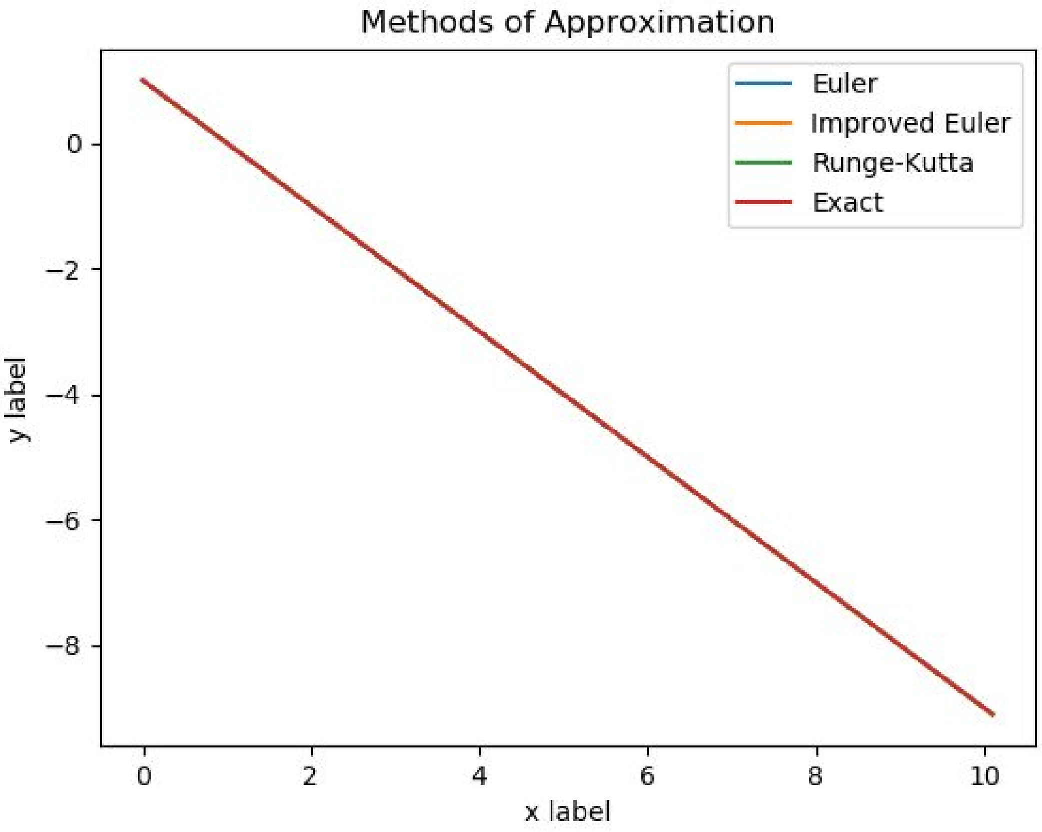
Differential Equations Report

Andriyanchenko Oleg

Variant #1

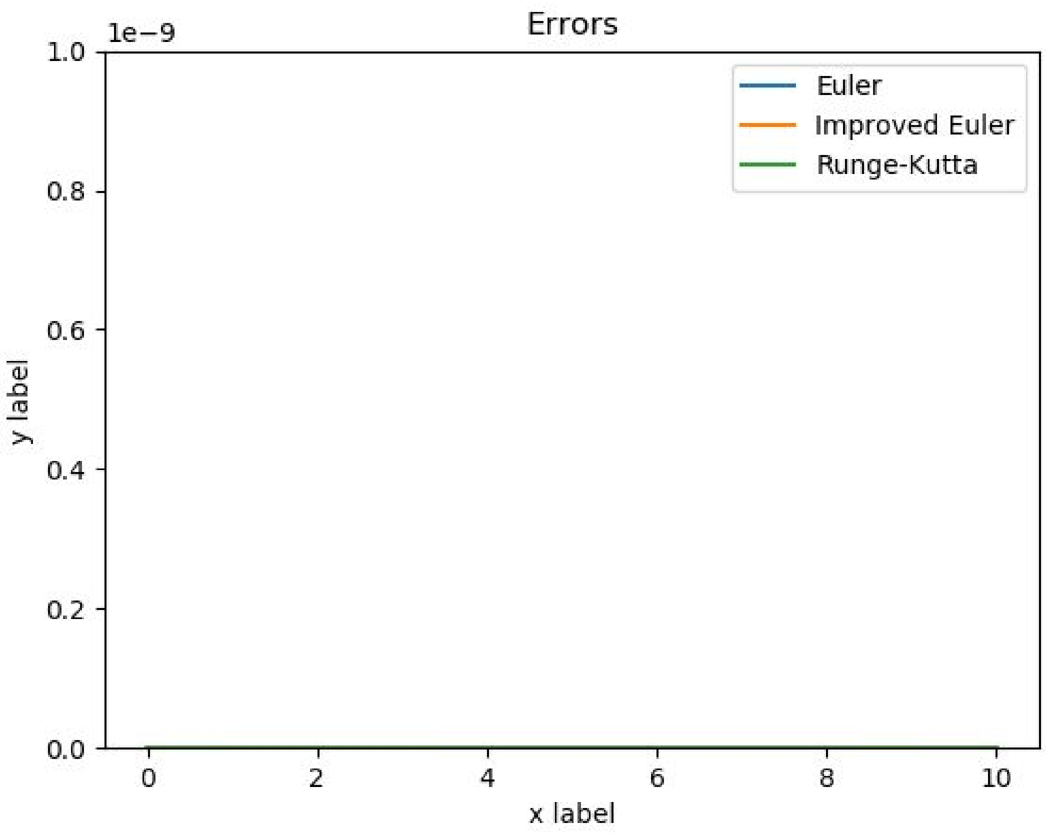
Given DE is $f(x, y) = - y - x$. Where $x_0 = 0$, $y_0 = 1$.
By solving this differential equation we get $y = 1 - x + ce^{-x}$. After substitution with initial values we get that c = 0. Therefore, exact equation is $y = 1 - x$. It is just a straight line.

