Differential Equations Computational Practice report

Problem statement

Given differential equation: $y'(x) = \frac{1}{2}\sin(2x) - y(x)\cos(x)$

Initial Value problem:
x0 = 0,
y0 = 1,
X = 5.2
Task : using Euler’s method, Improved Euler’s method and Runge-Kutta method provide a solution and analyze errors

Exact solution

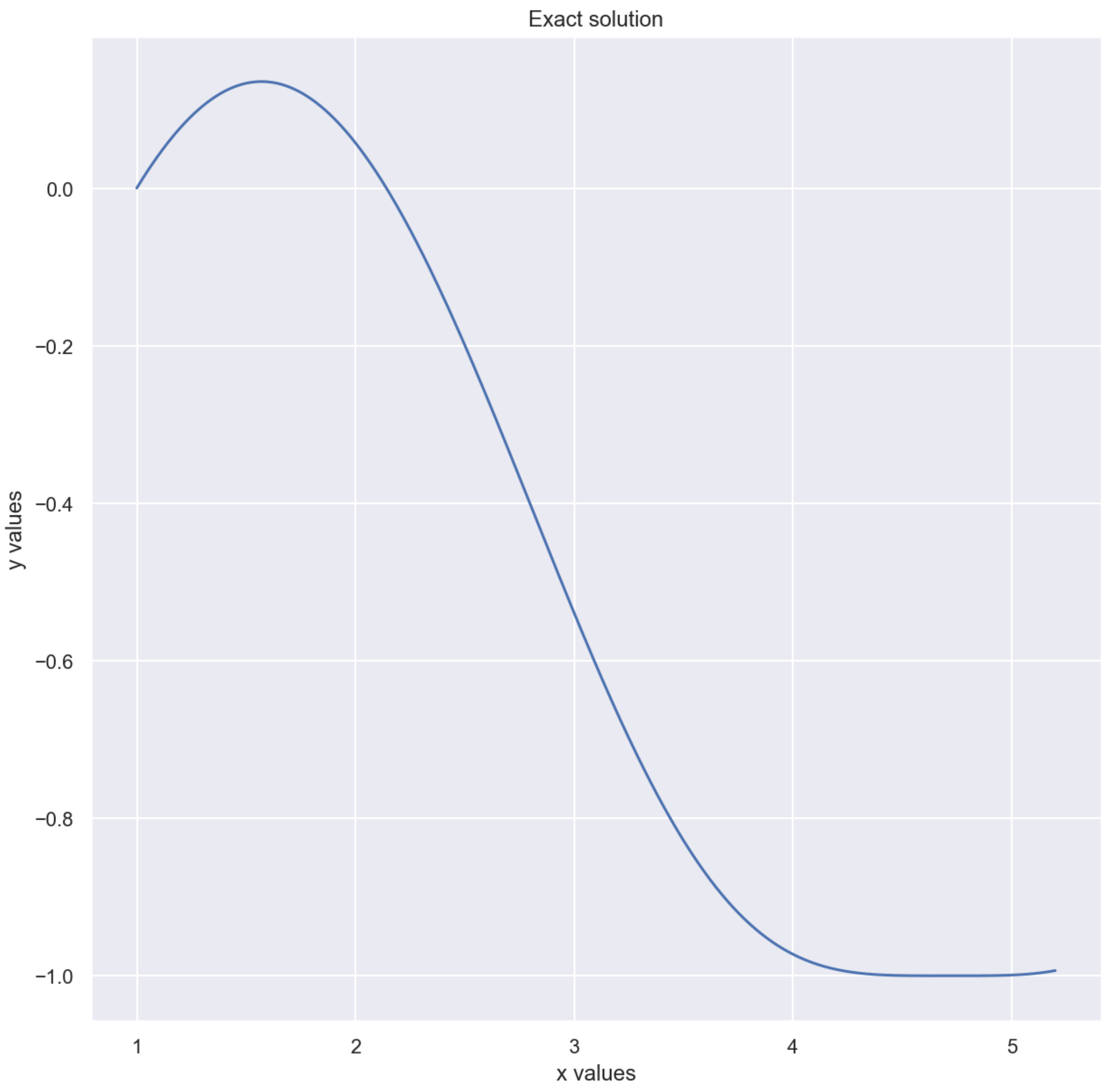
The first thing we need to do is to find the exact solution for given equation to compare it with computation results.

1. Type of the equation: first-order linear ordinary differential equation
2. Exact solution: $y(x) = c_1 e^{-\sin(x)} + \sin(x) - 1$
3. Find constant value: $\frac{y0 - \sin(x0) + 1}{e^{-\sin(x0)}}$
4. Calculate values for the defined step

```
def funct(x, y):
    return (1/2)*math.sin(2*x)-y*math.cos(x)
```

```
def solution(x, x0, y0):
    c = (y0-math.sin(x0)+1)/num.exp(-math.sin(x0))
    return math.sin(x) - 1 + c*num.exp(-math.sin(x))
```

```
def exact_sol(x0, y0, x, step):
    x_arr = num.arange(x0, x + step, step)
    y = []
    for x in x_arr:
        y.append(solution(x, x0, y0))
    return x_arr, y
```



