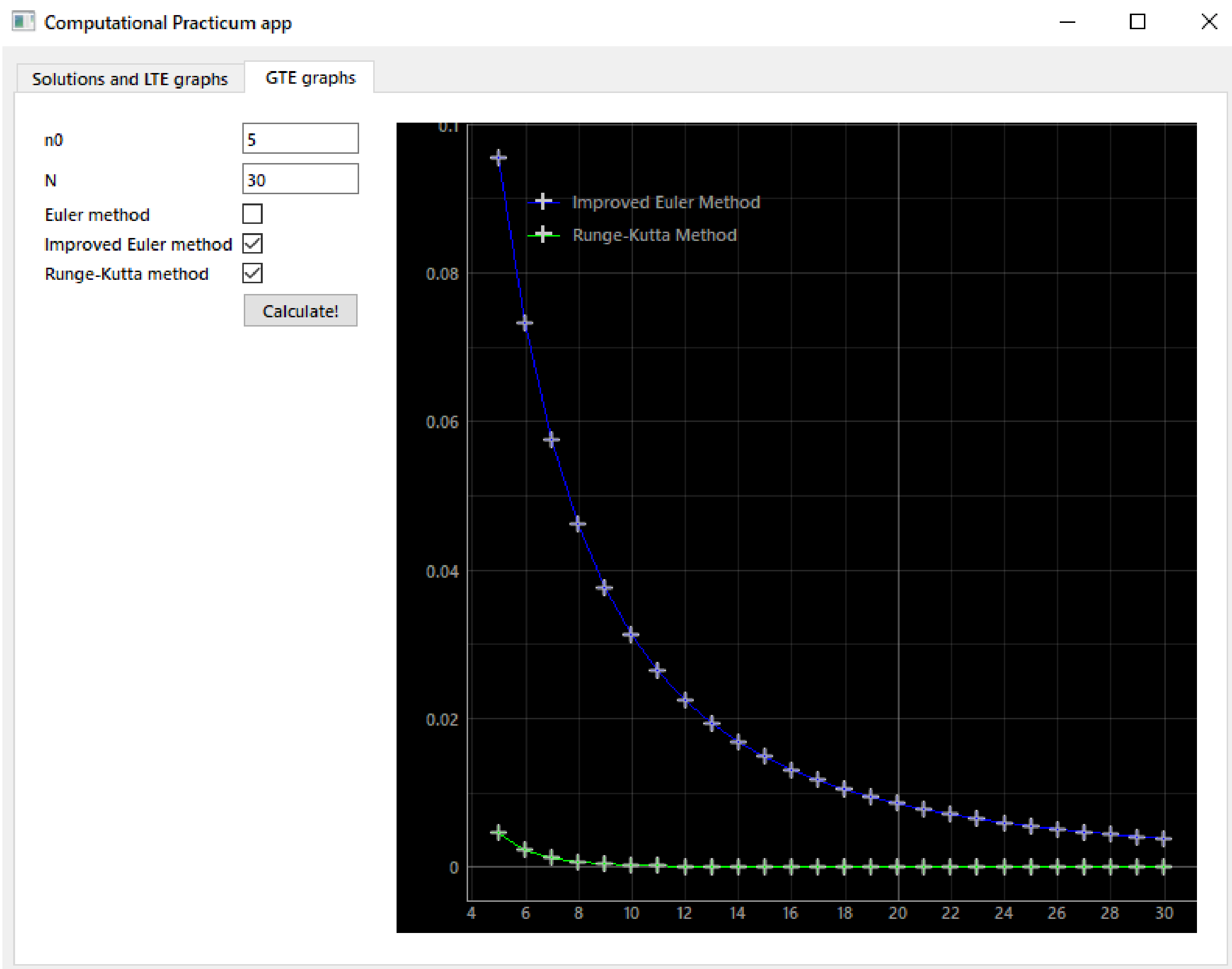
As for discontinuities, since y is a product of two functions that are continuous on $(1, 8)$, y is also continuous on the given interval. Therefore, there are no points of discontinuity on $(1, 8)$

