Differential Equations Programming Assignment

Kuleykin Vladislav B16-02
Variant 11

1 General information

1.1 Given equation

$$y' = e^{-\sin(x)} - ycos(x); \quad x_0 = 0, \ y_0 = 1, \ x_{max} = 9.3$$

So I was lucky because this equation does not have any breakpoints and continuous on all interval.

1.2 Implementation

I implemented all in Python using next libraries:

NumPy – computation (sin(), cos(), etc.)

PyQtGraph – visualization of graphs

PyQt5 – UI of the program.

1.3 Limits of input

Logic - only real numbers (except the field 'Number of steps'), x_{max}>x_0 and Number of steps in greater than or equal to 1

Computational limitations - Number of steps is limited by 1000 then computing Max error function and limited by 100000 otherwise due the time and memory consumption.

1.4 Graphs

All graphs shown in this document have this input:

$y_0 = 1$
$x_0 = 0$
$Number\ of\ steps = 100$
$x_{max} = 9.3$

1.5 Code

The full code can be found at https://github.com/kzvdar42/DE-Programming-Assignment.

To launch the program, follow the instructions from GitHub.

2 Exact solution

2.1 Solution

$$y' + ycos(x) = e^{-\sin(x)}$$
$$y' + ycos(x) = 0$$
$$y' = -ycos(x)$$
$$\frac{dy}{y} = -\cos(x)\,dx$$
$$\int \frac{dy}{y} = \int -\cos(x)\,dx$$
$$y = C(x)e^{-\sin(x)}$$
$$y' = C'(x)e^{-\sin(x)} - C(x)\cos(x)\,e^{-\sin(x)}$$
$$C'(x)e^{-\sin(x)} - C(x)\cos(x)\,e^{-\sin(x)} + C(x)e^{-\sin(x)}\cos(x) = e^{-\sin(x)}$$
$$C'(x) = 1$$
$$C(x) = x + C$$
$$y = (x + C)e^{-\sin(x)}$$

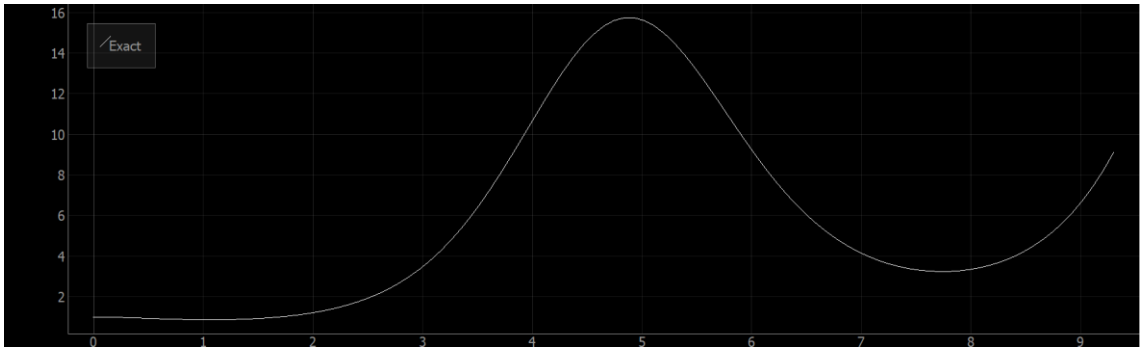
Therefore, to solve this equation for different IVP we need to calculate the constant:

$$C = \frac{y_0}{e^{-\sin(x)}} - x_0$$

Small comment: $e^{-\sin(x)}$ is never a zero, so we do not have breakpoints

2.2 Graphs

2.2.1 Solution



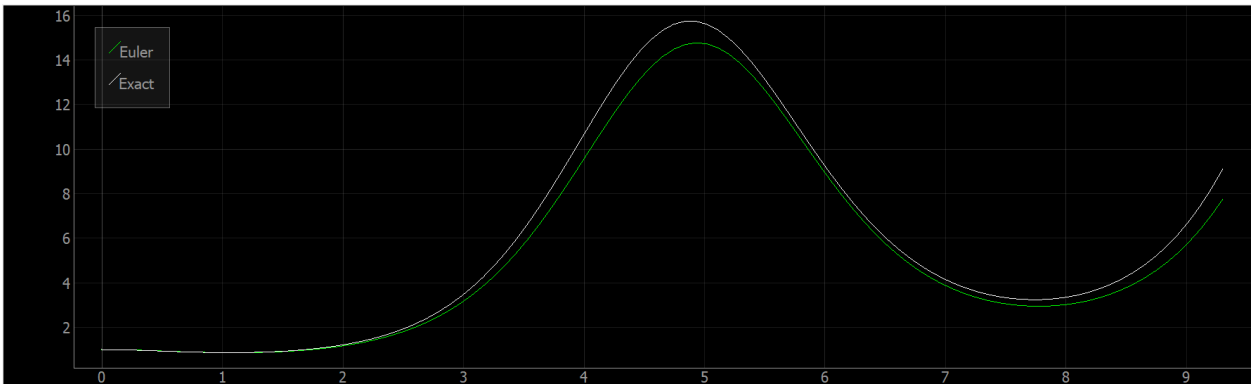
2.3 In code

```
def exact_sol(y0, x):
    """
    Calculates the exact solution of function given in task (Variant 11)
    :param y0: initial y
    :param x: x values
    :return: y' values
    """
    c = y0 / np.exp(-np.sin(x[0])) - x[0]
    y = (x + c) * np.exp(-np.sin(x))
    return y
```

