# Mullanurov Almir

## Differential equations programming assignment

Variant 2:

| | | | |
|---|---|---|---|
| $-2y + 4x$ | 0 | 0 | 3 |

https://github.com/Mirlan-code/DE_programming

# Exact solution:

$y' = -2y + 4x,\ y_0 = 0,\ x_0 = 0,\ X = 3$

$y' + 2y = 4x - $ *linear nonhomogeneous* 1*st order equation*

$p(x) = 2,\ q(x) = 4x$

*let's substitute* $y = uy_1$, *where* $y_1 -$ *the partial solution for*

$y' + 2y = 0\,.$

$y_1'/y_1 = -2$

$ln(y_1) = -2x \Rightarrow y_1 = e^{(-2x)}$

$u' = q(x)/y_1(x) = 4x/e^{(-2x)}$

$u = \int 4xdx/e^{(-2x)} = e^{2x}(2x-1) + C$

$y = uy_1 \Rightarrow y = (e^{2x}(2x-1) + C)e^{(-2x)} = Ce^{(-2x)} + 2x - 1$

$y_0 = Ce^{(-2x_0)} + 2x_0 - 1$

$C = (y_0 - 2x_0 + 1)/e^{(-2x_0)}$

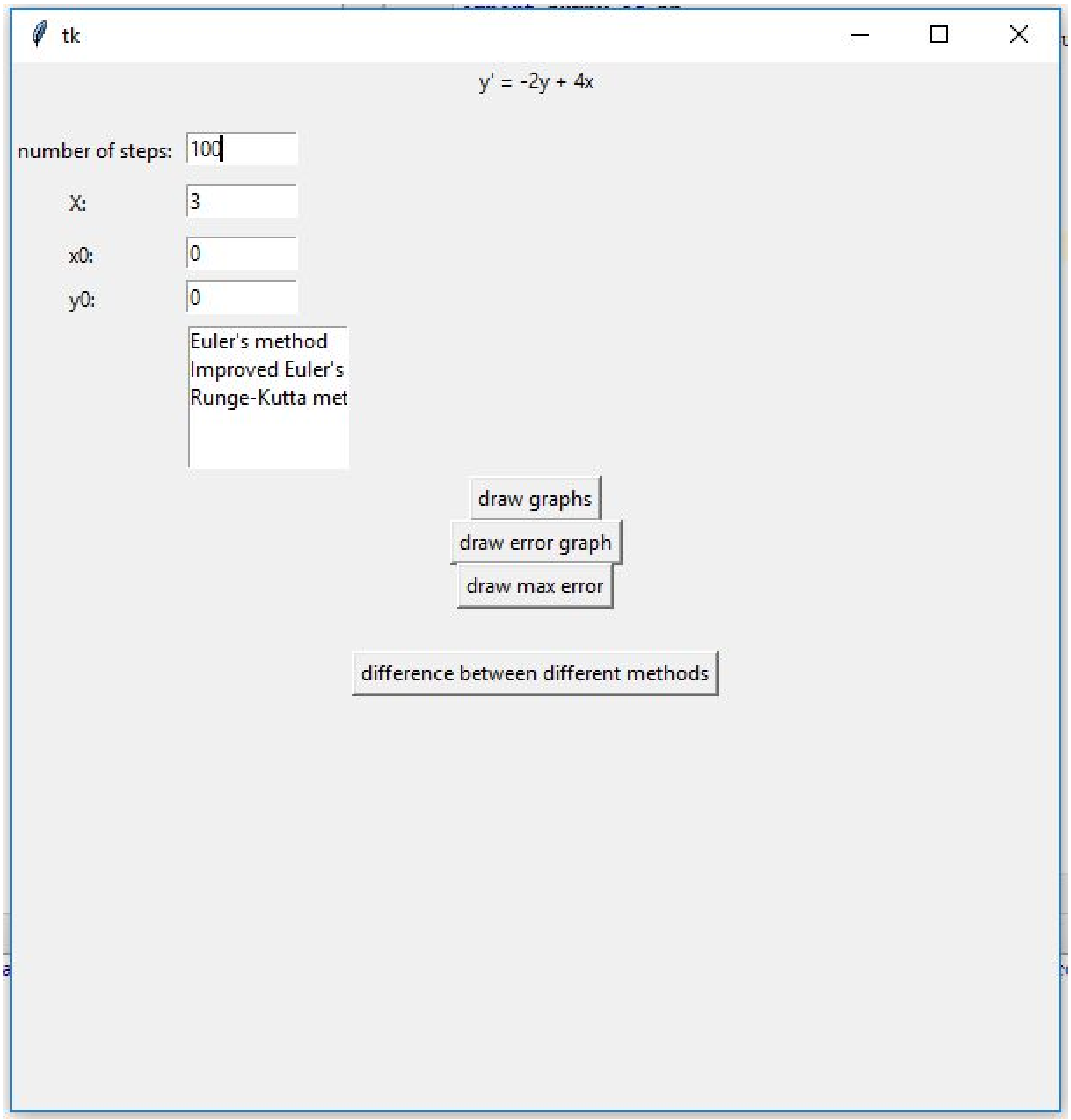
*In our case* : $C = (0 - 0 + 1)/e^0 = 1$

