Computational Assignment Report

Hasan Khadra - BS19-02

Nov 3, 2020

1 Exact Solution:

The given equation:

$$y' = (1 + \frac{y}{x}) \cdot \ln(\frac{x+y}{y}) + \frac{y}{x}$$

where $\frac{y}{x} > -1$, $x \neq 0$, $y_0 = 2$, $x_0 = 1$, $X = 6$. We can rewrite the equation as:

$$y' = (1 + \frac{y}{x}) \cdot \ln(1 + \frac{y}{x}) + \frac{y}{x} + 1 - 1 (*)$$

Now let $z = 1 + \frac{y}{x}$ where $z > 0$. Then we'll have

$$y = xz - x = x(z-1)$$

$$y' = z - 1 + xz'$$

Now we write $(*)$ again with substituting z as

$$z - 1 + xz' = z \cdot \ln(z) + z - 1$$

$$xz' = z \cdot \ln(z)$$

$$\frac{dz}{dx} \cdot x = z \cdot \ln(z)$$

$$\frac{dz}{z \cdot \ln(z)} = \frac{dx}{x}$$

$$\int \frac{dz}{z \cdot \ln(z)} = \int \frac{dx}{x}$$

$$\int \frac{d \ln(z)}{\ln(z)} = \int \frac{dx}{x}$$

$$\ln|\ln(z)| = \ln|x| + C$$

$$\ln z = Cx$$

$$z = e^{Cx}$$

where $C \in \mathbb{R}$. Now we substitute back to get $y = x(e^{Cx} - 1)$ where $x \neq 0$, $y > -x$, $C \in \mathbb{R}$. To find the constant C:

$$e^{Cx_0} = \frac{y_0}{x_0} + 1$$

$$C = \frac{1}{x_0} \cdot \ln(\frac{y_0}{x_0} + 1)$$

where $x_0 \neq 0$, $y_0 > -x_0$.

$$C = \ln(3)$$

$$y = x(3^x - 1)$$

where $x \in \mathbb{R}$. Substituting back we get the exact solution:

$$y = x \cdot (e^{\frac{x}{x_0} \cdot \ln(\frac{y_0}{x_0} + 1)} - 1)$$

where $x_0 \neq 0$, $y_0 > -x_0$. At the given interval $x \in (x_0, X) = (1, 6)$ there are no discontinuity points.

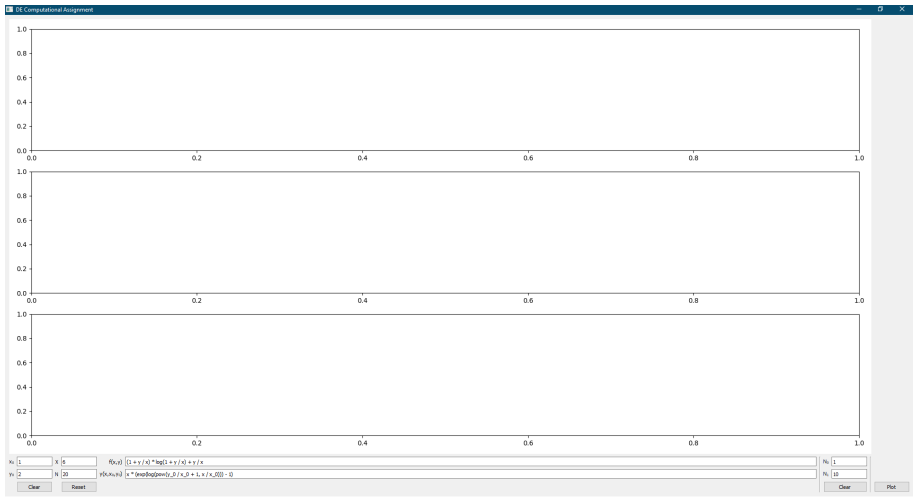
2 Application:

The program analyse different methods (Euler's method, improved Euler's method and Runge-Kutta), present them on the graph GUI and compare them to the exact solution of the given equation.