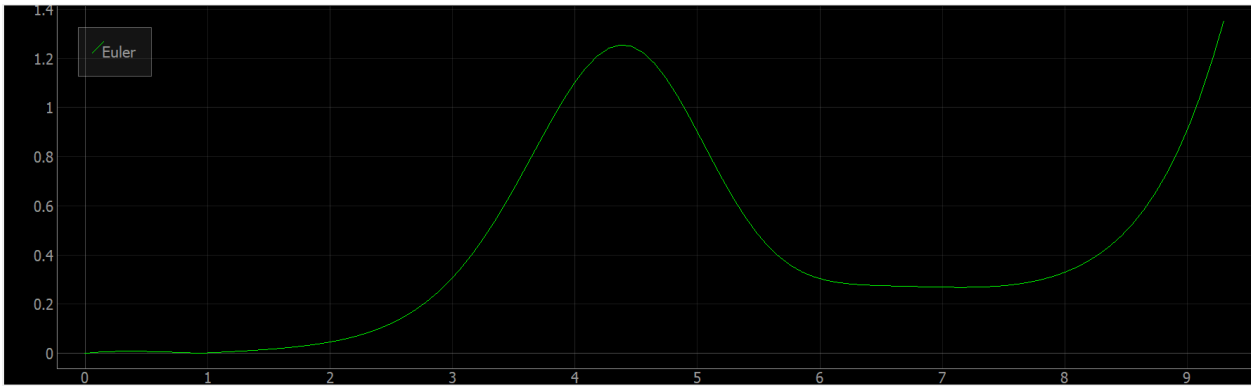
3 Euler method

3.1 Graphs

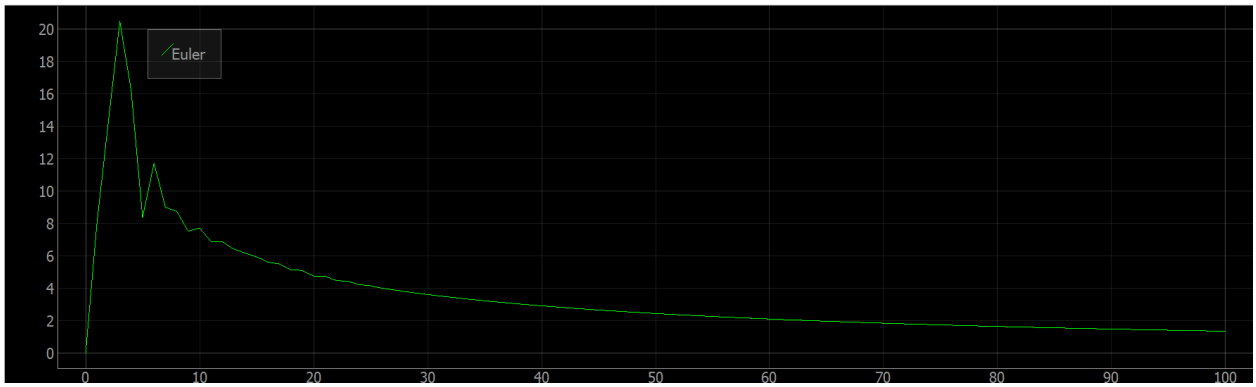
3.1.1 Solution



3.1.2 Error



3.1.3 Max Error



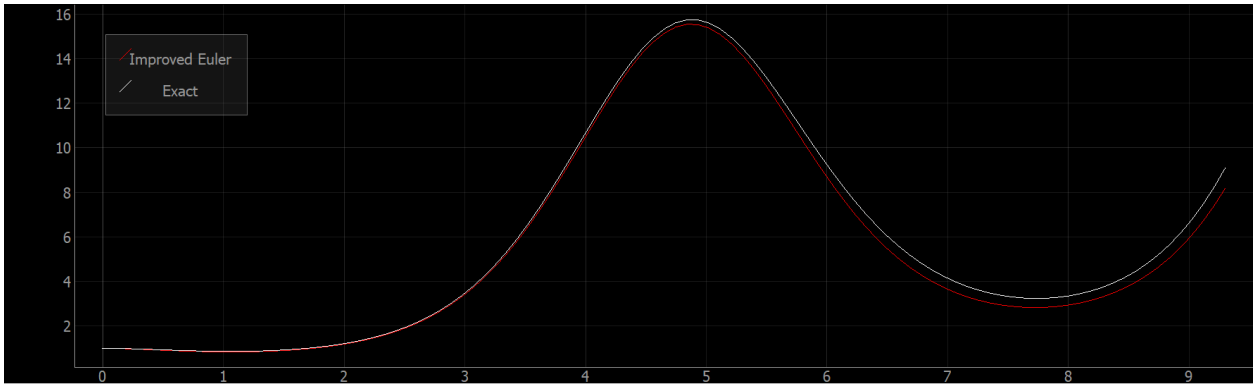
3.2 In code

```
def euler(y0, x, step):
    """
        Calculates the euler algorithm for given values
        :param y0: initial y
        :param x: all x values
        :param step: step of x
        :return: calculated y values
        """
    y = [y0]
    for i in range(1, len(x)):
        yi = y[i - 1] + step * f(x[i - 1], y[i - 1])
        y.append(yi)
    return y
```

