Computational Practicum Report

Student: Emil Latypov
Group: B20-03
E-mail: e.latypov@innopolis.university

Part I

$$\begin{cases} y' = \frac{-y^2}{3} - \frac{2}{3x^2} \\ y(1) = 2 \\ x \in (1;5) \end{cases}$$

To solve this equation, we can use Riccati method:

$$y_1 = \frac{a}{x}$$

$$y'_1 = \frac{-a}{x^2}$$

$$\frac{-a}{x^2} + \frac{a^2}{3x^2} + \frac{2}{3x^2} = 0$$

$$-3a + a^2 + 2 = 0 \;\; \Rightarrow \left[\begin{array}{l} a=2 \\ a=1 \end{array}\right.$$

Let's take $a=1$

$$y_1 = \frac{1}{x} \;\; \Rightarrow \;\; y'_1 = \frac{-1}{x^2}$$

$$y = \frac{1}{x} + z(x) \;\; \Rightarrow \;\; y' = \frac{-1}{x^2} + z'$$

$$\frac{-1}{x^2} + z' = \frac{-\left(\frac{1}{x} + z\right)^2}{3} - \frac{2}{3x^2}$$

$$\frac{-1}{x^2} + z' = \frac{-\left(\frac{1}{x^2} + \frac{2z}{x} + z^2\right)}{3} - \frac{2}{3x^2}$$

$$\frac{-1}{x^2} + z' + \frac{1}{3x^2} + \frac{2z}{3x} + \frac{z^2}{3} + \frac{2}{3x^2} = 0$$

$$z'+\frac{2z}{3x}+\frac{z^2}{3}=0$$

$$z'+\frac{2z}{3x}=\left(\frac{-1}{3}\right)z^2 \qquad |\ *3$$

$$3z'+\frac{2z}{x}=-z^2$$

$$u=\frac{1}{z} \quad => \quad z=\frac{1}{u}$$

$$u'=\frac{-z'}{z^2} \quad => z'=-u'z^2=\frac{-u'}{u^2}$$

$$\frac{-3u'}{u^2}+\frac{2}{ux}=\frac{-1}{u^2} \quad |\ *u^2$$

$$-3u'+\frac{2u}{x}=-1$$ - linear

$$-3u'+\frac{2u}{x}=0$$ - complementary

$$u'=\frac{du}{dx} \quad => \frac{3du}{dx}=\frac{2u}{x} \quad =>\left(\frac{3}{2}\right)\int\frac{du}{u}=\int\frac{dx}{x}$$

$$\ln|u|=\left(\frac{2}{3}\right)\ln|x|+C$$

$$u=x^{\left(\frac{2}{3}\right)}C_1, \quad C_1=C_1(x)$$

$$u'=\frac{2C_1}{3\sqrt[3]{x}}+x^{\left(\frac{2}{3}\right)}C'_1$$

$$-3\left(\frac{2C_1}{3\sqrt[3]{x}}+x^{\left(\frac{2}{3}\right)}C'_1\right)+\frac{2x^{\left(\frac{2}{3}\right)}C_1}{x}=-1$$

$$\frac{-2C_1}{\sqrt[3]{x}}-3x^{\left(\frac{2}{3}\right)}C'_1+\frac{2x^{\left(\frac{2}{3}\right)}C_1}{x}=-1$$

$$3x^{\left(\frac{2}{3}\right)}C'_1=1$$

$$C'_1=\left(\frac{1}{3}\right)x^{\left(\frac{-2}{3}\right)}$$

$$C_1=\left(\frac{1}{3}\right)\int x^{\left(\frac{-2}{3}\right)}=\left(\frac{1}{3}\right)(3x^{\frac{1}{3}}+C_2)=x^{\frac{1}{3}}+C_3$$

$$u=x^{\frac{2}{3}}(x^{\frac{1}{3}}+C_3)=x+x^{\frac{2}{3}}C_3$$

$$z=\frac{1}{x+x^{\frac{2}{3}}C_3}$$

$$y=\frac{1}{x}+\frac{1}{x+x^{\frac{2}{3}}C_3}$$ - General Solution

Let's deduce C_3 from general solution

$$C_3=\frac{x^{\frac{1}{3}}(2-xy)}{xy-1}$$

Let's find exact solution. As we know y(1)= 2

$$C_3=\frac{1^{\frac{1}{3}}(2-1*2)}{1*2-1}=0$$

So, exact solution will be:

$$y=\frac{2}{x}$$

It is obvious that for our exact solution discontinuity will occur at point x = 0 (infinite discontinuity).
For general solution discontinuity will occur at x = 0 and x =
$-C_3^3$