Due to the simplicity of equation all approximation methods work excellent ():



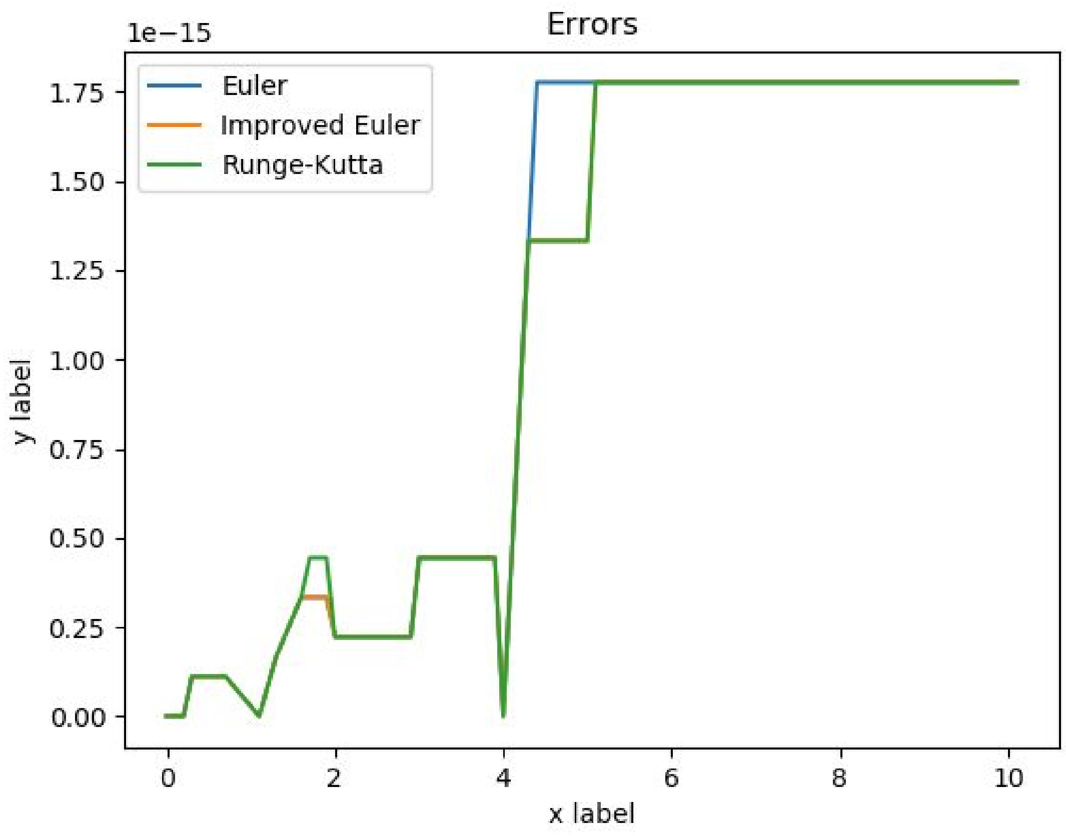
All methods give us roughly the same line. And because exact solution was plotted last it has only its color.

Because of the simplicity of the function global error is really small for all methods. We can just say it is quite “the same” (the step *h = 0.1*):

