Differential Equations Computational Practice report

Problem statement

Given differential equation: $y' = 2xy + 5 - x^2$
Initial Value problem:
$x_0 = 0$,
$y_0 = 1$,
$X = 3$
Task : using Euler’s method, Improved Euler’s method and Runge-Kutta method provide a solution and analyze errors

Exact solution

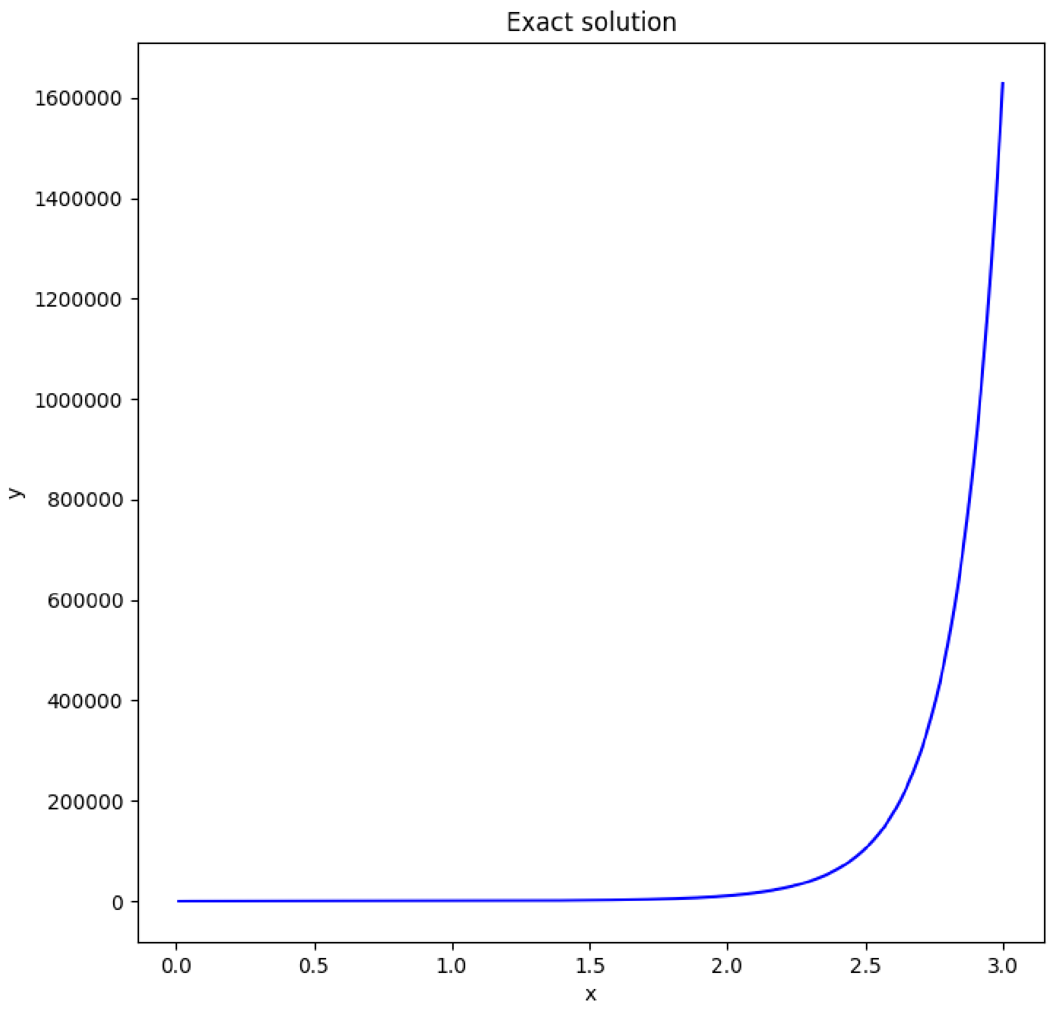
The first thing we need to do is to find the exact solution for given equation to compare it with computation results.

