Differential Equations programming assignment

Ramil Askarov, BS2-8
Fall18, Innopolis University
Link to GitHub project: https://github.com/Rome314/DE_Assigment

Initial Value problem

Problem:

$$\begin{cases} y' = (1 - 2y)e^2 + y^2 + e^{2x} \\ y(-5) = 2 \end{cases}$$

$x \in [-5:0]$

Exact solution for IVP is:

$$\frac{1 - 2e^5 - e^x(5+x) + e^{5+x}(9+2x)}{-5 - x + e^5(9+2x)}$$

Code explanation:

main.py:

Just for running and change some constants, such as steps count(Accuracy) or X(Bound for calculating interval)

```
from DE_Assigment.ploting import *

# Costs:
# ______________________________
steps_count = 1000
X = 0

glob_err_start = 10
glob_err_end = 100
# ______________________________


# Plotting graphs for each task:
plot_methods(steps_count, X)
plot_local_error(steps_count, X)
plot_global_error(glob_err_start, glob_err_end, X)
```

functions.py:
Here is main formulas for function and numeric method. If will be necessary to change function, need simply change formula in func **f** and **exact.**

```
from math import exp


# Original function and it's exact solution:
def f(x, y):
    return (1 - 2 * y) * exp(x) + y * y + exp(2 * x)


# assypmtot = (5 - 9 * exp(5)) / (2 * exp(5) - 1)


def exact(x):
    return (-exp(x) * (x + 5) + exp(x + 5) * (2 * x + 9) - (2 * exp(5)) + 1) / (-x + exp(5) * (2 * x + 9) - 5)


# Helping functions for Runge-Kutta method:

def k1(x, y):
    return f(x, y)


def k2(x, y, h):
    return f(x + h / 2, y + h * k1(x, y) / 2)


def k3(x, y, h):
    return f(x + h / 2, y + h * k2(x, y, h) / 2)


def k4(x, y, h):
    return f(x + h, y + h * k3(x, y, h))


def rk_delta(x, y, h):
    return h / 6 * (k1(x, y) + 2 * k2(x, y, h) + 2 * k3(x, y, h) + k4(x, y, h))


# Helping function for improved Euler method:

def imp_Euler_delta(x, y, h):
    return h * f(x + h / 2, y + h / 2 * f(x, y))
```

functions_calculator.py:

Module for calculating y values for each method. If function is changed and initial values too, have to change values in each def beginning, for example:
x = [-5.0] => x = [1.0]
y = [2] => y = [0]

```
from DE_Assigment.functions import *

# Here functions to find x's and y's for each function in range [X0:X]
# Where:
#    X0 = -5
#    Y0 = 2
#    X = final_x(You can change it in main function)

def euler_solution(count_steps, final_x):
    # Initial values
    x = [-5.0]
    y = [2]
    # step size
    h = (final_x - x[0]) / count_steps

    for i in range(count_steps):
        # here graph is created point by point
        x.append(x[i] + h)
        y.append(y[i] + h * (f(x[i], y[i])))

    return x,y


def exact_solution(steps_count, final_x):
    x = [-5.0]  # x0
    y = [2.0]  # y0

    h = (final_x - x[0]) / steps_count

    for i in range(steps_count):
        x.append(x[i] + h)
        y.append(exact(x[i]))

    return x, y


def runge_kutta_solution(steps_count, final_x):
    x = [-5.0]  # x0
    y = [2.0]  # y0

    h = (final_x - x[0]) / steps_count

    for i in range(steps_count):
        x.append(x[i] + h)
        y.append(y[i] + rk_delta(x[i], y[i], h))
    return x, y


def improved_euler_solution(steps_count, final_x):
    x = [-5.0]  # x0
    y = [2.0]  # y0

    h = (final_x - x[0]) / steps_count

    for i in range(steps_count):
        x.append(x[i] + h)
        y.append(y[i] + imp_Euler_delta(x[i], y[i], h))
    return x, y
```