2.1 Packages and tools:

The program requires the following packages and tools: **Python 3.8**, **Matplotlib**, **NumPy**, **PyQt5**.

2.2 The graph interface:



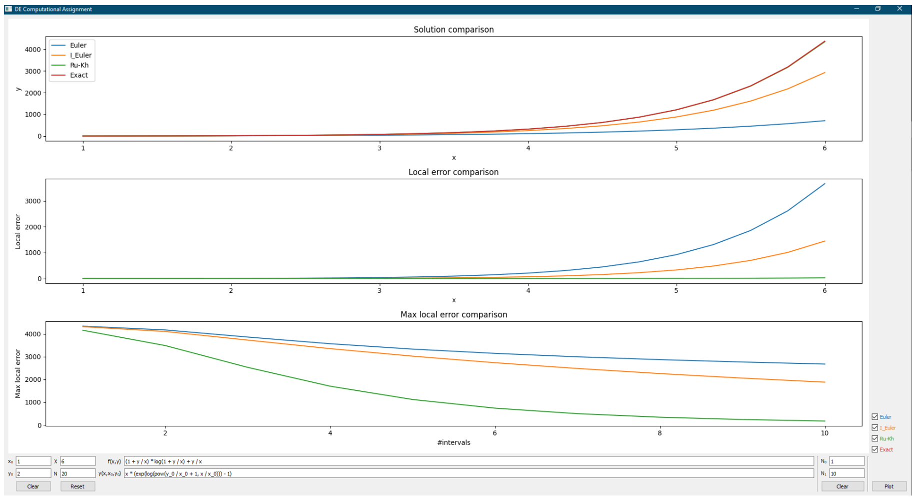
x_0: 1 X: 6 f(x,y): (1 + y / x) * log(1 + y / x) + y / x N_0: 1

y_0: 2 N: 20 $y(x,x_0,y_0)$: x * (exp(log(pow(y_0 / x_0 + 1, x / x_0))) - 1) N_1: 10

Figure 1: This is the default view of the graph GUI.

The GUI has the following features:

- The possibility of changing all the parameters (initial values, number of steps, etc..).
- The ability to show and hide any of the graphs.
- The ability to change the initial and exact function.



x_0 1 X 6 f(x,y) (1 + y / x) * log(1 + y / x) + y / x N_0 1

y_0 2 N 20 $y(x,x_0,y_0)$ x * (exp(log(pow(y_0 / x_0 + 1, x / x_0))) - 1) N_1 10

☑ Euler ☑ I_Euler ☑ Ru-Kh ☑ Exact

Figure 2: This is the GUI with all the graphs.

The parameters in the GUI:

- x_0, y_0: The initial values for the given equation.
- X: The end point for the given x interval.
- N: The number of steps that the methods will perform.
- $f(x, y)$: This is the same as y'.
- $y(x, x_0, y_0)$: The exact solution of the given IVP.
- N_0: Starting number of intervals.
- N_1: Ending number of intervals.