2. My app's GUI

Tab 1. Plots of solutions and LTEs



Tab 2. Plots of maximal GTEs



3. Implementation details

Source code

I leave it in my repository on github

UML diagram of classes

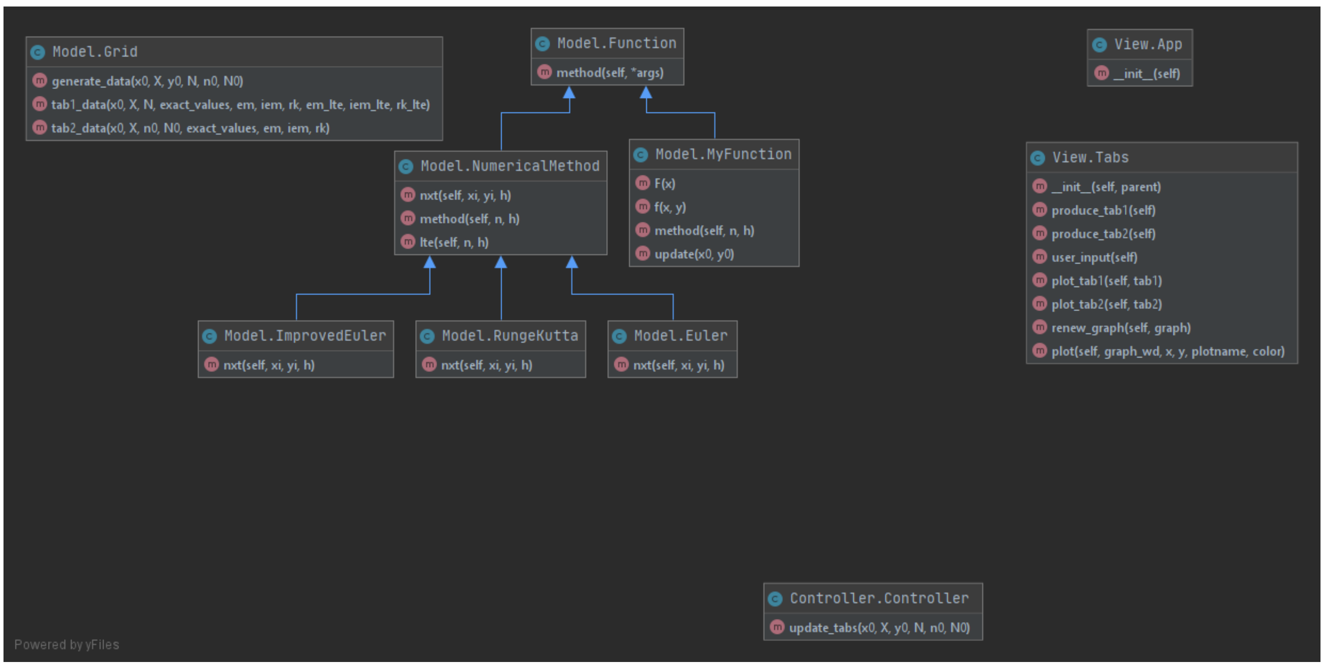
I implemented the `Model - View - Controller` scheme in Python programming language.

The libraries `pyqtgraph, PyQt5, numpy,` and `pandas` were used to achieve this goal

In the upper-left corner, there is a group of classes related to `Model`.

Similarly, the upper-right corner shows the `View` classes.

Finally, the bottom-most entity is the `Controller`.



The most interesting parts of code

Controller

`Controller` has only one method that accesses several `PyQt5.QtWidgets.QLineEdit`-s from `View`. It then extracts user input, processes it and asks `Model` to change its state and return the updated values. `Controller` then passes them to `View`, so that the latter can update itself properly.

```python
class Controller:

    @staticmethod
    def update_tabs(x0, X, y0, N, n0, N0):

        # processing user input

        x0 = float(x0.text())
        X = float(X.text())
        y0 = float(y0.text())
        N = int(N.text())

        n0 = int(n0.text())
        N0 = int(N0.text())

        # Updating Model in accordance with user input
        return Grid.generate_data(x0, X, y0, N, n0, N0)
```

Model

My `Model` contains the whole "business logic". `Model` has an initial state, with which `View` is initialized (default values in input fields).

These classes provide methods for recalculating all necessary plot points. You can see the inheritance hierarchy here.

```python
class Function: ...
class MyFunction(Function): ...
class NumericalMethod(Function): ...
class Euler(NumericalMethod): ...
class ImprovedEuler(NumericalMethod): ...
class RungeKutta(NumericalMethod): ...
```

A more important class is

```python
class Grid:
    @staticmethod
        def generate_data(x0, X, y0, N, n0, N0):

        # putting definitions and updating Model
        x0, X, y0 = float(x0), float(X), float(y0)
        MyFunction.update(x0, y0)

        exact_values = MyFunction().method
        em, iem, rk = Euler().method, ImprovedEuler().method, RungeKutta().method
        em_lte, iem_lte, rk_lte = Euler().lte, ImprovedEuler().lte, RungeKutta().lte

        # gathering plot data for tabs
        tab1 = Grid.tab1_data(x0, X, N, exact_values, em, iem, rk, em_lte, iem_lte, rk_lte)
        tab2 = Grid.tab2_data(x0, X, n0, N0, exact_values, em, iem, rk)

        return tab1, tab2
```

`Grid` updates the `Model` and returns the `pandas.DataFrame`-s `tab1` and `tab2` that contain all plot points for tabs of my application. They will be passed by `Controller` to `View`.

View

In the View layout, there is a `QPushButton "Calculate"`. When user presses it, the following methods are executed:

```
def user_input(self):
    # sending user input to Controller
    (tab1, tab2) = Controller.update_tabs(self.x0, self.X, self.y0, self.N, self.n0, self.N0)
    self.plot_tab1(tab1)
    self.plot_tab2(tab2)

def plot_tab2(self, tab2):
    #   plotting tab 2
    self.renew_graph(self.g)

    ns = tab2['ns']

    if self.em_check2.isChecked():
        self.plot(self.g, ns, tab2['em_gte'], 'Euler Method', 'r')
    if self.iem_check2.isChecked():
        self.plot(self.g, ns, tab2['iem_gte'], 'Improved Euler Method', 'b')
    if self.rk_check2.isChecked():
        self.plot(self.g, ns, tab2['rk_gte'], 'Runge-Kutta Method', 'g')

def renew_graph(self, graph):
    graph.clear()
    graph.showGrid(x=True, y=True)

def plot(self, graph_wd, x, y, plotname, color):
    pen = qtg.mkPen(color=color)
    graph_wd.addLegend()
    graph_wd.plot(x, y, name=plotname, pen=pen, symbol='+', symbolSize=10, symbolBrush=(color))
```

The method `user_input` fetches renewed plot data in dataframes `tab1, tab2`, and then calls method `renew_graph` to instanteneously clear a `pyqtgraph.PlotWiget` (the `plot_tab1` call is very similar). In `plot_tab2`, I use `QCheckBox`-es (their `isChecked()` method, to be more precise) to decide whether to `plot` a certain graph.

These were the most memorable parts of my code. Further details can be found in my github repository.