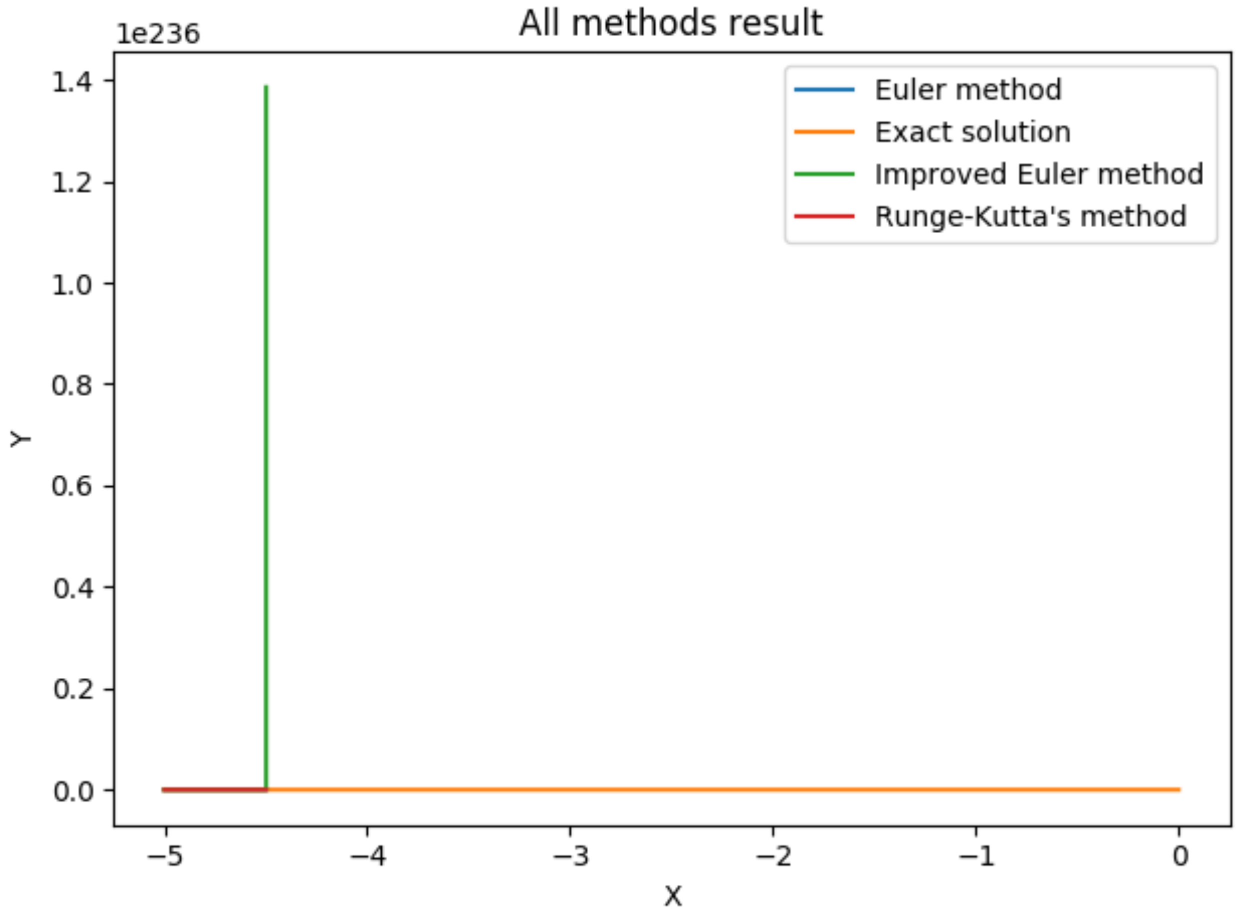
ploting.py:

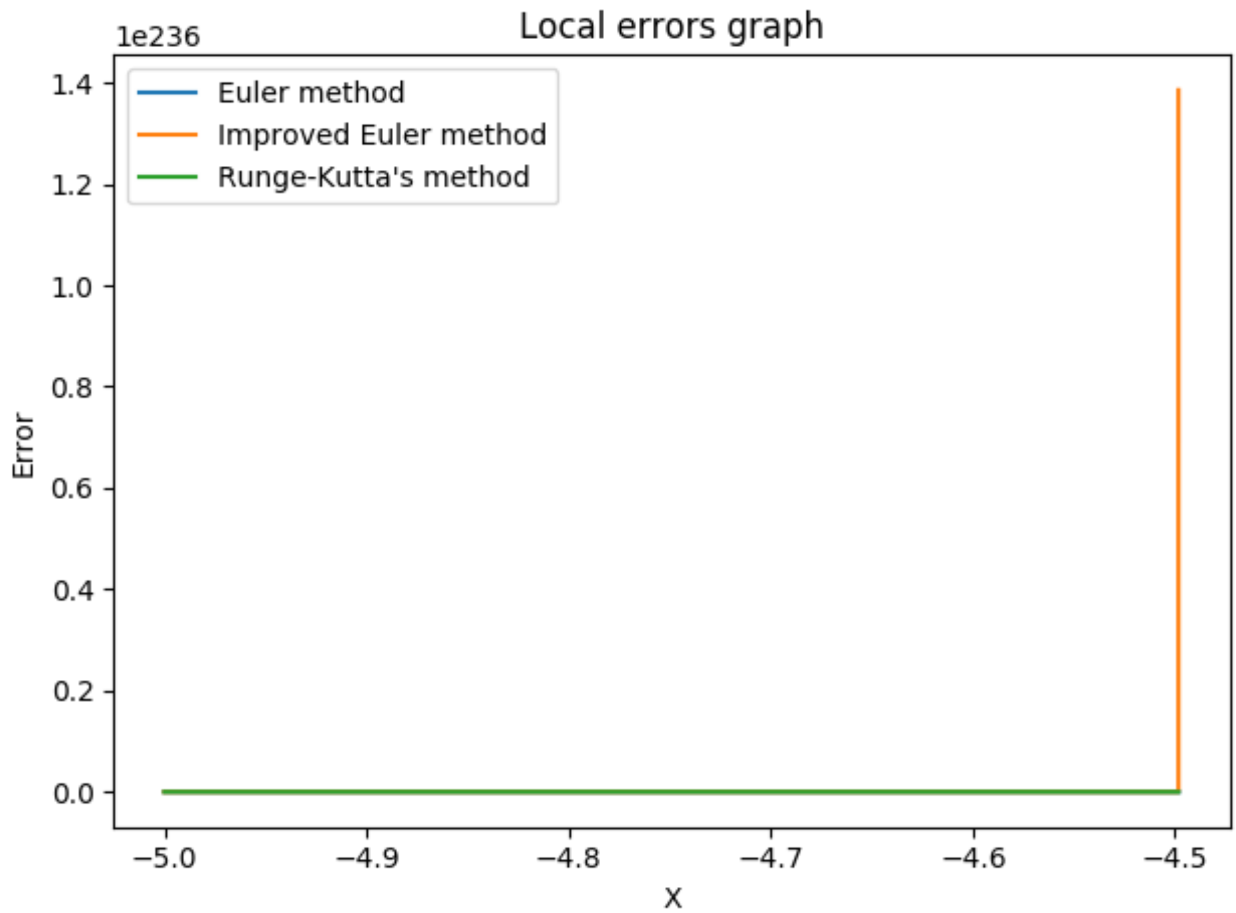
Module where graphs are creating

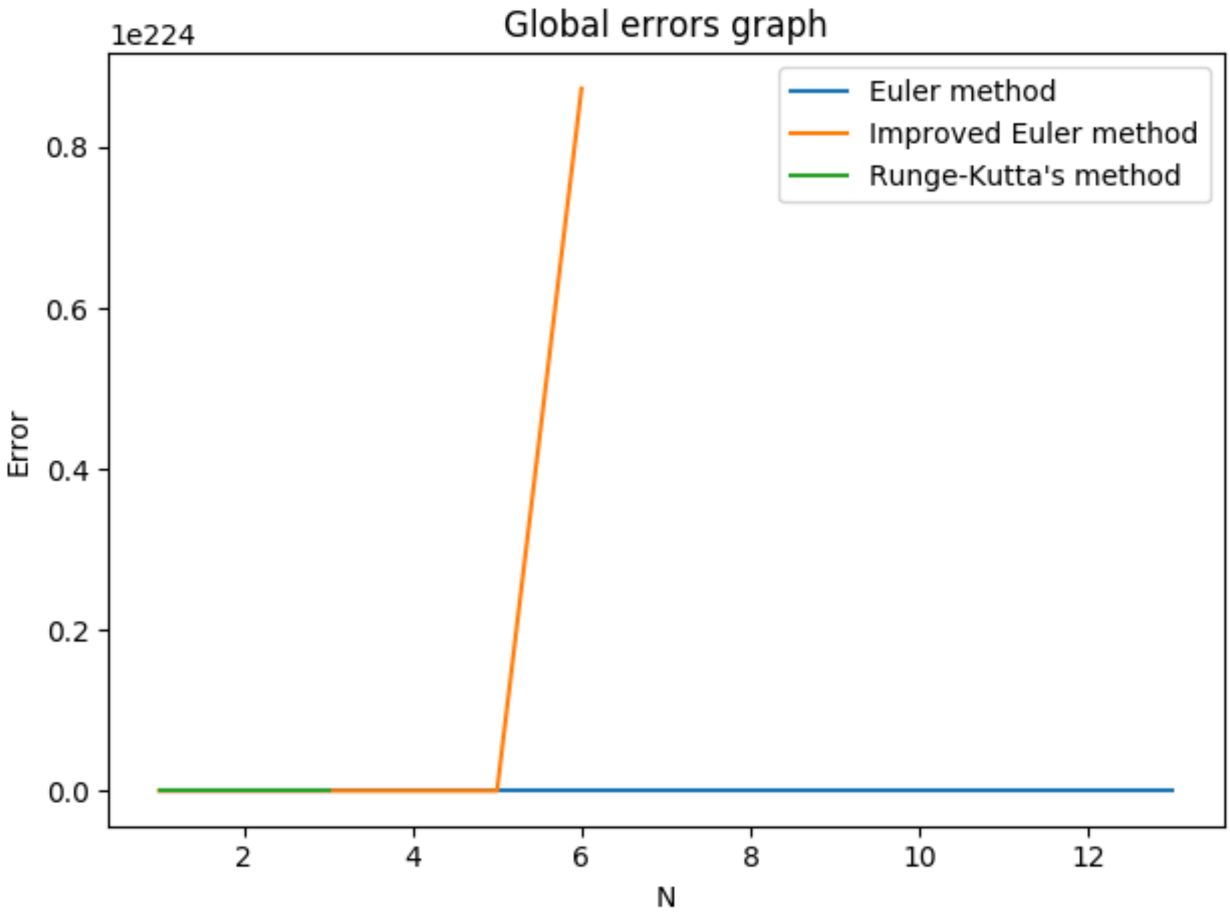
```python
from DE_Assigment.functions_calculator import *
from matplotlib import pyplot as plt
from math import fabs

#Plotting usual graphs for each ,ethod:
#   @steps_count - accuracy value
#   @x_final - end of plotting interval
def plot_methods(steps_count, x_final):
    x_euler, y_euler = euler_solution(steps_count, x_final)
    x_exact, y_exact = exact_solution(steps_count, x_final)
    x_euler_improved, y_euler_improved = improved_euler_solution(steps_count, x_final)
    x_runge_kutta, y_runge_kutta = runge_kutta_solution(steps_count, x_final)

    plt.title("All methods result")
    plt.plot(x_euler, y_euler, label="Euler method")
    plt.plot(x_exact, y_exact, label="Exact solution")
    plt.plot(x_euler_improved, y_euler_improved, label="Improved Euler method")
    plt.plot(x_runge_kutta, y_runge_kutta, label="Runge-Kutta's method")
    plt.ylabel("Y")
    plt.xlabel("X")
    plt.legend()
    plt.show()

# Plotting local errors graphs for each method:
#   @steps_count - accuracy value
#   @x_final - end of plotting interval
def plot_local_error(steps_count, x_final):
    x_exact, y_exact = exact_solution(steps_count, x_final)
    x_euler_improved, y_euler_improved = improved_euler_solution(steps_count, x_final)
    x_euler, y_euler = euler_solution(steps_count, x_final)
    x_runge_kutta, y_runge_kutta = runge_kutta_solution(steps_count, x_final)

    runge_kutta_error = [0.0]
    euler_imp_error = [0.0]
    euler_error = [0.0]

    for i in range(steps_count):
        runge_kutta_error.append(fabs(y_exact[i] - y_runge_kutta[i]))
        euler_imp_error.append(fabs(y_exact[i] - y_euler_improved[i]))
        euler_error.append(fabs(y_exact[i] - y_euler[i]))

    plt.title("Local errors graph")