# Programming assignment:

Language: Python

Libraries: Matplotlib, numpy, tkinter

Application has the following view:




In application we can change number of steps (from 10 to 1000), initial values x0 and y0, and range X

In my variant x is from x0 to X, so if x0 is bigger than X, then I set x0 to 0

# Structure of the program:

exact_sol method will give us the exact solution of initial value problem. C is going to be determined using initial x and y

```
def exact_sol(X, opt):
    c = (opt.INITIAL_Y - opt.INITIAL_X * 2 + 1) / math.exp(-2 * opt.INITIAL_X)
    return c * np.exp(-2 * X) + 2 * X - 1
```

Function f returns us f(x,y)

```
def f(x, y):
    return -2 * y + 4 * x
```

approx_method returns us the name of the method and pair (x,y) to plot the graph

```
def approx_method(opt):  # returns two parameters:
```

These methods returns us the approximate solution using correspondent method. After each method we transform the array into numpy array for plotting

```
def Eulers(opt):

def ImprovedEulers(opt):

def RungeKutta(opt):
```

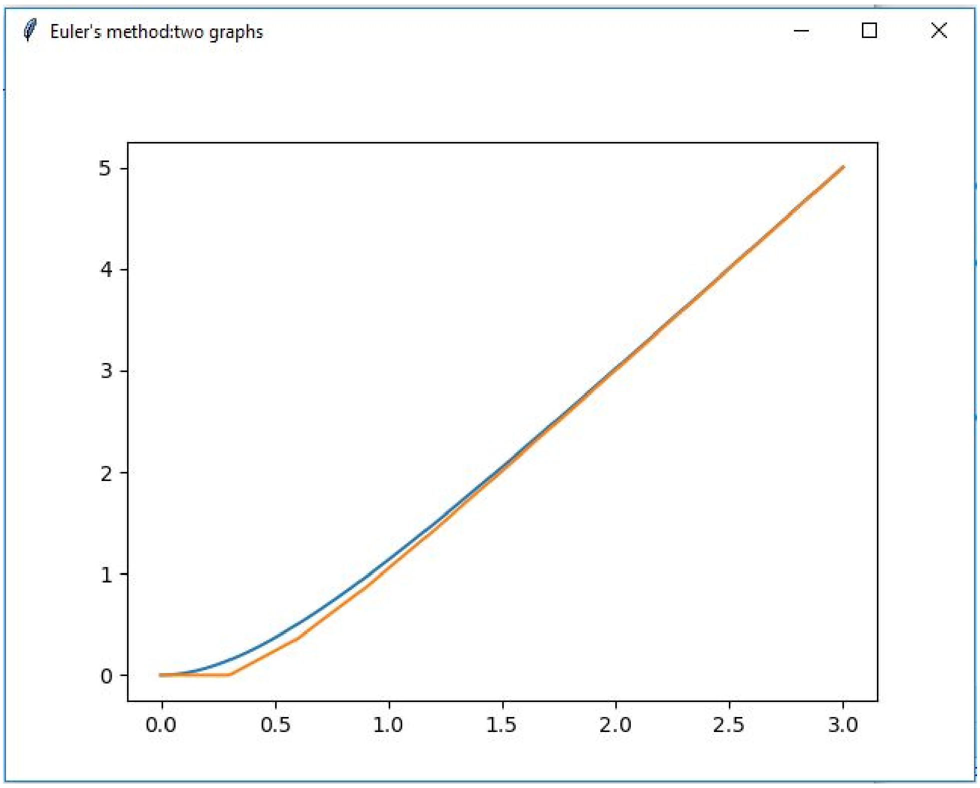
These methods draws the graph and error graphs

```
def draw_graph(opt):

def draw_error_graph(opt):

def draw_max_error_graph(opt):

def draw_all_error_graph(opt):
```