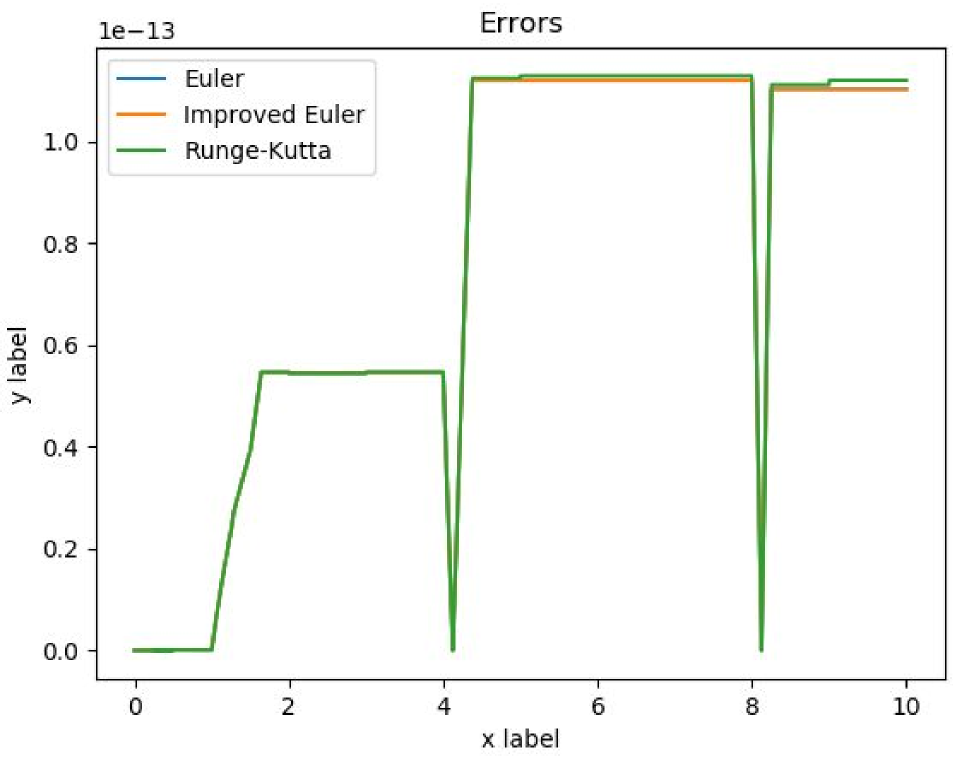


The green line at the bottom is the upper layer of error plot. This means that all 3 methods work well with these types of differential equations. Errors are so small that we could absolutely neglect them. Changing the step h doesn’t really affect the plot at that scaling.

If we scale up a little bit we can see that errors are a bit different (Note that scaling is 1 * 1e-15):

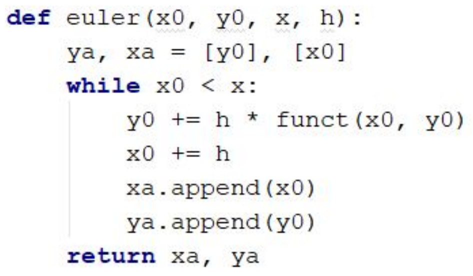


If we change the step h from 0.1 to 0.001 the plot will be a bit different. So the step does affect the global error (Note that scaling is 1 * 1e-13):

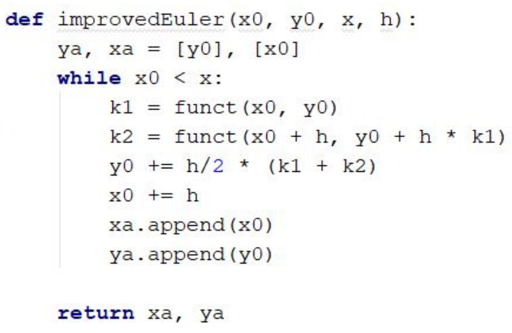


While implementing the methods I used Python 3.6, plotting library matplotlib and scientific library NumPy. I implemented all 3 methods of approximation independently according to the given textbook.

Features implementing methods of approximation:

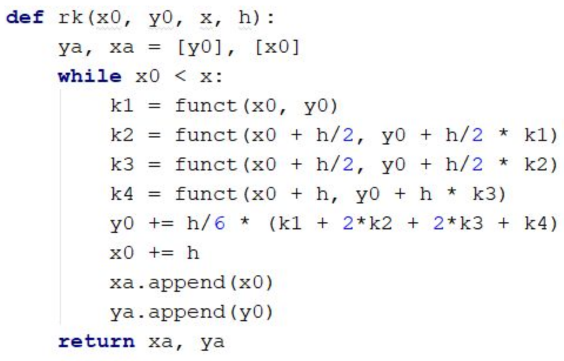


```python
def euler(x0, y0, x, h):
    ya, xa = [y0], [x0]
    while x0 < x:
        y0 += h * funct(x0, y0)
        x0 += h
        xa.append(x0)
        ya.append(y0)
    return xa, ya
```



```python
def improvedEuler(x0, y0, x, h):
    ya, xa = [y0], [x0]
    while x0 < x:
        k1 = funct(x0, y0)
        k2 = funct(x0 + h, y0 + h * k1)
        y0 += h/2 * (k1 + k2)
        x0 += h
        xa.append(x0)
        ya.append(y0)

    return xa, ya
```



```python
def rk(x0, y0, x, h):
    ya, xa = [y0], [x0]
    while x0 < x:
        k1 = funct(x0, y0)
        k2 = funct(x0 + h/2, y0 + h/2 * k1)
        k3 = funct(x0 + h/2, y0 + h/2 * k2)
        k4 = funct(x0 + h, y0 + h * k3)
        y0 += h/6 * (k1 + 2*k2 + 2*k3 + k4)
        x0 += h
        xa.append(x0)
        ya.append(y0)
    return xa, ya
```

To plot the exact solution I used the solution of given equation. By substitution with some *x* I get the *y*. These pairs of *x, y* were used to plot the graph.