Euler’s method

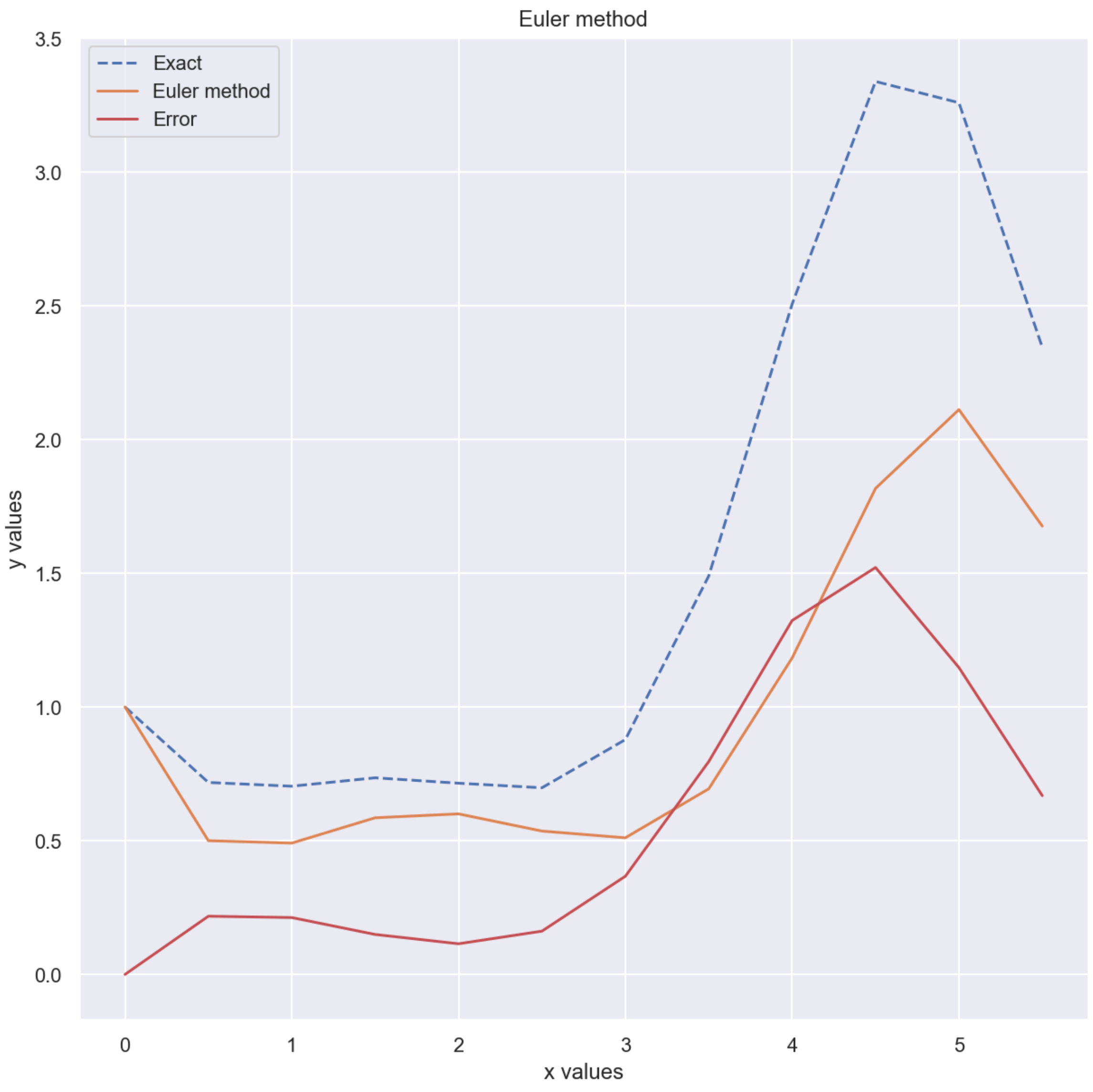
Euler’s method is a numerical method to generate an approximate solution to given DE and IVP for it. Using iterative formulaes

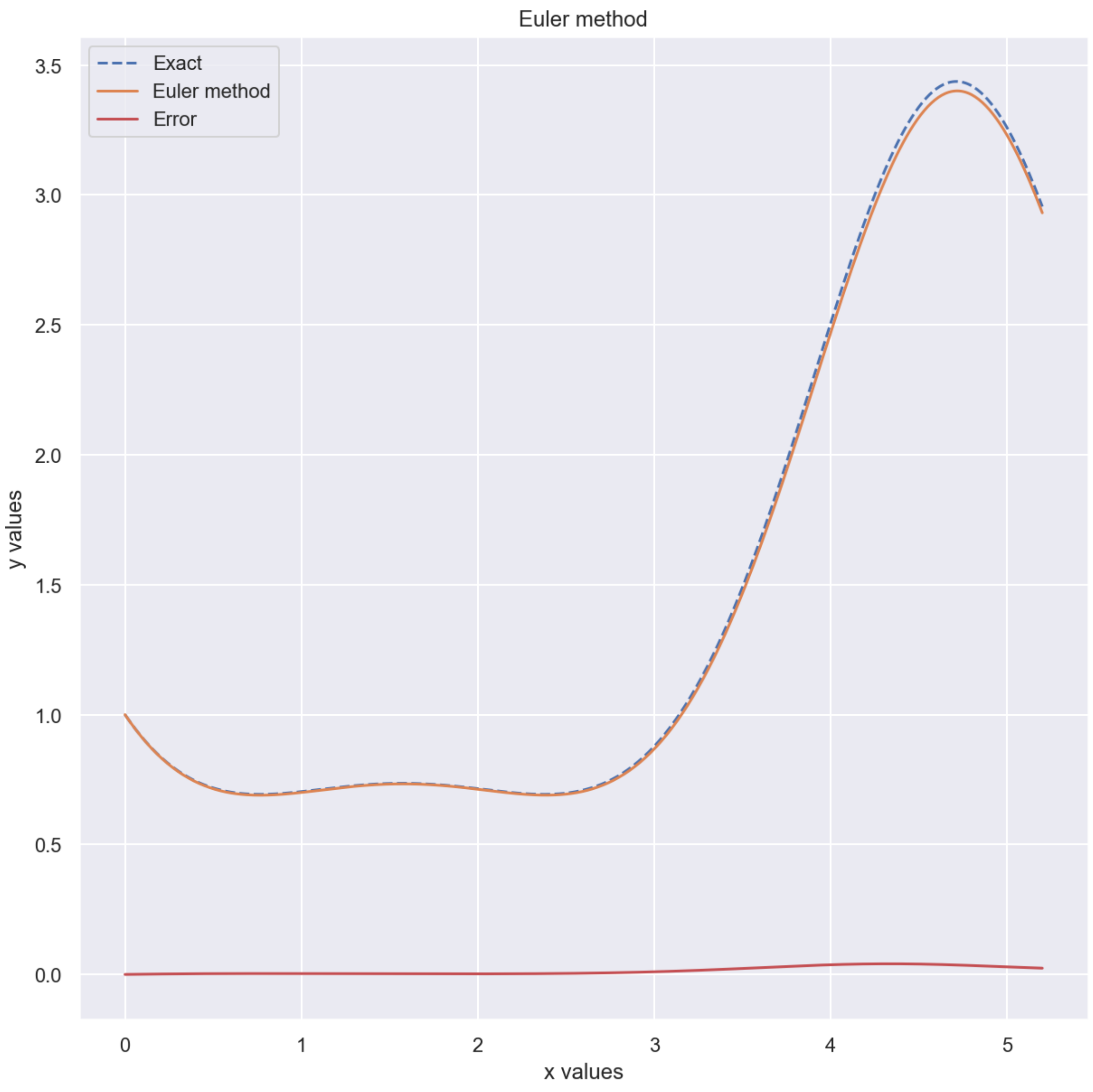
$$x_{n+1} = x_n + h$$

$$y_{n+1} = y_n + h\ f(x_n,\ y_n)$$

we build the solution.

```
def eulers(x0, y0, x, step):
    y = [y0]
    x_arr = num.arange(x0, x + step, step)
    error = []
    exact = []
    for x in x_arr:
        y_n = y[-1] + step * funct(x, y[-1])
        error.append(abs(solution(x, x0, y0) - y[-1]))
        y.append(y_n)
    return x_arr, y, error
```


Euler method
Exact
Euler method
Error
y values
x values
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5



If we will make step size smaller we will see that the exact solution and the plot of Euler's method plot will become more the same. Moreover, the smaller step we take the more precise solution we will get because for small step we can see that the error is close to zero and we can't see the same for the bigger one.