2.3 Program structure and UML diagrams:

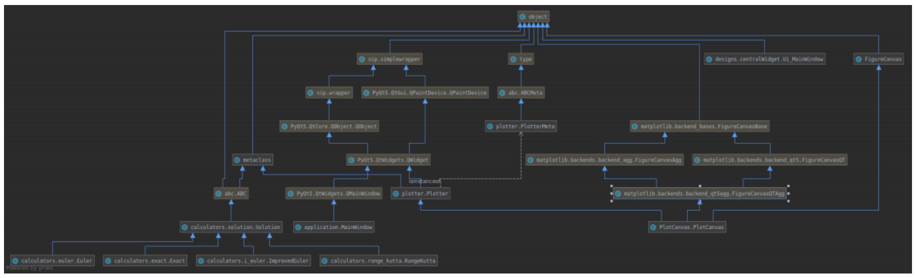

object
sip.simplewrapper
type
designs.centralWidget.Ui_MainWindow
FigureCanvas
sip.wrapper
PyQt5.QtGui.QPaintDevice.QPaintDevice
abc.ABCMeta
PyQt5.QtCore.QObject.QObject
plotter.PlotterMeta
matplotlib.backend_bases.FigureCanvasBase
metaclass
PyQt5.QtWidgets.QWidget
matplotlib.backends.backend_agg.FigureCanvasAgg
matplotlib.backends.backend_qt5.FigureCanvasQT
isinstanceof
abc.ABC
PyQt5.QtWidgets.QMainWindow
plotter.Plotter
matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg
calculators.solution.Solution
application.MainWindow
PlotCanvas.PlotCanvas
calculators.euler.Euler
calculators.exact.Exact
calculators.i_euler.ImprovedEuler
calculators.runge_kutta.RungeKutta

Figure 3: Packages diagram.

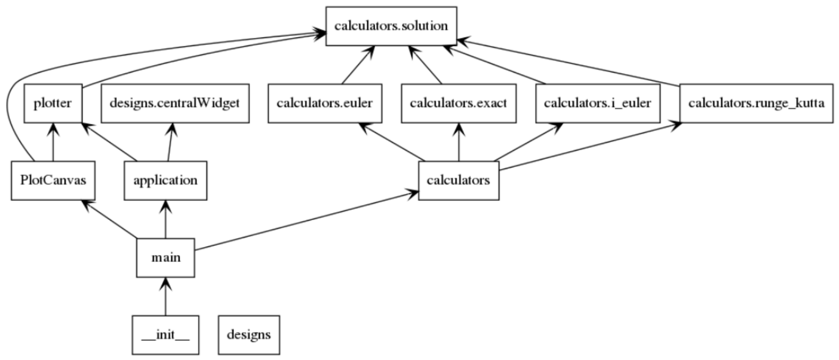


Figure 4: classes diagram.

Detailed structure of the application can be seen at package diagram, class diagram with internal classes included and expanded class diagram of newly created classes.

The source code obeys SOLID principle: every part is responsible over single part of functionality, all modules are extendable and interchangeable up to abstraction. The challenging part was to create an abstraction for a plotter class due to features in implementations of PyQt5 python integration and Python's classes.

In theory a plotter is a subclass of PyQt5.QWidget with some desired functions. Since QT originally implemented in C++, all PyQt classes are C++ bindings, not pure python types, thus multiple inheritance from a binding and a python class results into TypeError exception caused by super-classes' layout conflict at C level.

The trick is to create metaclass PlotterMeta that inherits from type(QWidget) that is a subclass of metaclass type, then create actual abstraction Plotter which is a subclass QWidget with a metaclass PlotterMeta.

Thus inheriting PlotCanvas from FigureCanvas (matplotlib integration) and Plotter produces no errors since both super-classes are subclasses of wrappertype and have compatible layout.

