BS19-F20-DE Computational Practicum

Danila Danko (d.danko@innopolis.university)

October 18, 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.

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

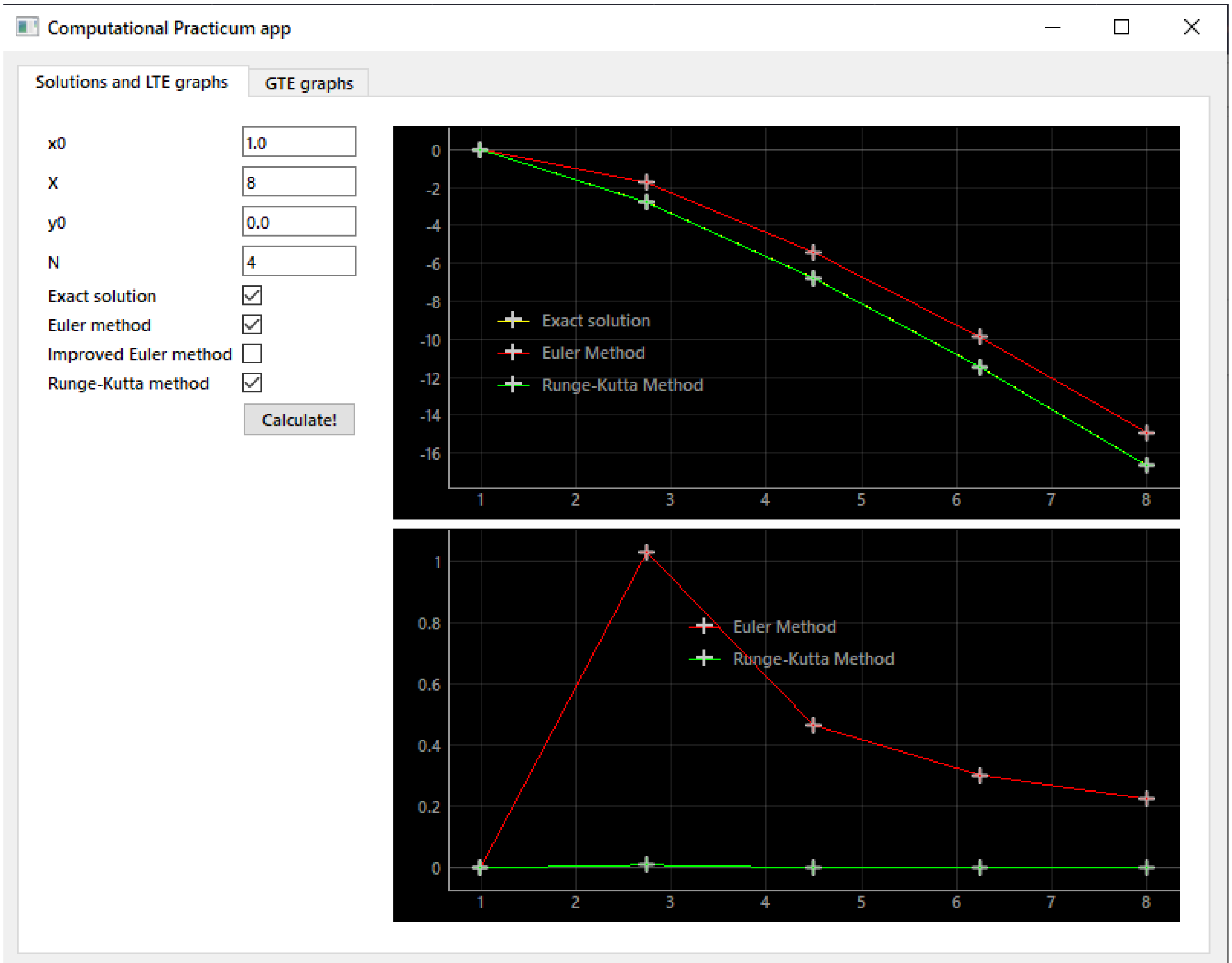
We can now write the exact solution to this **I.V.P**: $y = -x\ln(x)$