Improved Euler's method

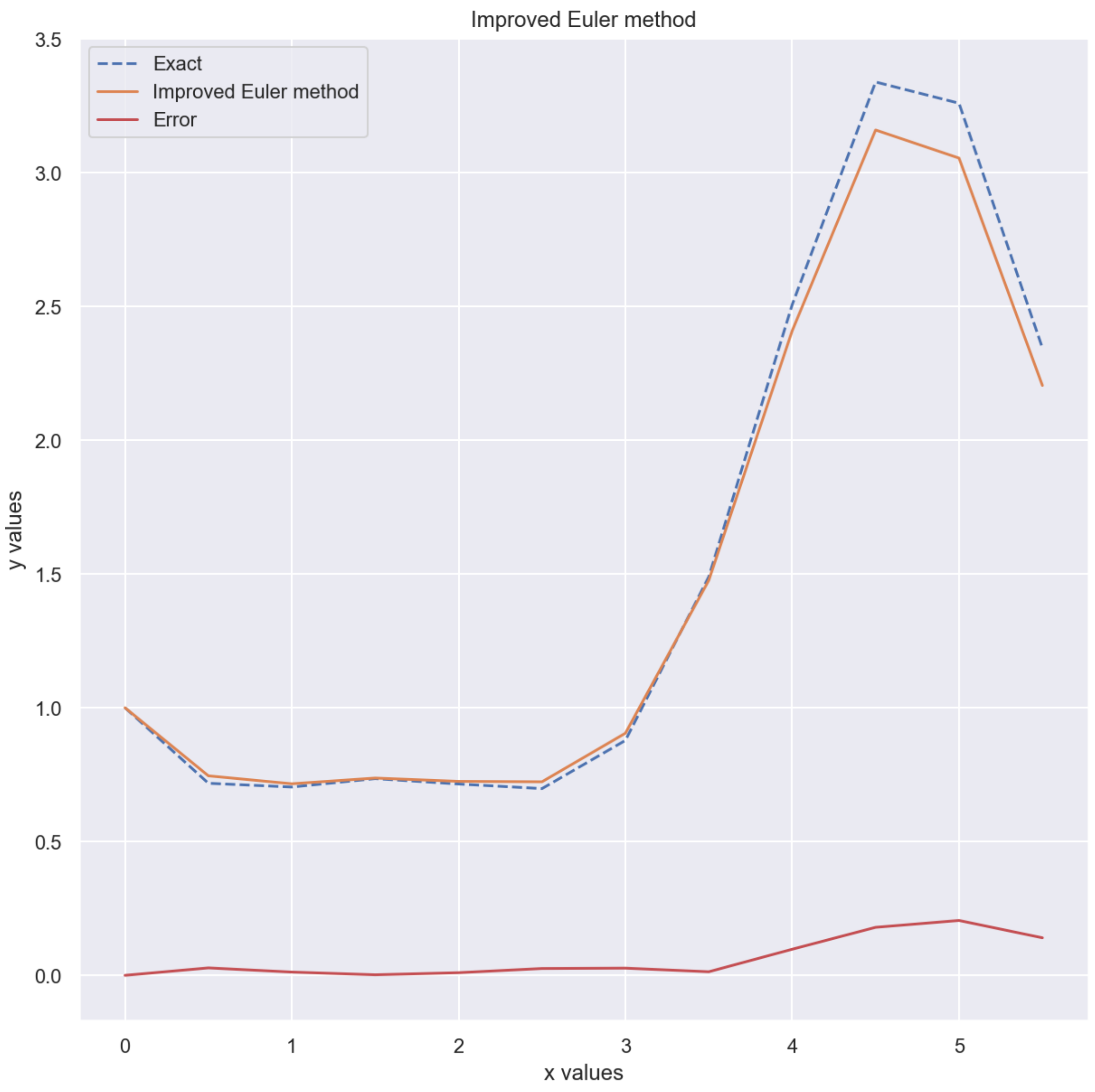
As approximation with Euler's method is not precise enough, we can apply Improved Euler's method to get more accurate solutions.

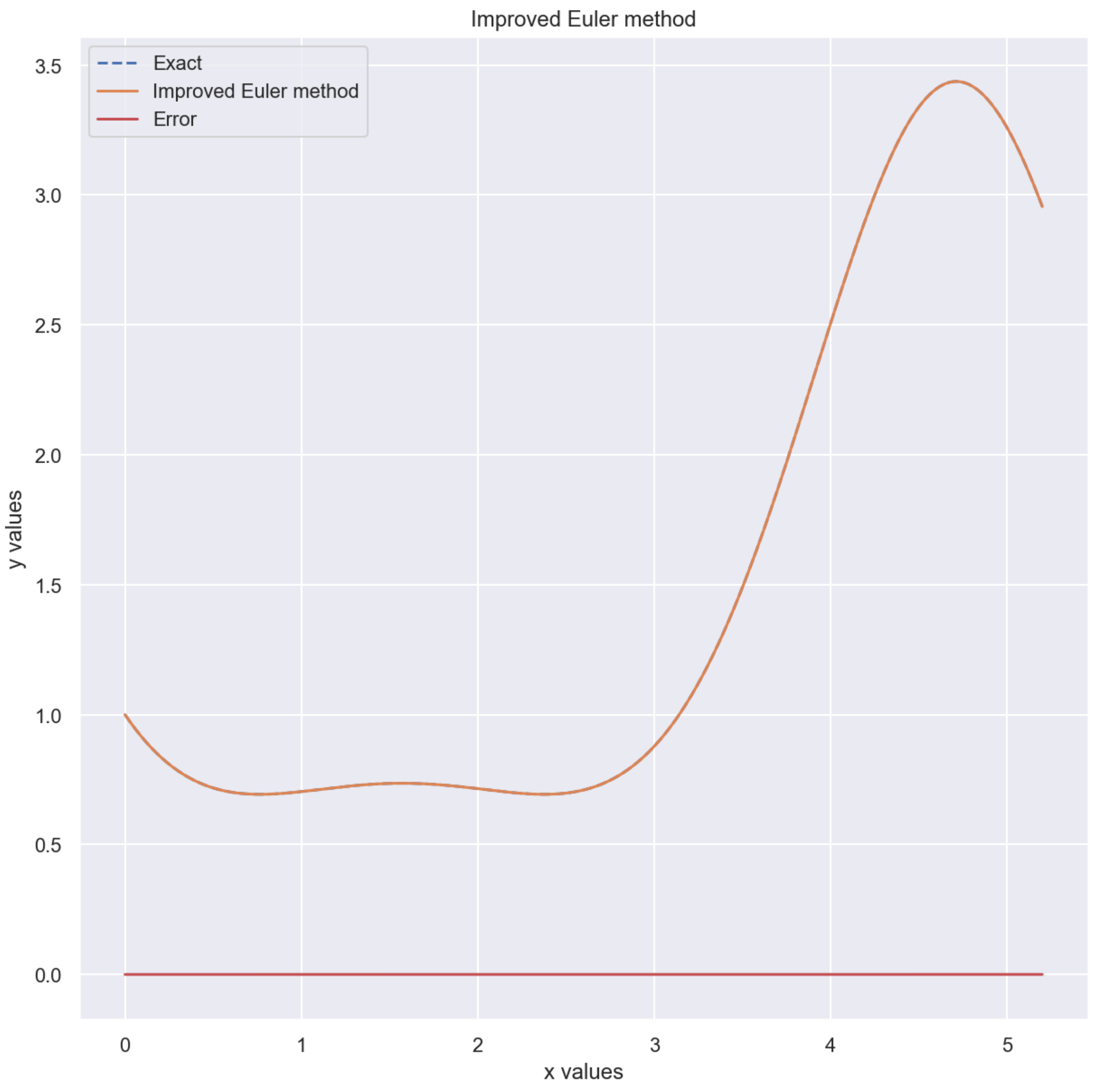
The Improved Euler's Method addressed these problems by finding the average of the slope based on the initial point and the slope of the new point, which will give an average point to estimate the value. It also decreases the errors that Euler's Method would have.