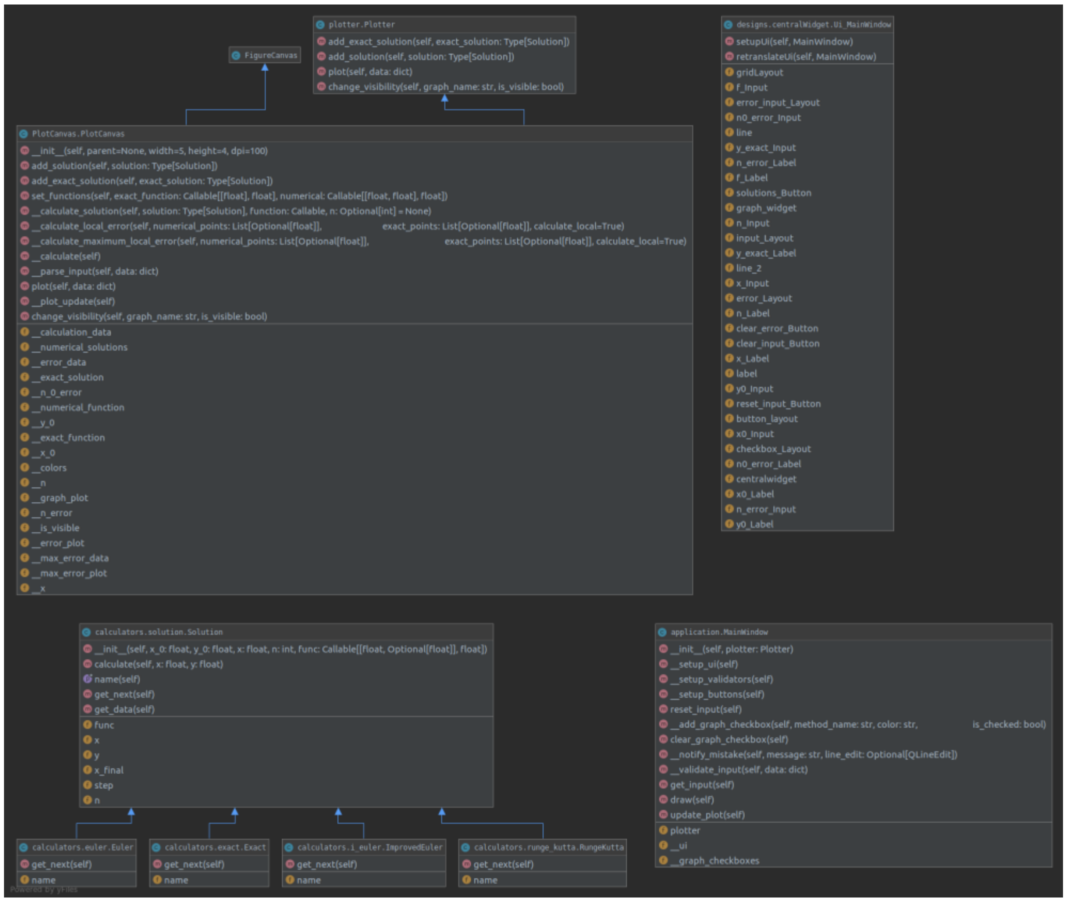
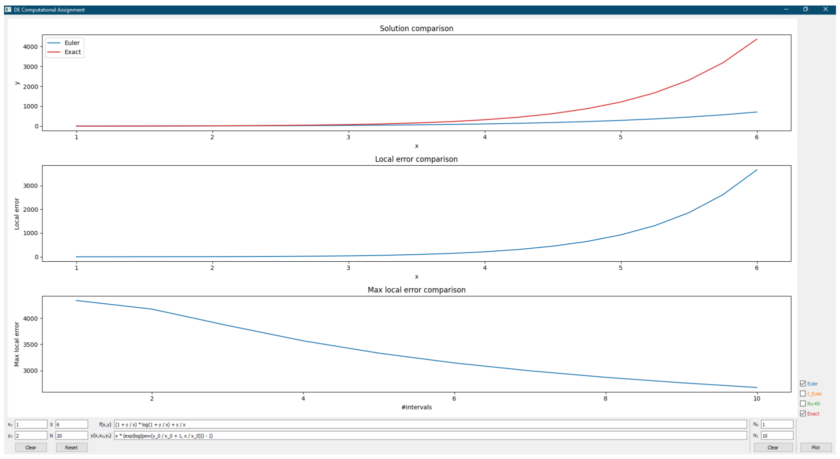