2. Analysis of discontinuity points

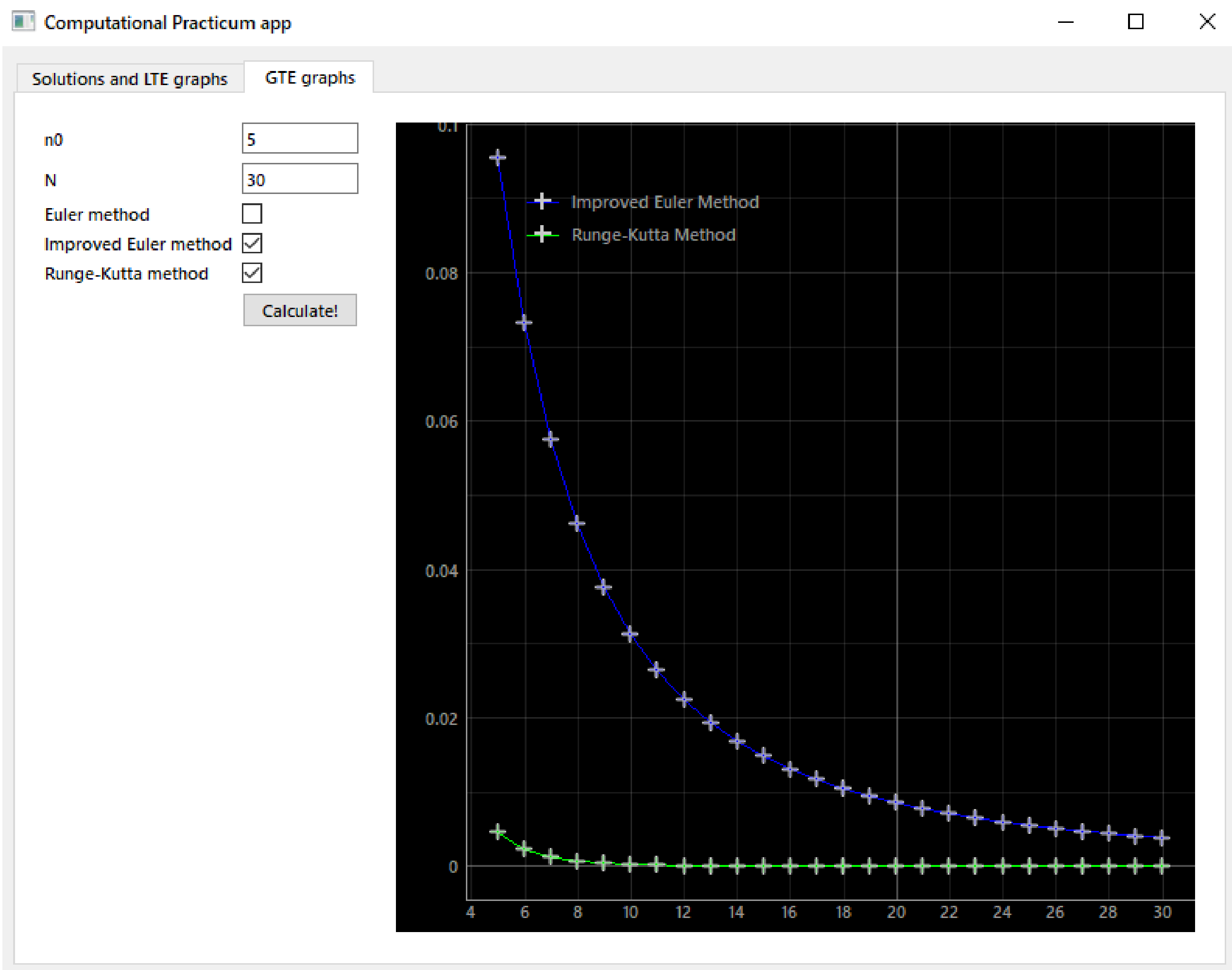
As for discontinuities, since y is a product of two functions that are continuous on $(1, 8)$,
y is also continuous on the given interval. Therefore, there are no points of discontinuity on $(1, 8)$