$$k_{1i} = f(x_i, y_i),$$
$$k_{2i} = f(x_i + h, y_i + hk_{1i}),$$
$$y_{i+1} = y_i + \frac{h}{2}(k_{1i} + k_{2i})$$

```
def improved_eulers(x0, y0, x, step):
    y = [y0]
    x_arr = num.arange(x0, x + step, step)
    error = []

    for x in x_arr:
        k1 = funct(x, y[-1])
        k2 = funct(x + step, y[-1] + step * k1)
        y_n = y[-1] + step * (k1 + k2) / 2
        error.append(abs(solution(x, x0, y0) - y[-1]))
        y.append(y_n)
    return x_arr, y, error
```





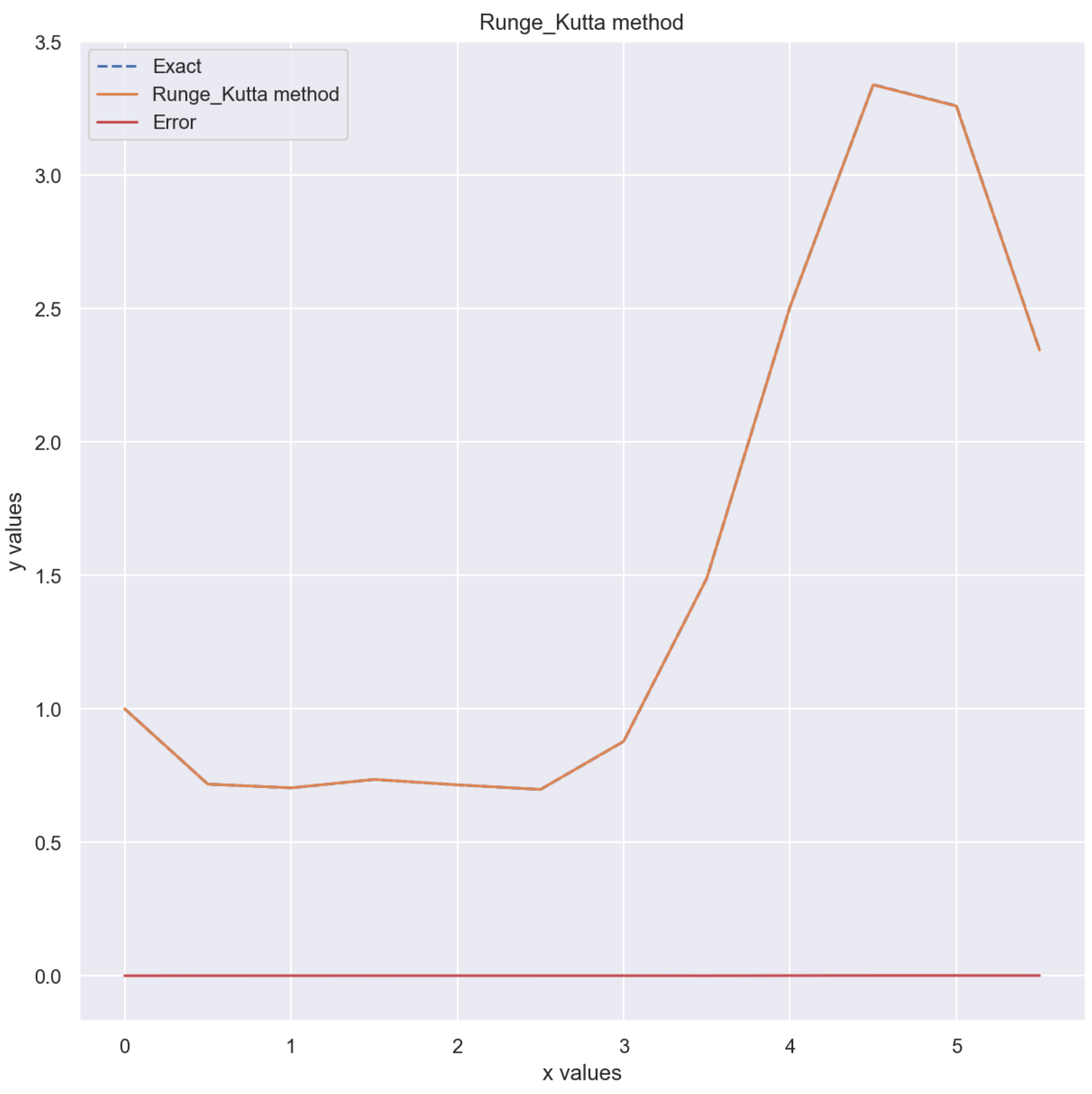
On bigger step we see that we have error so close to the zero and the plot that is more precise than the Euler's method. So, we can see that the Improved Euler's method is more accurate than the Euler's method. On the small step size we will see that the exact solution plot and Improved Euler's plot are identical and the error is equal to zero.