Part II and III

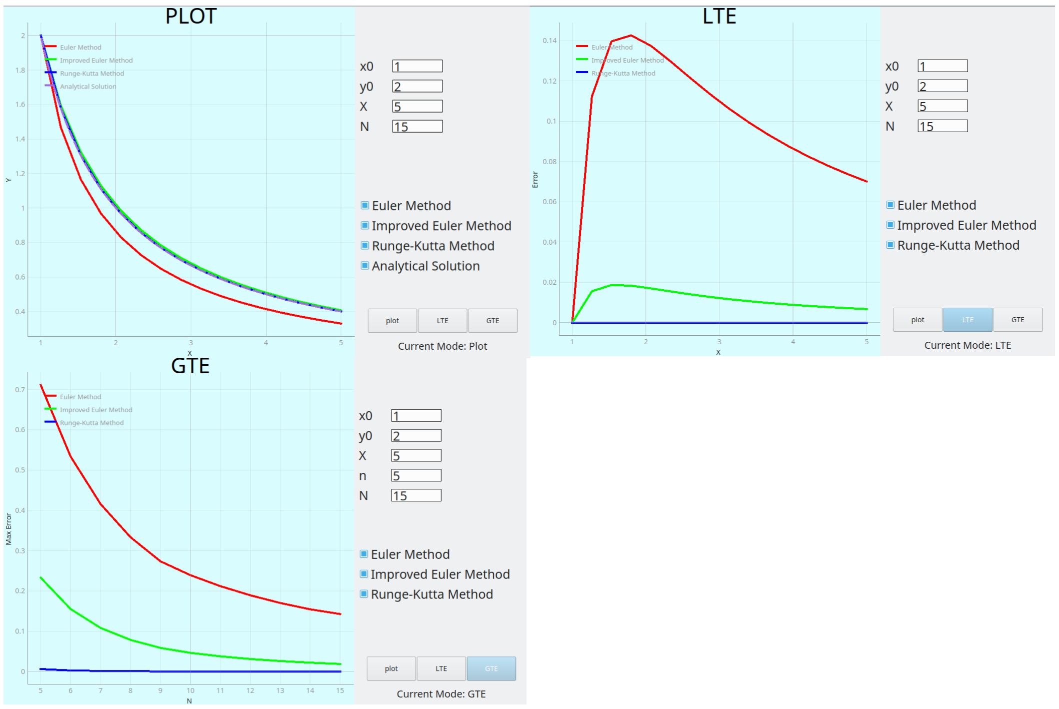
My application allows building graph of analytical solution, obtained above and apply Euler, Improved Euler and Runge-Kutta numerical methods to plot approximate graph from differential equation. Moreover, that gives us an ability to compare numerical methods and make some conclusions.

The project is written in python 3 language and
User Interface is created PyQt5 library. To run it, type:

(you should be in project root directory)

```
pip3 install requirements.txt
python3 main.py
```

After that you will see application window, which already plotted graph with some initial values.