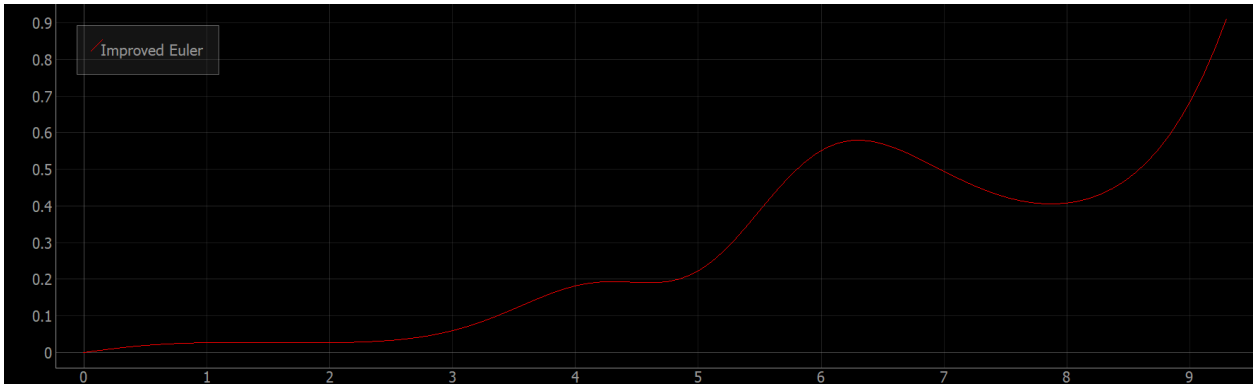
4 Improved Euler method

4.1 Graphs

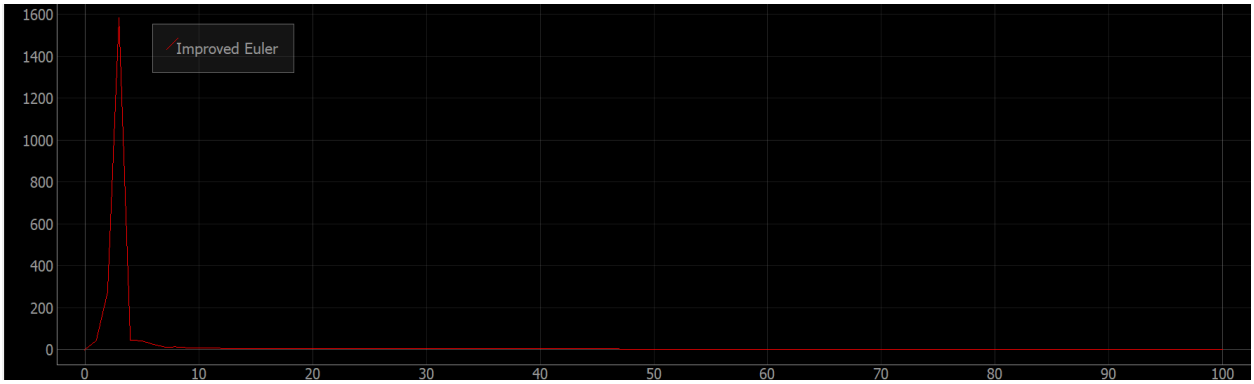
4.1.1 Solution



4.1.2 Errors



4.1.3 Max error



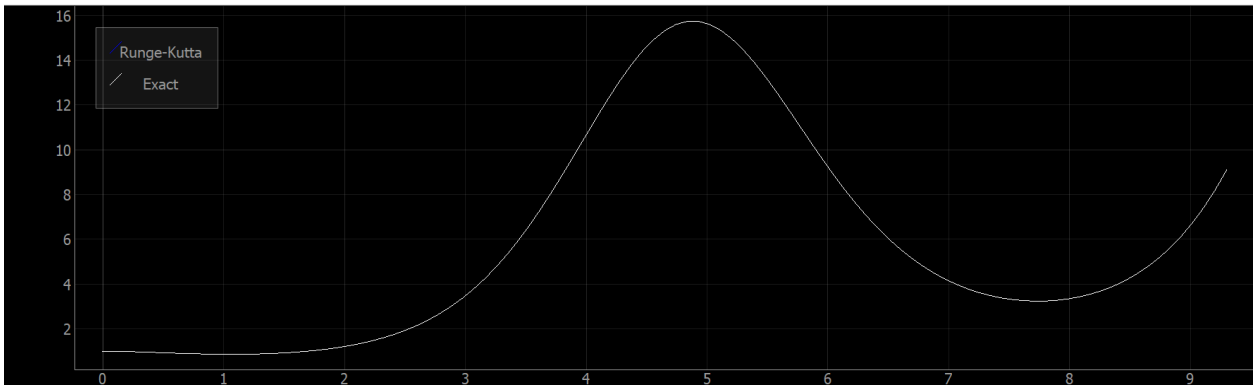
4.2 In code

```
def impr_euler(y0, x, step):
    """
    Calculates the improved euler algorithm for given values
    :param y0: initial y
    :param x: all x values
    :param step: step of x
    :return: calculated y values
    """
    y = [y0]
    for i in range(1, len(x)):
        k1 = y[i - 1] + step * f(x[i - 1], y[i - 1])
        yi = y[i - 1] + step * (f(x[i - 1], y[i - 1]) + f(x[i - 1] + step, y[i - 1]
+ step * k1)) / 2.0
        y.append(yi)
    return y
```