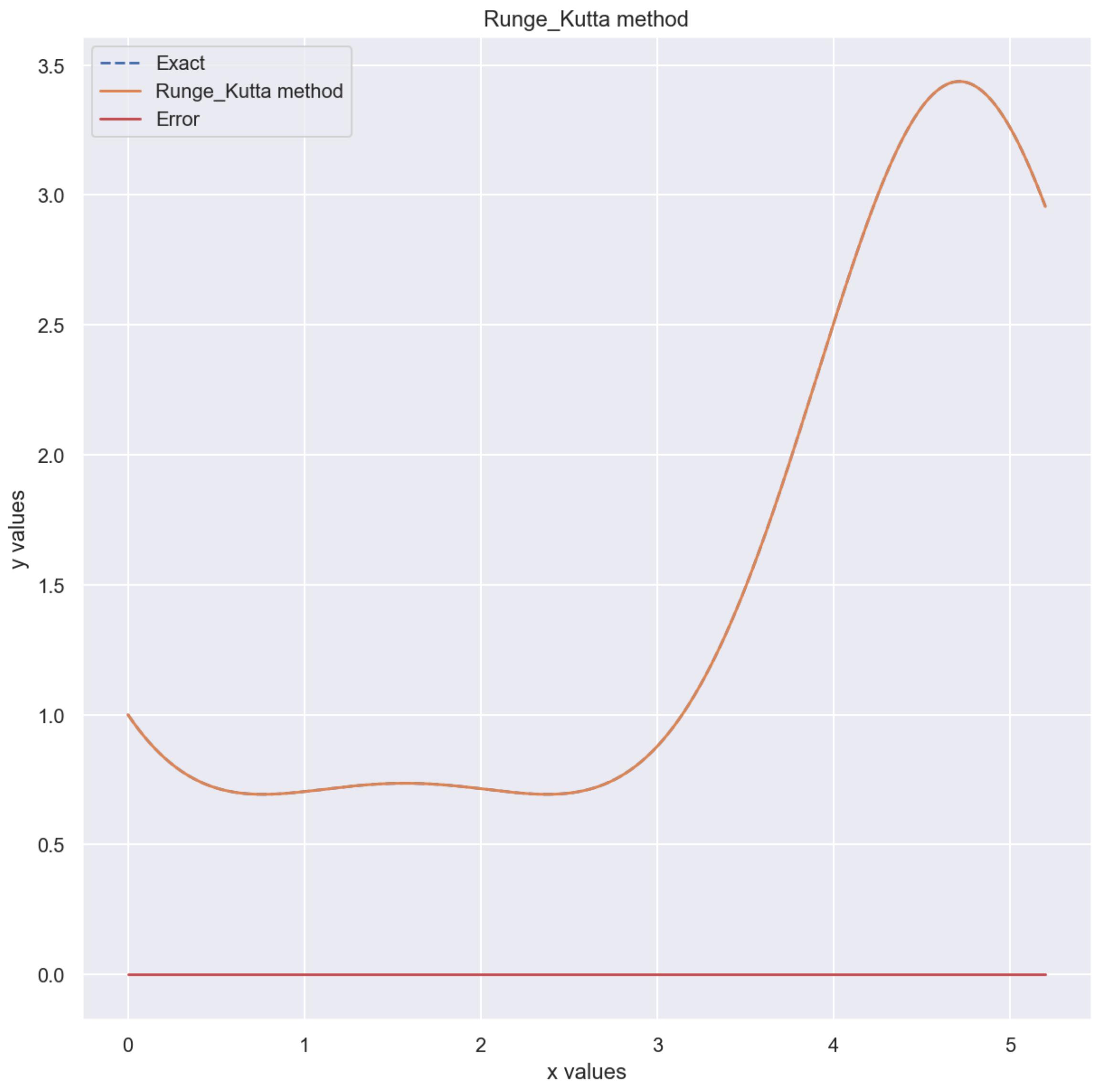
Runge-Kutta method

There are many ways to evaluate the right-hand side
that all agree to first order, but that have different coefficients of higher-order
error terms. Adding up the right combination of these, we can eliminate the error
terms order by order. That is the basic idea of the Runge-Kutta method.
The fourth-order Runge-Kutta method requires four evaluations of the right-
hand side per step.

$$k_{1i} = f(x_i, y_i)$$
$$k_{2i} = f(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_{1i})$$
$$k_{3i} = f(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_{2i})$$
$$k_{4i} = f(x_i + h, y_i + hk_{3i})$$
$$y_{i+1} = y_i + \tfrac{h}{6}(k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i})$$

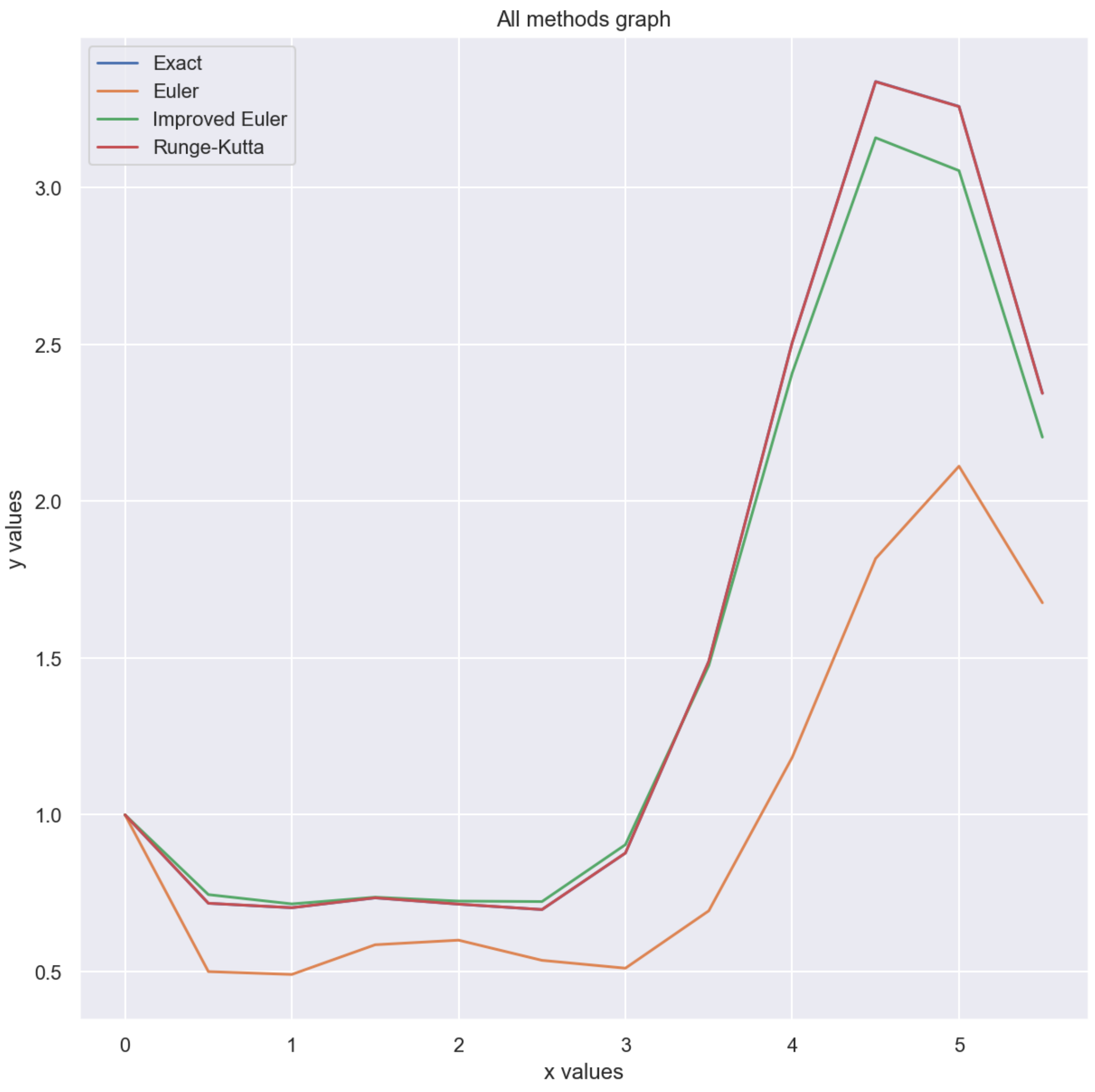
```
def runge_kutta(x0, y0, x, step):
    y = [y0]
    x_arr = num.arange(x0, x + step, step)
    error = []
    for x in x_arr:
        k1 = funct(x, y[-1])
        k2 = funct(x + step / 2, y[-1] + step * k1 / 2)
        k3 = funct(x + step / 2, y[-1] + step * k2 / 2)
        k4 = funct(x + step, y[-1] + step * k3)
        y_n = y[-1] + step * (k1 + 2 * k2 + 2 * k3 + k4) / 6
        error.append(abs(solution(x, x0, y0) - y[-1]))
        y.append(y_n)
    return x_arr, y, error
```