FigureCanvas
plotter.Plotter
add_exact_solution(self, exact_solution: Type[Solution])
add_solution(self, solution: Type[Solution])
plot(self, data: dict)
change_visibility(self, graph_name: str, is_visible: bool)
designs.centralWidget.Ui_MainWindow
setupUi(self, MainWindow)
retranslateUi(self, MainWindow)
gridLayout
f_Input
error_input_Layout
n0_error_Input
line
y_exact_Input
n_error_Label
f_Label
solutions_Button
graph_widget
n_Input
input_Layout
y_exact_Label
line_2
x_Input
error_Layout
n_Label
clear_error_Button
clear_input_Button
x_Label
label
y0_Input
reset_input_Button
button_layout
x0_Input
checkbox_Layout
n0_error_Label
centralwidget
x0_Label
n_error_Input
y0_Label
PlotCanvas.PlotCanvas
__init__(self, parent=None, width=5, height=4, dpi=100)
add_solution(self, solution: Type[Solution])
add_exact_solution(self, exact_solution: Type[Solution])
set_functions(self, exact_function: Callable[[float], float], numerical: Callable[[float, float], float])
__calculate_solution(self, solution: Type[Solution], function: Callable, n: Optional[int] = None)
__calculate_local_error(self, numerical_points: List[Optional[float]], exact_points: List[Optional[float]], calculate_local=True)
__calculate_maximum_local_error(self, numerical_points: List[Optional[float]], exact_points: List[Optional[float]], calculate_local=True)
__calculate(self)
__parse_input(self, data: dict)
plot(self, data: dict)
__plot_update(self)
change_visibility(self, graph_name: str, is_visible: bool)
__calculation_data
__numerical_solutions
__error_data
__exact_solution
__n_0_error
__numerical_function
__y_0
__exact_function
__x_0
__colors
__n
__graph_plot
__n_error
__is_visible
__error_plot
__max_error_data
__max_error_plot
__x
calculators.solution.Solution
__init__(self, x_0: float, y_0: float, x: float, n: int, func: Callable[[float, Optional[float]], float])
calculate(self, x: float, y: float)
name(self)
get_next(self)
get_data(self)
func
x
y
x_final
step
n
application.MainWindow
__init__(self, plotter: Plotter)
__setup_ui(self)
__setup_validators(self)
__setup_buttons(self)
reset_input(self)
__add_graph_checkbox(self, method_name: str, color: str, is_checked: bool)
clear_graph_checkbox(self)
__notify_mistake(self, message: str, line_edit: Optional[QLineEdit])
__validate_input(self, data: dict)
get_input(self)
draw(self)
update_plot(self)
plotter
__ui
__graph_checkboxes
calculators.euler.Euler
get_next(self)
name
calculators.exact.Exact
get_next(self)
name
calculators.i_euler.ImprovedEuler
get_next(self)
name
calculators.runge_kutta.RungeKutta
get_next(self)
name

Figure 5: Expanded classes diagram.

3 Analysis

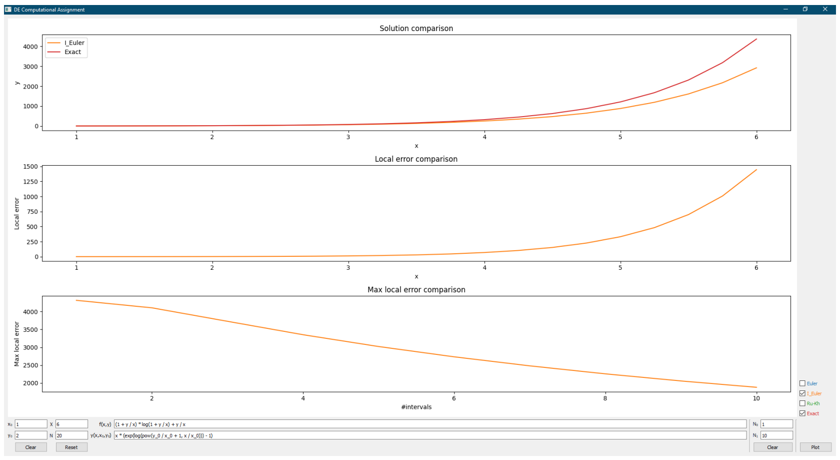


[✓ Euler] I_Euler Ru-Kh [✓ Exact]

x_0: 1 X: 6 f(x,y): (1 + y / x) * log(1 + y / x) + y / x N₀: 1

y_0: 2 N: 20 $y(x,x_0,y_0)$: x * (exp(log(pow(y_0 / x_0 + 1, x / x_0))) - 1) N₁: 10

Figure 6: Graphs plotting of the exact solution and Euler method.