2. My app's GUI

Tab 1. Plots of solutions and LTEs



Tab 2. Plots of maximal GTEs



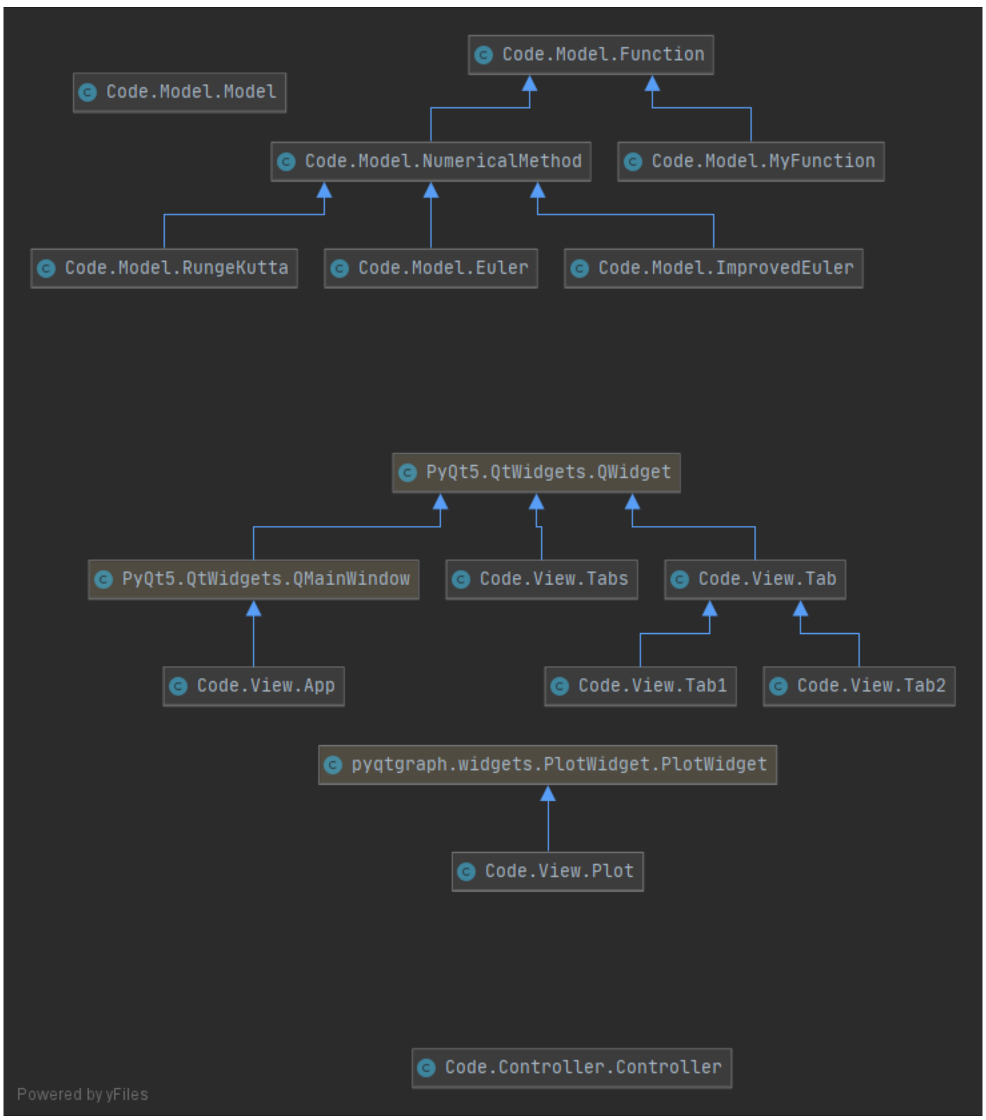
3. Implementation details

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. The topmost group of classes is related to `Model`. Similarly, the middle entities belong to `View` classes group. Finally, the bottom-most element is the `Controller`.



The most interesting parts of code

Controller

`Controller` has only one method that accesses several `PyQt5.QtWidgets.QLineEdit`-s from `View`, but it does not know anything else about `View`, though. It then extracts user input, processes it and asks `Model` to return its state, given several parameters. `Controller` then passes them to `View`, so that the latter can update itself properly.

```
from Code.Model import Model

class Controller:

    @staticmethod
    def model_state(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 Model.get_state(x0, X, y0, N, n0, N0)
```

Model

My `Model` part 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.

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

A more important class is

```
class Model:
    @staticmethod
    def get_state(x0, X, y0, N, n0, N0):

        # putting definitions and updating Model
        ExactSolution.update(x0, y0)

        exact = ExactSolution().values
        methods = (Euler().values, ImprovedEuler().values, RungeKutta().values)
        ltes = (Euler().lte, ImprovedEuler().lte, RungeKutta().lte)

        # gathering plot data for tabs
        tab1 = Model.tab1_data(x0, X, N, exact, methods, ltes)
        tab2 = Model.tab2_data(x0, X, n0, N0, exact, methods)

        return tab1, tab2
```

`get_state` returns the state of `Model` with given parameters in `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

`Tabs` has attributes `tab1` and `tab2`. Each of them, in turn, has a `QPushButton "Calculate"`. When user presses it in any tab of the app, `Tabs`' method `user_input` is executed:

```
class Tabs(QWidget):

    def __init__(self, parent):
        super().__init__(parent)
        self.layout = QVBoxLayout(self)

        # create tabs
        self.tabs = QTabWidget()

        self.tab1 = Tab1(self)
        self.tabs.addTab(self.tab1, self.tab1.name)

        self.tab2 = Tab2(self)
        self.tabs.addTab(self.tab2, self.tab2.name)

        self.layout.addWidget(self.tabs)
        self.setLayout(self.layout)

        self.user_input()

    def user_input(self):
        tab1_data, tab2_data = \
            Controller.get_model_state(self.tab1.x0, self.tab1.X, self.tab1.y0, self.tab1.N, self.tab2.r
        self.tab1.update_plots(tab1_data)
        self.tab2.update_plots(tab2_data)
```

The method `user_input` provides Controller with user input and receives from it the renewed plot data in dataframes `tab1_data, tab2_data`. Tabs `tab1` and `tab2` are inherited from `PyQt5.QtWidgets.QtWiget` and contain `Plot`(s) inherited from `pyqtgraph.PlotWiget`. They can update on their own, given necessary data.

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