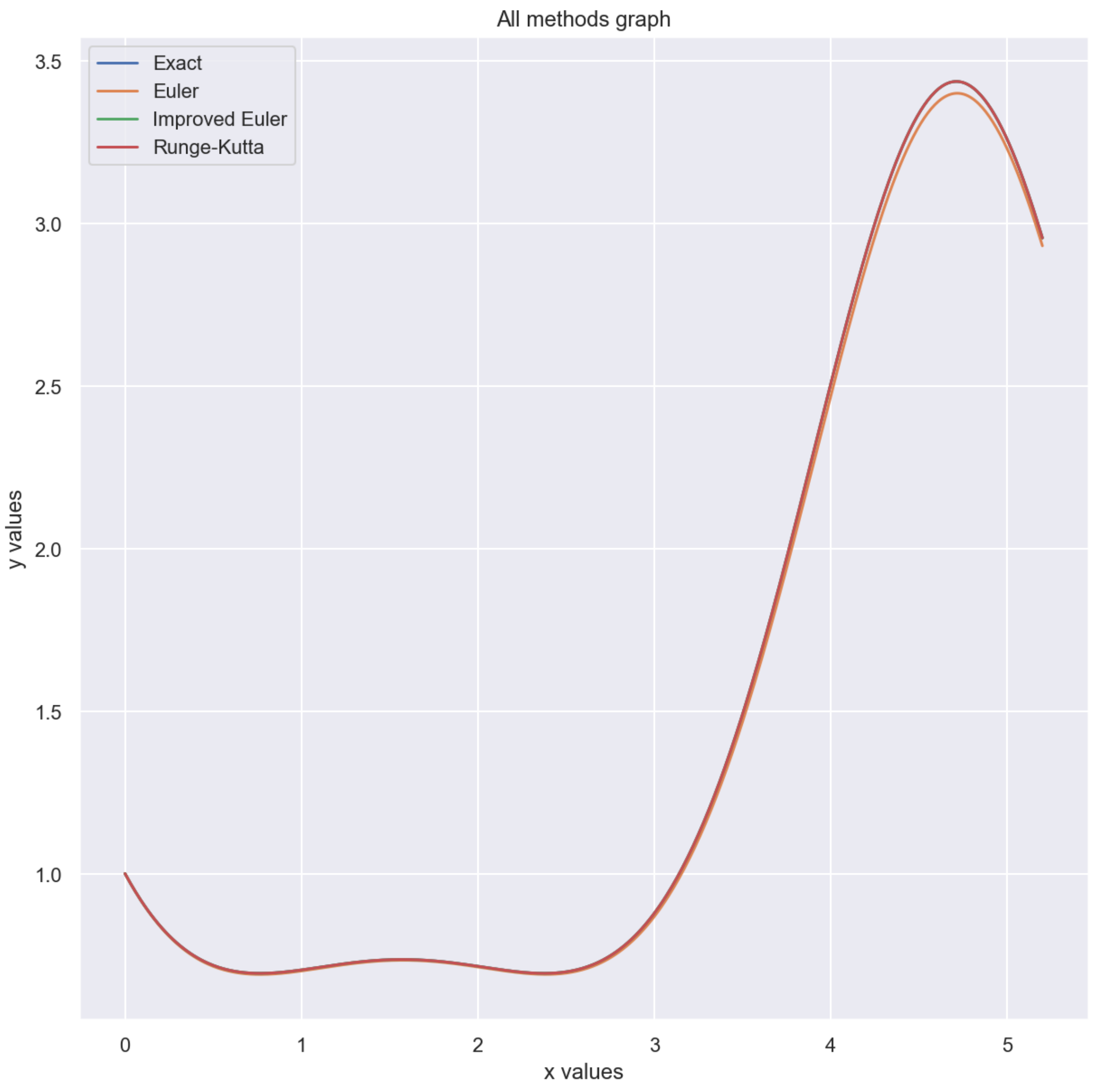




There we can see that plot of this method is identical to the exact solution on sny step size becuse its error is zero.