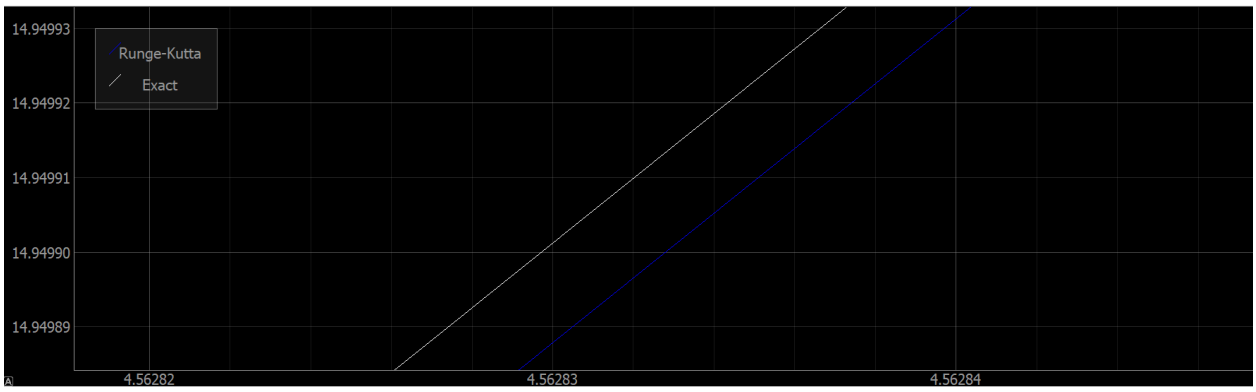
5 Runge-Kutta method

5.1 Graphs

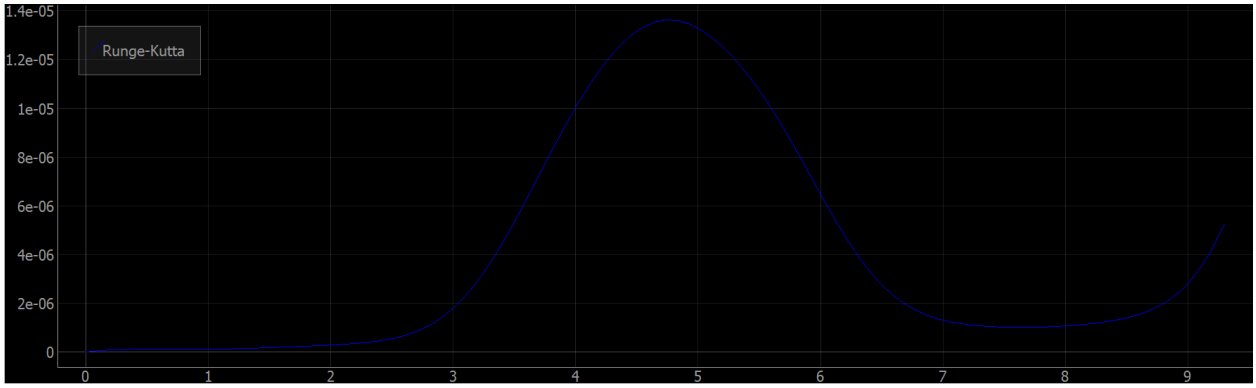
5.1.1 Solution



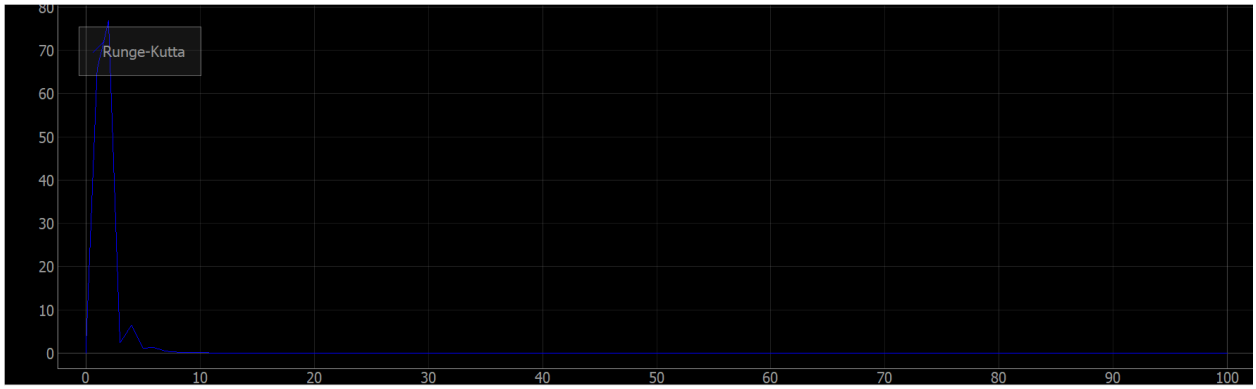
So, as you can see the Runge-Kutta graphs just merges with the exact solution, so here is a closer look:



5.1.2 Error



5.1.3 Max errors



5.2 In code

```
def runge_kutt(y0, x, step):
    """
    Calculates the runge_kutt algorithm for given values
    :param y0: initial y
    :param x: all x values
    :param step: step of x
    :return: calculated y values
    """
    y = [y0]
    for i in range(1, len(x)):
        k1 = f(x[i - 1], y[i - 1])
        k2 = f(x[i - 1] + step / 2, y[i - 1] + step * k1 / 2)
        k3 = f(x[i - 1] + step / 2, y[i - 1] + step * k2 / 2)
        k4 = f(x[i - 1] + step, y[i - 1] + step * k3)
        yi = y[i - 1] + step / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
        y.append(yi)
    return y
```