    plt.plot(x_euler, euler_error, label="Euler method")
    plt.plot(x_euler_improved, euler_imp_error, label="Improved Euler method")
    plt.plot(x_runge_kutta, runge_kutta_error, label="Runge-Kutta's method")
    plt.ylabel("Error")
    plt.xlabel("X")
    plt.legend()
    plt.show()
```

```python
# Plotting global errors graphs for each method:
#   @start,end - accuracy values
#   @X - end of plotting interval

def plot_global_error(start, end, X):
    arr = [i for i in range(start, end)]
    euler_glob_err = []
    euler_imp_glob_err = []
    runge_kutta_glob_err = []
    for i in arr:
        x_euler, y_euler = euler_solution(i, X)
        x_exact, y_exact = exact_solution(i, X)
        x_euler_improved, y_euler_improved = improved_euler_solution(i, X)
        x_runge_kutta, y_runge_kutta = runge_kutta_solution(i, X)

        runge_kutta_error = 0
        euler_imp_error = 0
        euler_error = 0
        for k in range(i):
            runge_kutta_error = max((fabs(y_exact[k] - y_runge_kutta[k])), runge_kutta_error)
            euler_imp_error = max((fabs(y_exact[k] - y_euler_improved[k])), euler_imp_error)
            euler_error = max((y_exact[k] - y_euler[k]), euler_error)

        euler_glob_err.append(euler_error)
        euler_imp_glob_err.append(euler_imp_error)
        runge_kutta_glob_err.append(runge_kutta_error)

    plt.title("Global errors graph")
    plt.plot(arr, euler_glob_err, label="Euler method")
    plt.plot(arr, euler_imp_glob_err, label="Improved Euler method")
    plt.plot(arr, runge_kutta_glob_err, label="Runge-Kutta's method")
    plt.ylabel("Error")
    plt.xlabel("N")
    plt.legend()
    plt.show()
```

Program running results:
With accuracy = 1000000






Global errors graph
1e224
Euler method
Improved Euler method
Runge-Kutta's method
0.8
0.6
0.4
0.2
0.0
Error
2
4
6
8
10
12
N