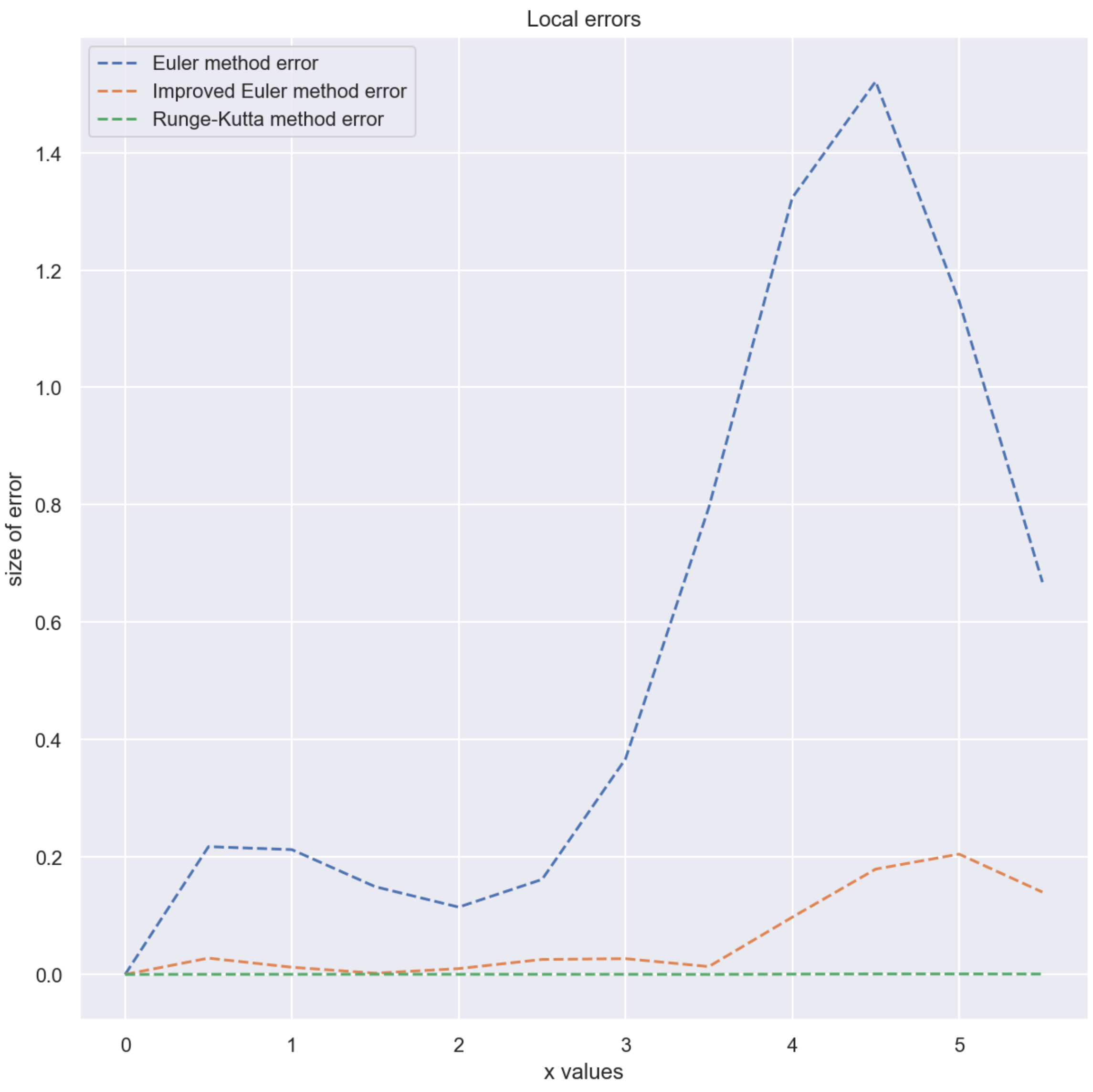
All methods plot

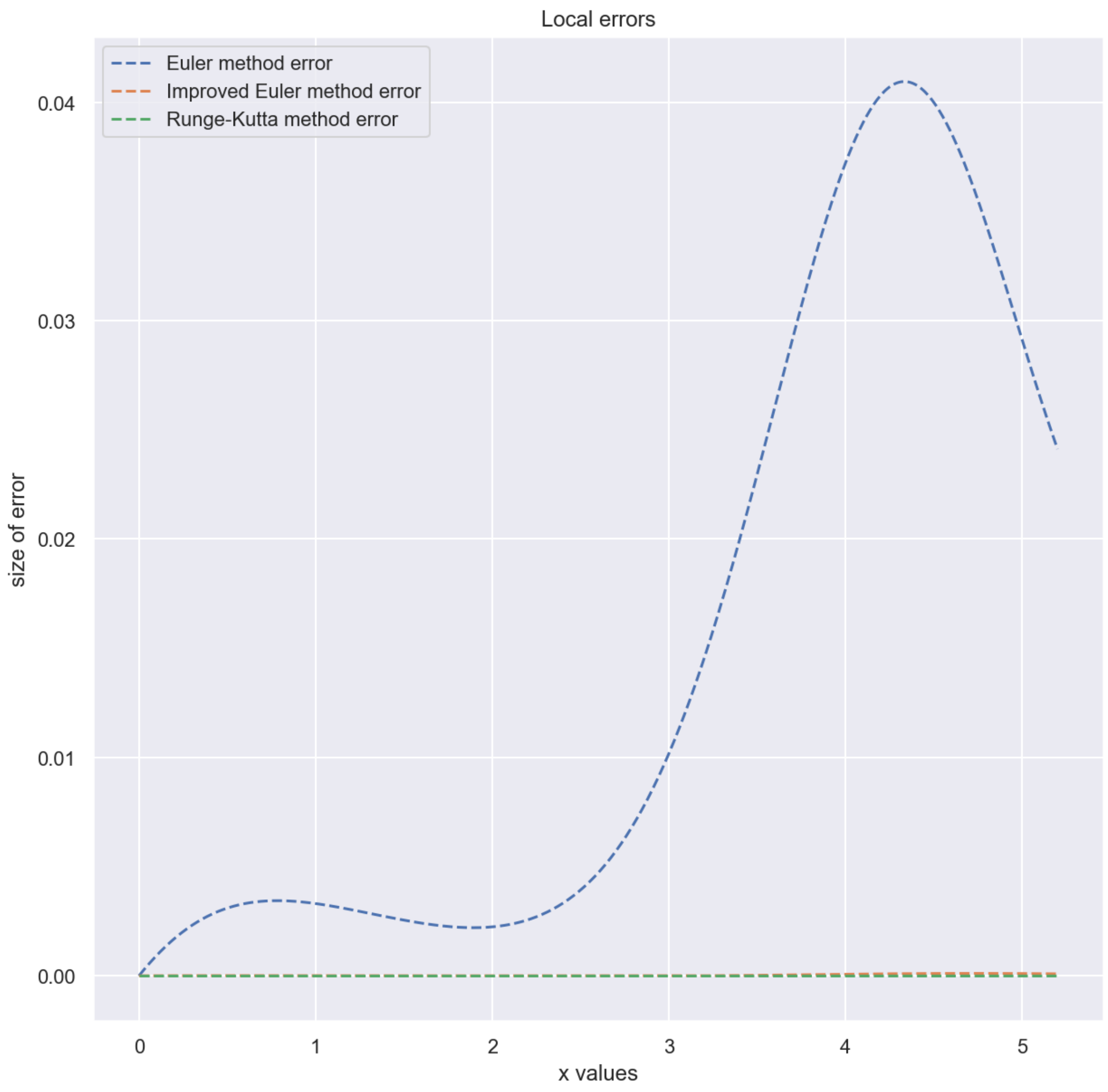

All methods graph
Exact
Euler
Improved Euler
Runge-Kutta
y values
x values
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5



On this plot we can see the difference between all of this methods. So, we see that the Euler's method is rough and less accurate and precise than two others. Improved Euler's method is close to the exact solution on small step bud idetical to it on the small steps, it means that it is more accurate than ordinary Euler's method but worse than Runge-Kutta. Runge-kutta is the most precise and accurate from this three methods.