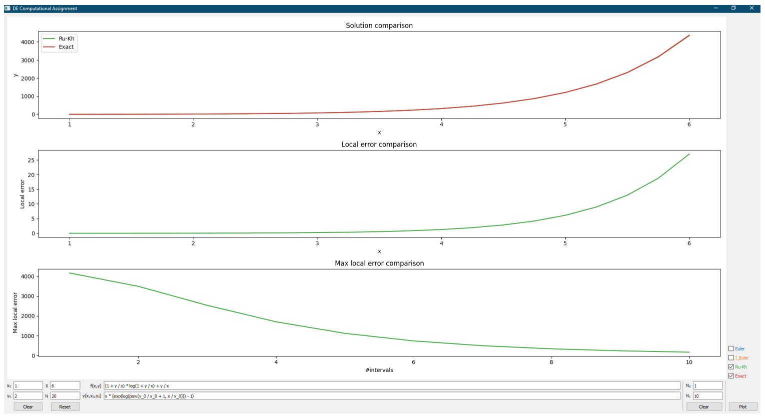


Euler [✓ I_Euler] Ru-Kh [✓ Exact]

x_0: 1 X: 6 f(x,y): (1 + y / x) * log(1 + y / x) + y / x N₀: 1

y_0: 2 N: 20 $y(x,x_0,y_0)$: x * (exp(log(pow(y_0 / x_0 + 1, x / x_0))) - 1) N₁: 10

Figure 7: Graphs plotting of the exact solution and Improved Euler method.



x_0: 1 X: 6 f(x,y): (1 + y / x) * log(1 + y / x) + y / x N_0: 1

y_0: 2 N: 20 $y(x,x_0,y_0)$: x * (exp(log(pow(y_0 / x_0 + 1, x / x_0))) - 1) N_1: 10

☐ Euler ☐ I_Euler ☑ Ru-Kh ☑ Exact

Figure 8: Graphs plotting of the exact solution and Runge-Kutta.

3.1 Euler method:

As seen in figure 6, it is clear that for given function Euler method's approximation has relatively big local error (up to 928.584) starting from $x = 3.5$ as well as global error (1792.719) due to using small amount of parameters to perform approximation.

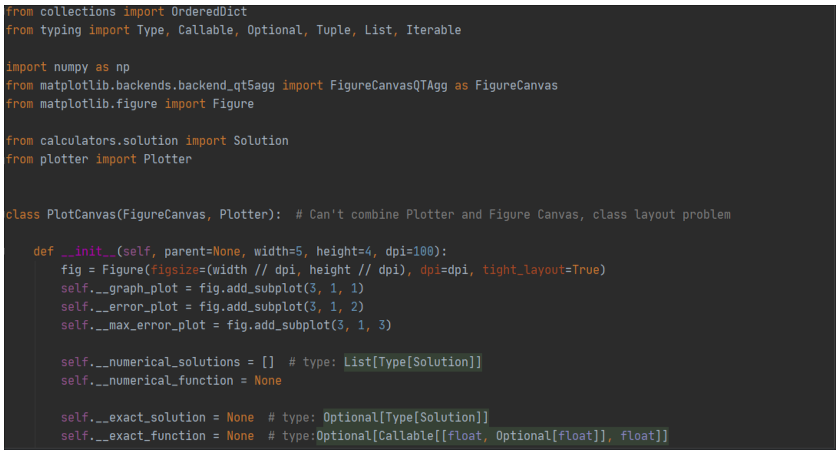
3.2 Improved Euler method:

As seen in figure 7, Improved Euler method performs much better then Euler, since it consider average slope of a function, that is definitely better for such exponential function. This method also have dramatic relative increase in local error (up to 52.032), but this time an error is not so big, and global error is just 100.376.