PLOT

x0 1
y0 2
X 5
N 15

Current Mode: Plot

LTE

x0 1
y0 2
X 5
N 15

Current Mode: LTE

GTE

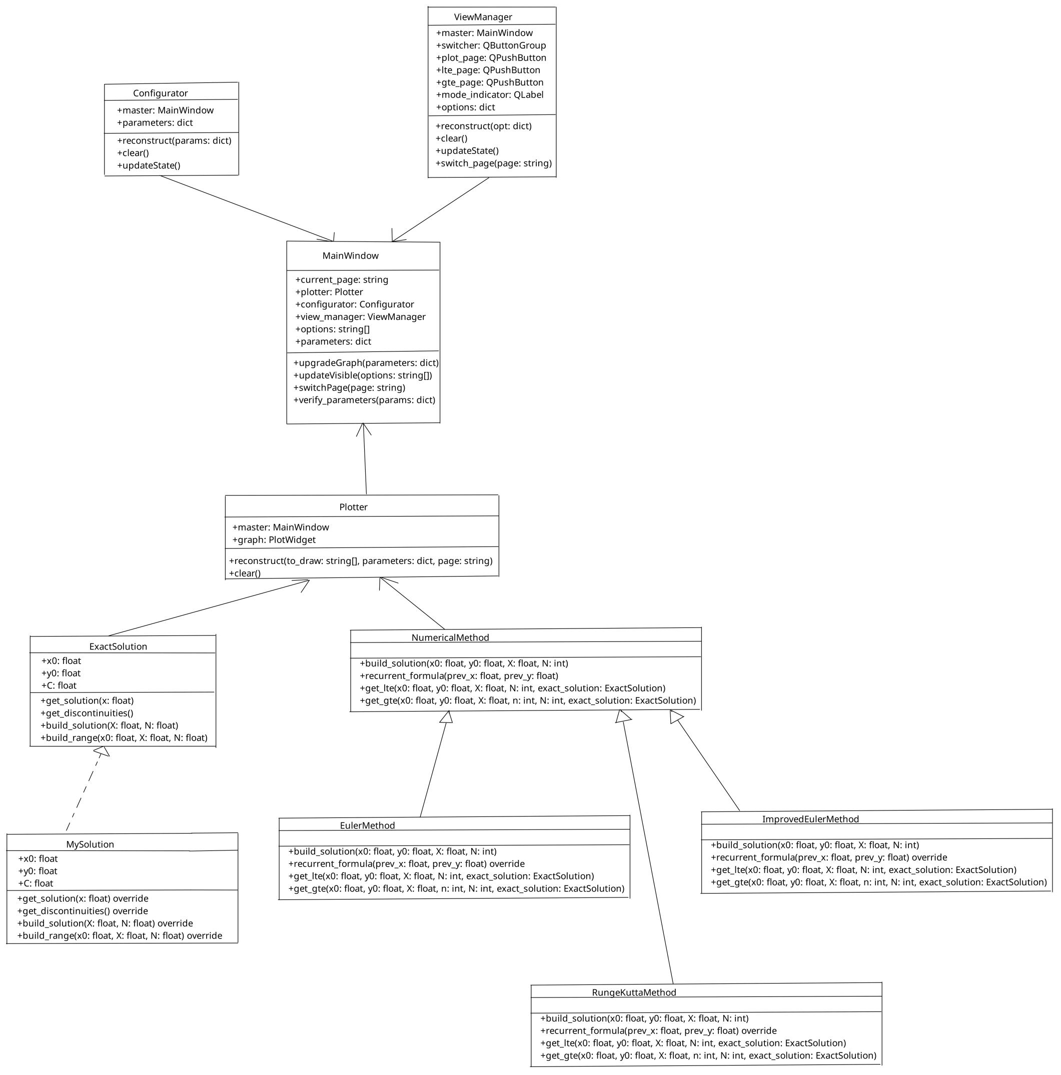
x0 1
y0 2
X 5
n 5
N 15

Current Mode: GTE

Here we can see plot on the left side, editable parameters, which methods to visualize and mode select (plot/LTE/GTE) on the right side. We can manipulate parameters, hide some methods and go to LTE and GTE page.

UML

The project has following UML diagram:


ViewManager
+master: MainWindow
+switcher: QButtonGroup
+plot_page: QPushButton
+lte_page: QPushButton
+gte_page: QPushButton
+mode_indicator: QLabel
+options: dict
+reconstruct(opt: dict)
+clear()
+updateState()
+switch_page(page: string)
Configurator
+master: MainWindow
+parameters: dict
+reconstruct(params: dict)
+clear()
+updateState()
MainWindow
+current_page: string
+plotter: Plotter
+configurator: Configurator
+view_manager: ViewManager
+options: string[]
+parameters: dict
+upgradeGraph(parameters: dict)
+updateVisible(options: string[])
+switchPage(page: string)
+verify_parameters(params: dict)
Plotter
+master: MainWindow
+graph: PlotWidget
+reconstruct(to_draw: string[], parameters: dict, page: string)
+clear()
ExactSolution
+x0: float
+y0: float
+C: float
+get_solution(x: float)
+get_discontinuities()
+build_solution(X: float, N: float)
+build_range(x0: float, X: float, N: float)
NumericalMethod
+build_solution(x0: float, y0: float, X: float, N: int)
+recurrent_formula(prev_x: float, prev_y: float)
+get_lte(x0: float, y0: float, X: float, N: int, exact_solution: ExactSolution)
+get_gte(x0: float, y0: float, X: float, n: int, N: int, exact_solution: ExactSolution)
MySolution
+x0: float
+y0: float
+C: float
+get_solution(x: float) override
+get_discontinuities() override
+build_solution(X: float, N: float) override
+build_range(x0: float, X: float, N: float) override
EulerMethod
+build_solution(x0: float, y0: float, X: float, N: int)
+recurrent_formula(prev_x: float, prev_y: float) override
+get_lte(x0: float, y0: float, X: float, N: int, exact_solution: ExactSolution)
+get_gte(x0: float, y0: float, X: float, n: int, N: int, exact_solution: ExactSolution)
ImprovedEulerMethod
+build_solution(x0: float, y0: float, X: float, N: int)
+recurrent_formula(prev_x: float, prev_y: float) override
+get_lte(x0: float, y0: float, X: float, N: int, exact_solution: ExactSolution)
+get_gte(x0: float, y0: float, X: float, n: int, N: int, exact_solution: ExactSolution)
RungeKuttaMethod
+build_solution(x0: float, y0: float, X: float, N: int)
+recurrent_formula(prev_x: float, prev_y: float) override
+get_lte(x0: float, y0: float, X: float, N: int, exact_solution: ExactSolution)
+get_gte(x0: float, y0: float, X: float, n: int, N: int, exact_solution: ExactSolution)

Code

The application has following structure: class `MainWindow` is responsible for all user interface action. It consists of 3 widgets: `Plotter`, `Configurator` and `ViewManager`, which are responsible for plotting graph, managing parameters (`x0`, `y0`, etc.) and managing which graphs should be plotted respectively. In case of `Configurator` and `ViewManager`, they are waiting for some changes made by user and then they are notifying parent `MainWindow` about that. Then `MainWindow` should make a decision what to do next: either redraw graph with other parameters or hide/show some methods in graph. For example, in this way `Configurator` notifies parent when some parameters are changed by user:

```
    def updateState(self):
        states = {}
        # collects states of all parameters (x0, y0, etc.)
        for param in self.parameters:
            # gets value of parameter
            states[param] = self.parameters[param][1].text()
        # makes a signal for parent that some changes have to be applied
        self.master.updateGraph(states)
```

In some cases `Configurator` and `ViewManager` are also can be redrawn: for example, when we switch to GTE page, there are one additional parameter `n` is added to `Configurator` and `Analytical Solution` field is removed from `ViewManager` as it is not needed in this page. All 3 widgets provide method `reconstruct()`, so `MainWindow` can easily apply any changes.

Exact Solution

There is an abstract class `ExactSolution`, which provides methods to obtain `Analytical Solution`. Its implementation `MySolution` calculates constant `C` from initial `x0` and `y0` (formulas are obtained in my solution above), and calculates y(x) using obtained formula and calculated `C`:

```
...
# calculate constant C from initial x0 and y0
self.C = ((sign_x0 * abs(x0)**(1/3)) * (2 - x0 * y0)) / (x0 * y0 - 1)
...

def get_solution(self, x):
    # apply formula from analytical solution to get y(x)
    sign_x = 1 if x >=  0 else -1
    return 1 / x + 1 / (x + (sign_x * abs(x) ** (2 / 3)) * self.C)
```

Also `ExactSolution` provides method `build_range(x0, X, N)`, which generates arrays of `x` and `y` points to be plotted in `Plotter`