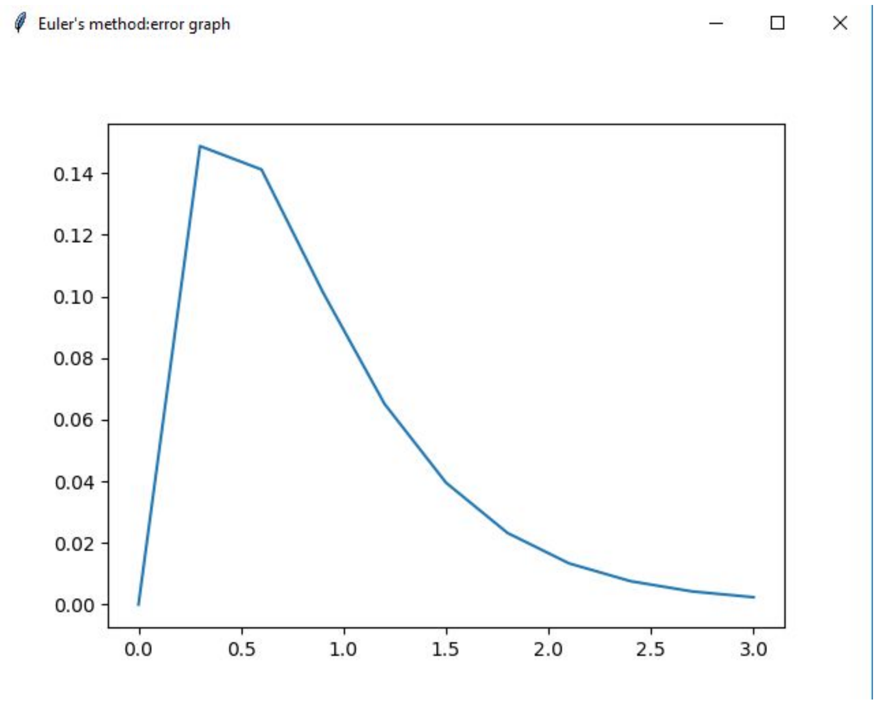
# Graphs:

$X = 3,\ x_0 = 0,\ y_0 = 0,$ n = 10 (it is so small to see the difference on the graphs, because the given range X is not big, the h is small)