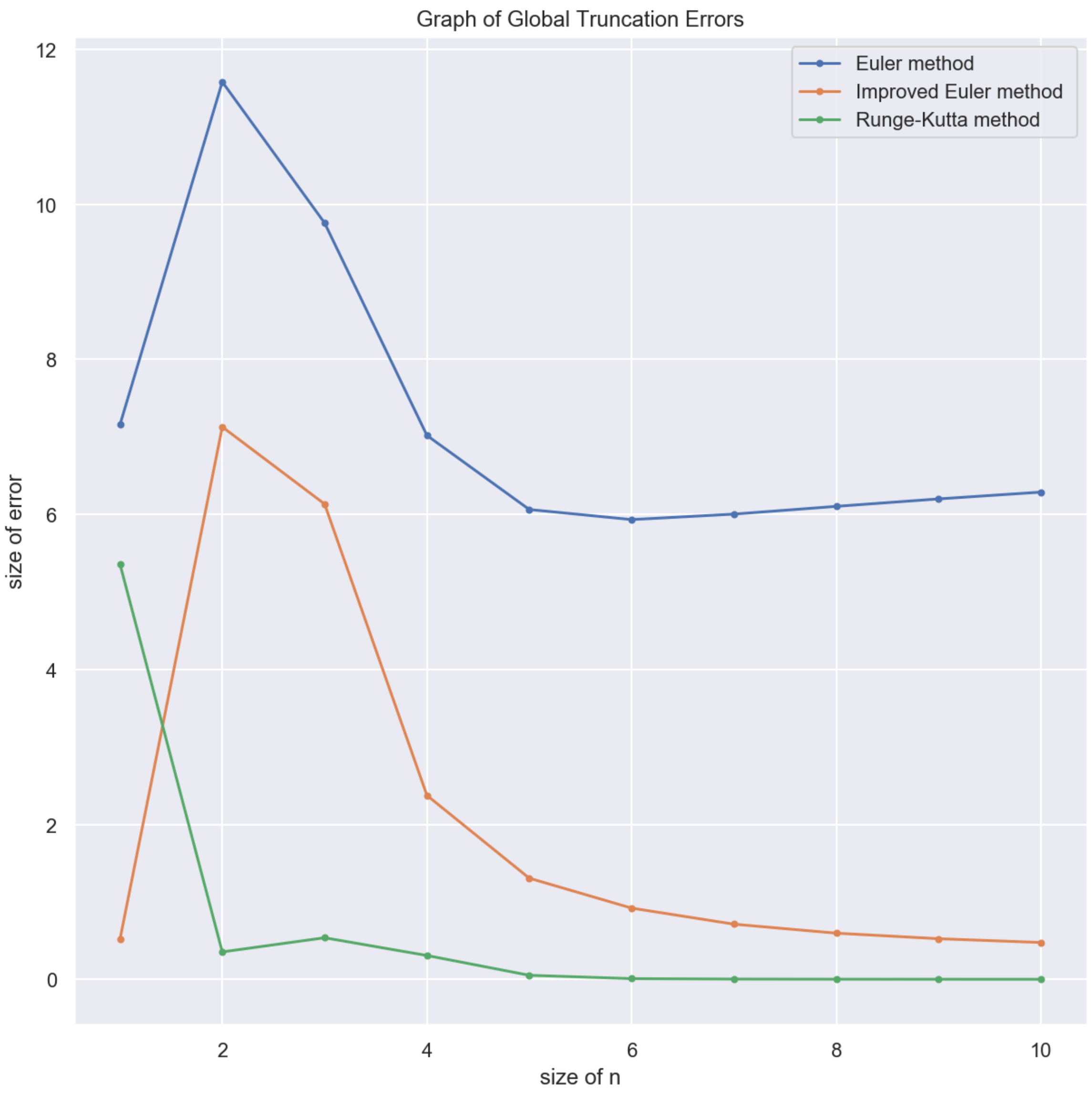
Local errors

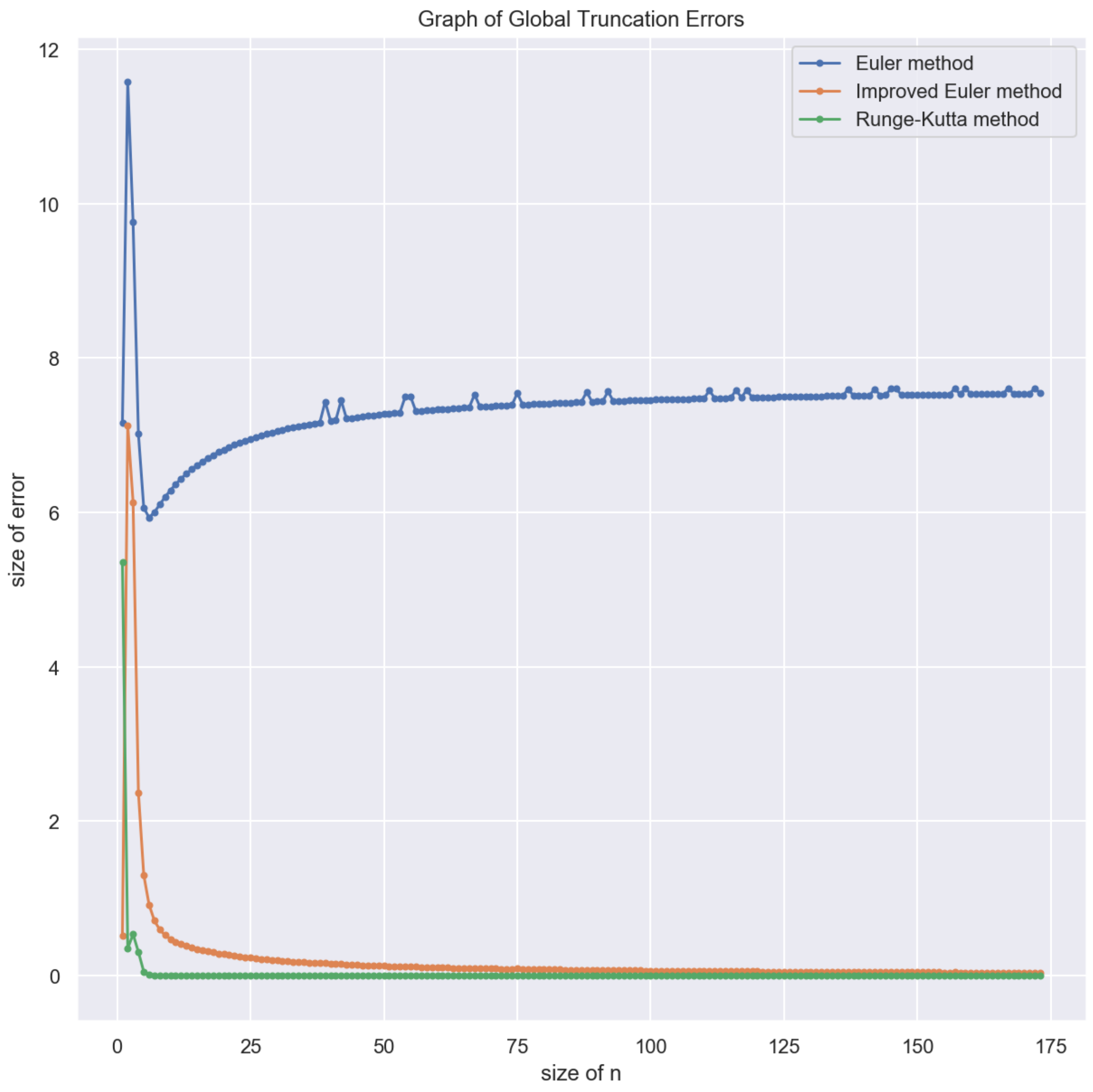

Local errors
Euler method error
Improved Euler method error
Runge-Kutta method error
size of error
x values
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
1
2
3
4
5



There we can see that errors of this methods define its accuracy and precision. So, the Euler's has the the biggest error and it is the most inaccurate method. Improved Euler's has less error but it is bigger than Runge-Kutta. Runge-Kutta has the error that is equal to zero and because of that it is the most accurate method.

Global errors


Graph of Global Truncation Errors
Euler method
Improved Euler method
Runge-Kutta method
size of error
size of n
0
2
4
6
8
10
12
2
4
6
8
10



There we can see that errors of Runge-Kutta and Improved Euler’s are tent to zero but for Euler’s errors are standing on the same level. So, Runge-Kutta and Improved Euler’s are tent to zero but with increasing n, errors for Runge-Kutta become smaller much faster than for the Euler’s method.