3.3 Runge-Kutta method:

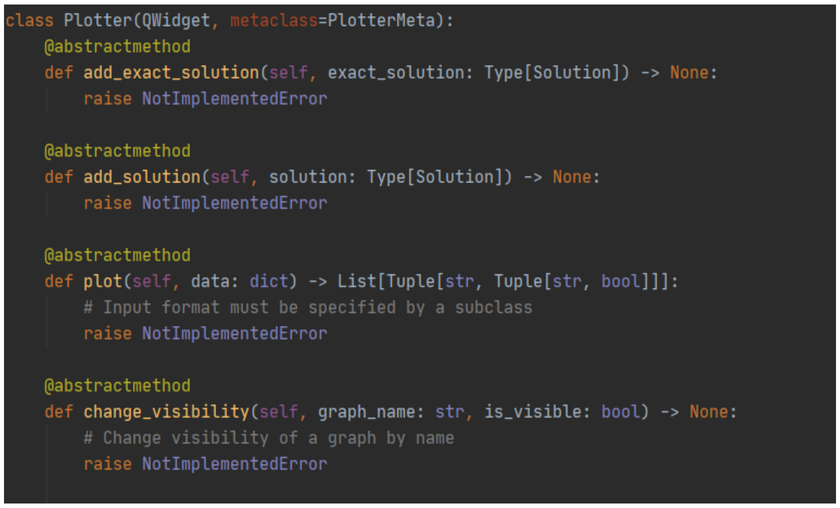
As seen in figure 8, The best approximation is obtained using Runge-Kutta method. This method, as well as others, loses accuracy after $x = 3.5$, but the approximation error is negligible (up to 0.031) with the total $error = 0.064$.

4 Code snippets:



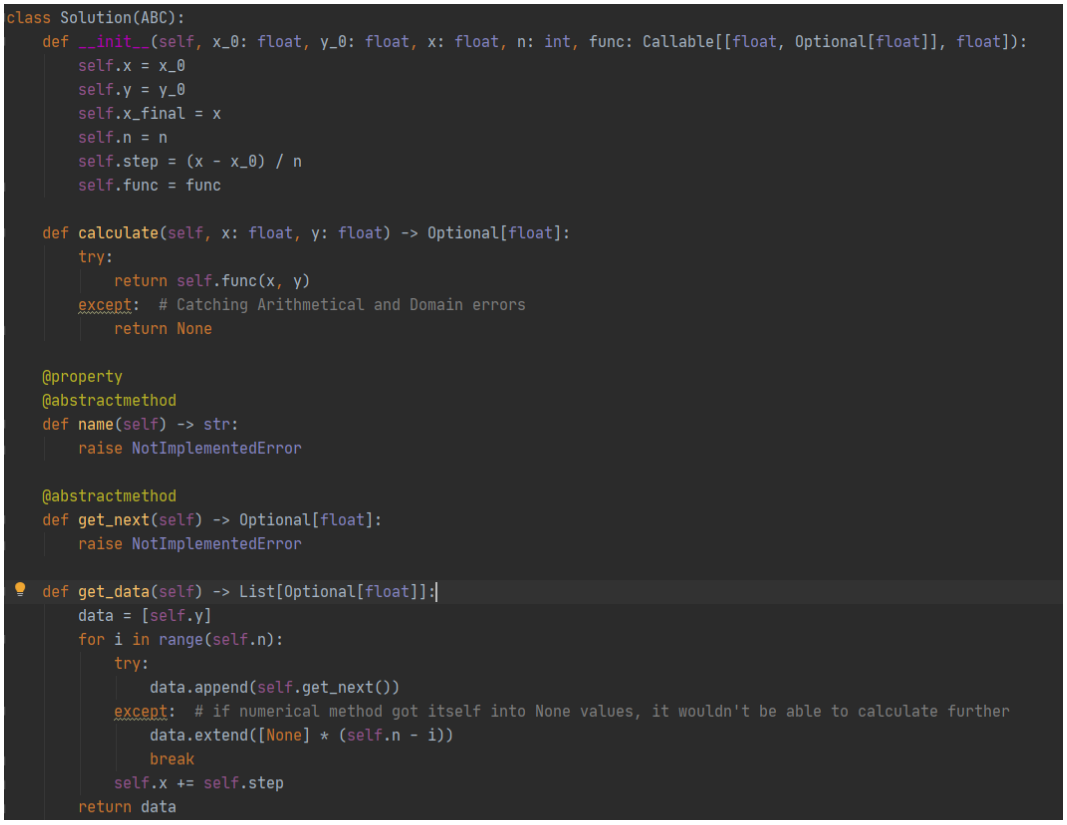
```python
from collections import OrderedDict
from typing import Type, Callable, Optional, Tuple, List, Iterable

import numpy as np
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure

from calculators.solution import Solution
from plotter import Plotter


class PlotCanvas(FigureCanvas, Plotter):  # Can't combine Plotter and Figure Canvas, class layout problem

    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width // dpi, height // dpi), dpi=dpi, tight_layout=True)
        self.__graph_plot = fig.add_subplot(3, 1, 1)
        self.__error_plot = fig.add_subplot(3, 1, 2)
        self.__max_error_plot = fig.add_subplot(3, 1, 3)

        self.__numerical_solutions = []  # type: List[Type[Solution]]
        self.__numerical_function = None

        self.__exact_solution = None  # type: Optional[Type[Solution]]
        self.__exact_function = None  # type:Optional[Callable[[float, Optional[float]], float]]
```