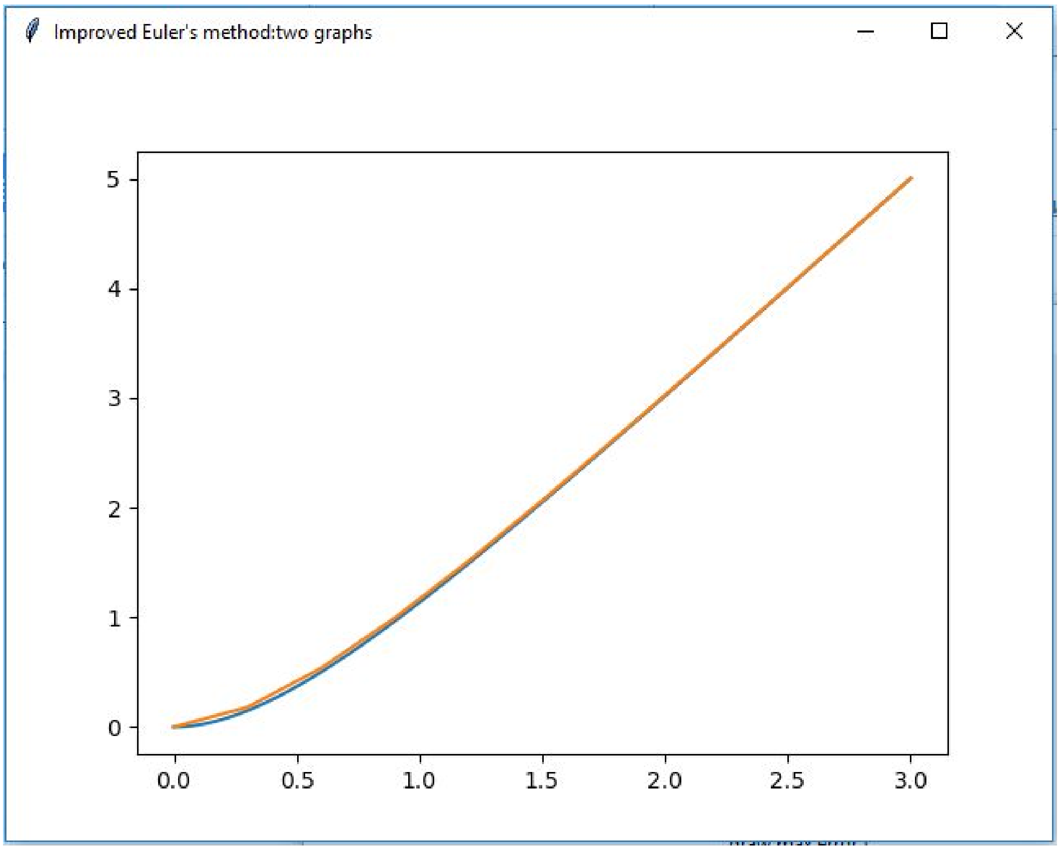
# Euler’s method:



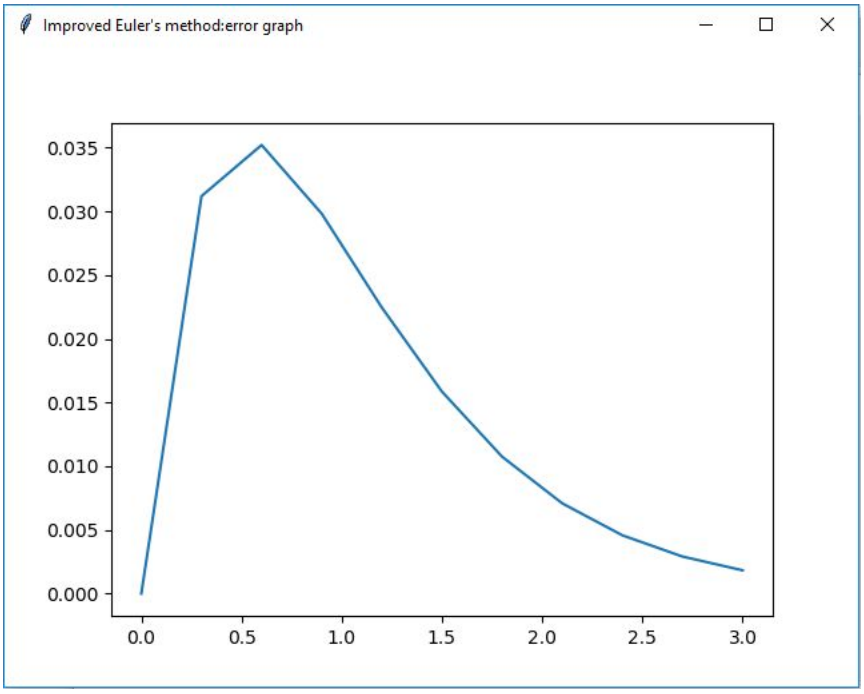

Blue - exact solution, red - Euler’s method



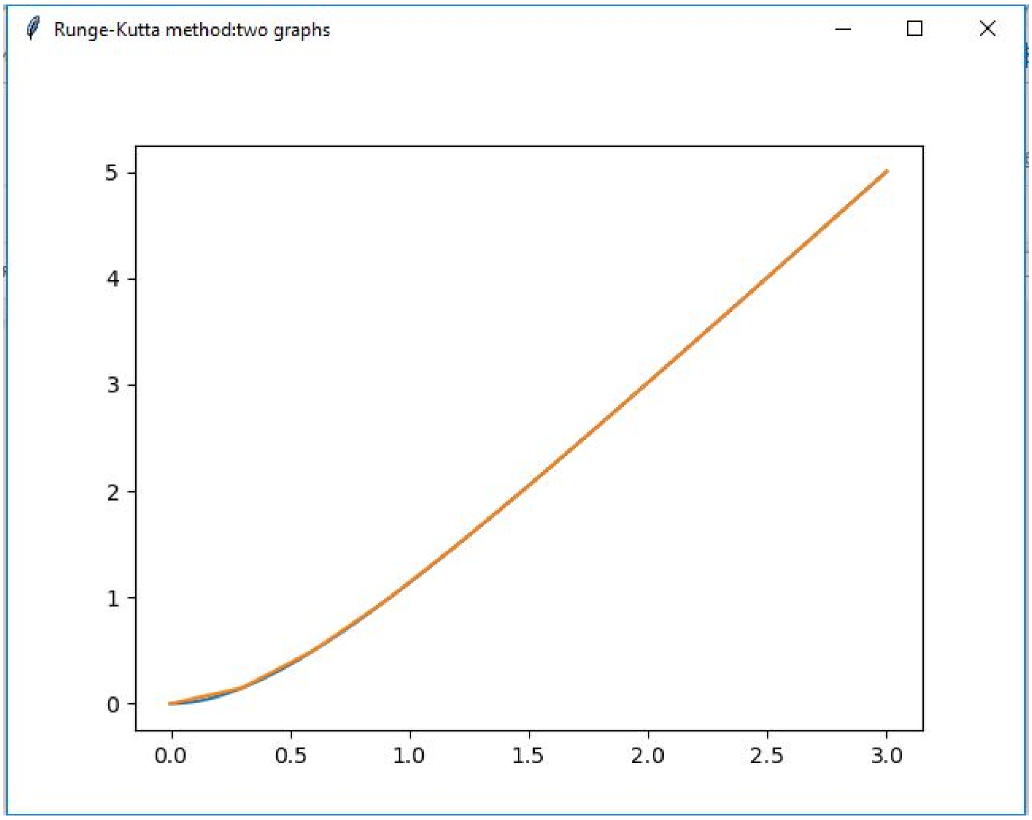
# Improved Euler’s method:



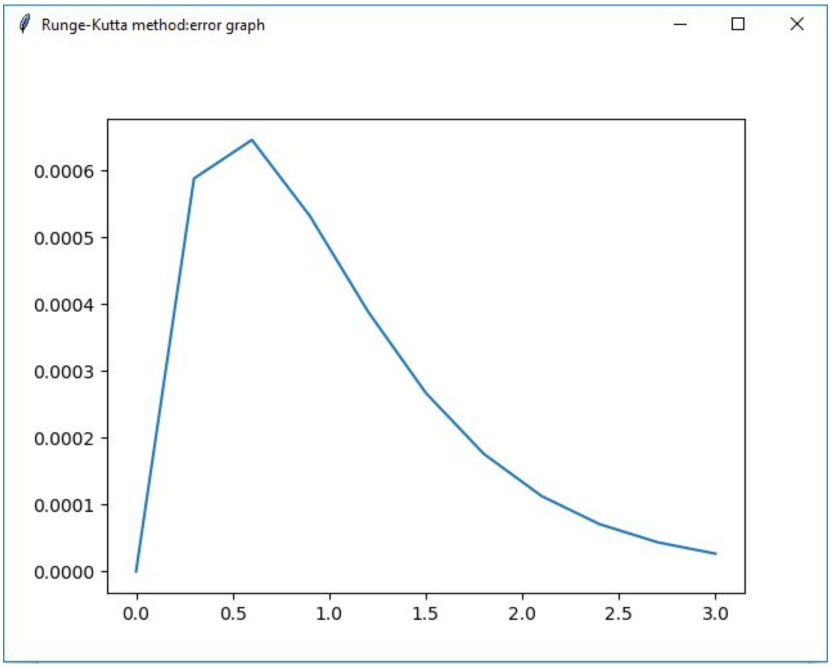

Blue - exact solution, red - Improved Euler’s method



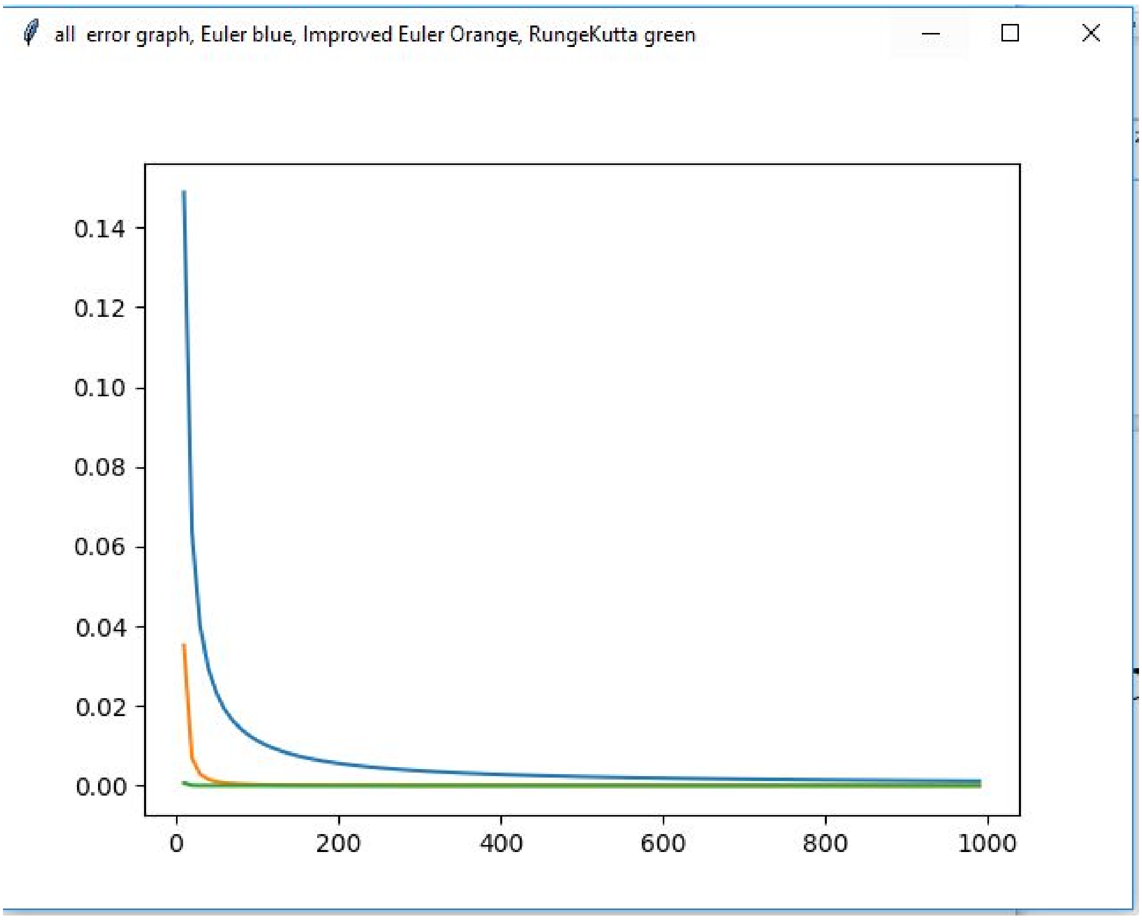
# Runge-Kutta method:




Blue - exact solution, red - Runge-Kutta method method



# The difference between different methods:




As we may see, the Runge-Kutta is the best one, the second one is the Improved Euler and the last one is Euler method