Numerical Methods

class `NumericalMethod` provides methods `build_solution`, `get_lte` and `get_gte` for arrays of points of graph, its local and global error respectively. It implements this methods (because they are the same for each numerical method) and only uses 1 abstract function `recurrent_formula`, which should be implemented in `EulerMethod`, `ImprovedEulerMethod` and `RungeKuttaMethod` classes.

```
...
# calculating current approximate y-value based on previous step
y_points += [y_points[i - 1] + self.recurrent_formula(x_points[i - 1], y_points[i
- 1], step)]
...

def dy_over_dx(x, y):
    # hard-coded equiation from task
    return - ((y * y) / 3) - 2 / (3 * x * x)
```

```
...
def recurrent_formula(self, prev_x, prev_y, step):
    # recurrent formula for Euler Method
    return step * dy_over_dx(prev_x, prev_y)
...
```

```
...
def recurrent_formula(self, prev_x, prev_y, step):
    # recurrent formula for Improved Euler Method
    return step * dy_over_dx(prev_x + step / 2, prev_y + (step / 2) *
dy_over_dx(prev_x, prev_y))
...
```

```
...
def recurrent_formula(self, prev_x, prev_y, step):
    # recurrent formula for Runge-Kutta Method
    half_step = step /  2

    k1 = dy_over_dx(prev_x, prev_y)
    k2 = dy_over_dx(prev_x + half_step, prev_y + half_step * k1)
    k3 = dy_over_dx(prev_x + half_step, prev_y + half_step * k2)
    k4 = dy_over_dx(prev_x + step, prev_y + step * k3)

    return (step / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
...
```

LTE

Local Error is calculated as difference between exact and approximate solutions (absolute value)

GTE

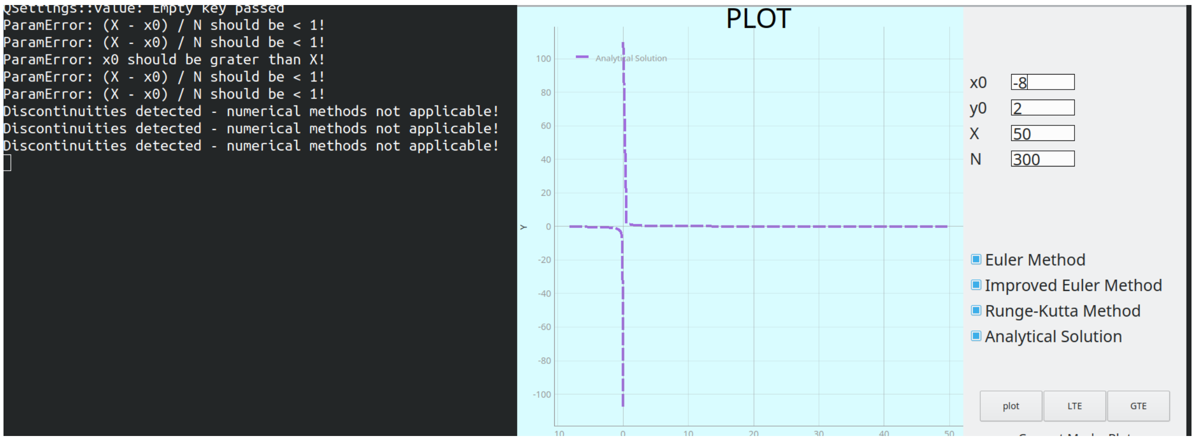
Global Error is calculated as max error for current number of steps

Workflow

The working process is following:

1. User makes some changes (for example, change parameter x0)
2. `Configurator` provides to `MainWindow` raw parameters
3. `MainWindow` verifies parameters: make sure that all inputs are correct and doesn't contradict to logic (for example, x0 can't be > than X, etc.)1
4. 1. If parameters are declined, user get notification and explanation in their **terminal window**
   2. Otherwise, depending on current page `MainWindow` forwards relevant configurations to `Plotter`
5. `Plotter` firstly searches for discontinuities and create intervals related to them
6. 1. If there are more than 1 interval (from x0 to X), then points of discontinuity are lying on this interval and numerical not applicable in this case. Make notification in **terminal window** and plot only analytical solution.
   2. If there is only one interval (from x0 to X), then we can successfully apply all numerical methods, so plot them.

   Note: All errors and notifications are printed in termial window!



```
QSettings::value: Empty key passed
ParamError: (X - x0) / N should be < 1!
ParamError: (X - x0) / N should be < 1!
ParamError: x0 should be grater than X!
ParamError: (X - x0) / N should be < 1!
ParamError: (X - x0) / N should be < 1!
Discontinuities detected - numerical methods not applicable!
Discontinuities detected - numerical methods not applicable!
Discontinuities detected - numerical methods not applicable!
```

PLOT

x0 -8
y0 2
X 50
N 300

Euler Method
Improved Euler Method
Runge-Kutta Method
Analytical Solution