Figure 9: Implementation of PlotCanvas.



```python
class Plotter(QWidget, metaclass=PlotterMeta):
    @abstractmethod
    def add_exact_solution(self, exact_solution: Type[Solution]) -> None:
        raise NotImplementedError

    @abstractmethod
    def add_solution(self, solution: Type[Solution]) -> None:
        raise NotImplementedError

    @abstractmethod
    def plot(self, data: dict) -> List[Tuple[str, Tuple[str, bool]]]:
        # Input format must be specified by a subclass
        raise NotImplementedError

    @abstractmethod
    def change_visibility(self, graph_name: str, is_visible: bool) -> None:
        # Change visibility of a graph by name
        raise NotImplementedError
```

Figure 10: Implementation of Plotter.



```python
class Solution(ABC):
    def __init__(self, x_0: float, y_0: float, x: float, n: int, func: Callable[[float, Optional[float]], float]):
        self.x = x_0
        self.y = y_0
        self.x_final = x
        self.n = n
        self.step = (x - x_0) / n
        self.func = func

    def calculate(self, x: float, y: float) -> Optional[float]:
        try:
            return self.func(x, y)
        except:  # Catching Arithmetical and Domain errors
            return None

    @property
    @abstractmethod
    def name(self) -> str:
        raise NotImplementedError

    @abstractmethod
    def get_next(self) -> Optional[float]:
        raise NotImplementedError

    def get_data(self) -> List[Optional[float]]:
        data = [self.y]
        for i in range(self.n):
            try:
                data.append(self.get_next())
            except:  # if numerical method got itself into None values, it wouldn't be able to calculate further
                data.extend([None] * (self.n - i))
                break
            self.x += self.step
        return data
```

Figure 11: Implementation of all methods solutions using inheritance.

5 Conclusion:

As a result, imperially has been proven that for the given function and continuous functions from different variants Runge-Kutta method performs much better that others achieving relatively small error even on functions that grows exponentially fast.

6 GitHub:

All the materials used can be found on my GitHub.