1. Type of the equation: first-order linear ordinary differential equation
2. Exact solution: $y = \frac{(x^2-4)}{2x}$
3. Find constant value: $c_1 = \frac{y_0 * 2x_0 - x_0^2 - 4}{2x_0 e^{x_0^2}}$
4. Calculate values for the defined step

```
def f(x, y):
    return 2 * y * x + 5 - x * x

def y_ivp(x, x0, y0):
    c1 = (y0 - (((x0 ** 2) - 4) / (2 * x0))) / (np.exp(x0 ** 2))
    return (((x ** 2) - 4) / (2 * x)) + c1 * np.exp(x ** 2)

def exact(y0, x0, xn, step):
    x_arr = np.arange(x0, xn + step, step)
    y = []
    for x in x_arr:
        y.append(y_ivp(x, x0, y0))
    return x_arr, y
```



Euler’s method

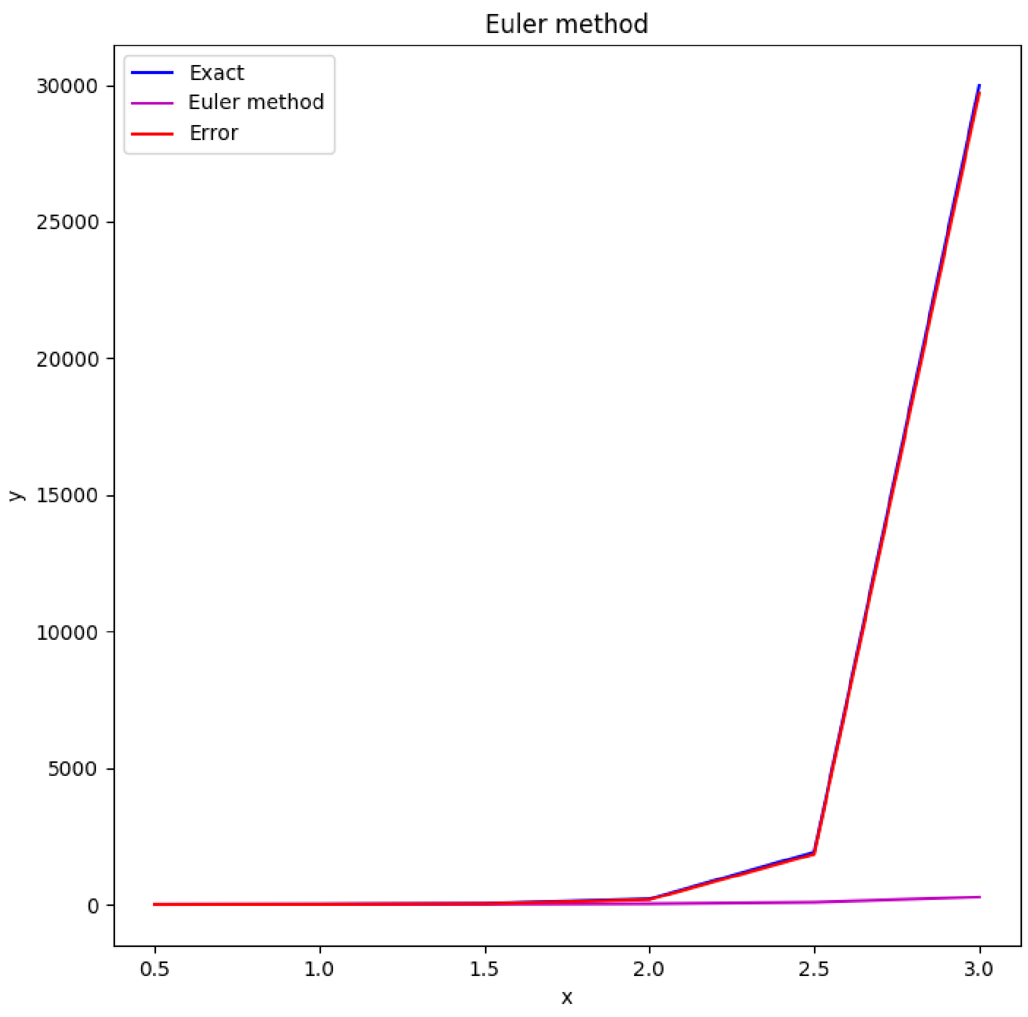
Euler’s method is a numerical method to generate an approximate solution to given DE and IVP for it. Using iterative formulas

$$x_{n+1} = x_n + h$$
$$y_{n+1} = y_n + hf(x_n, y_n)$$

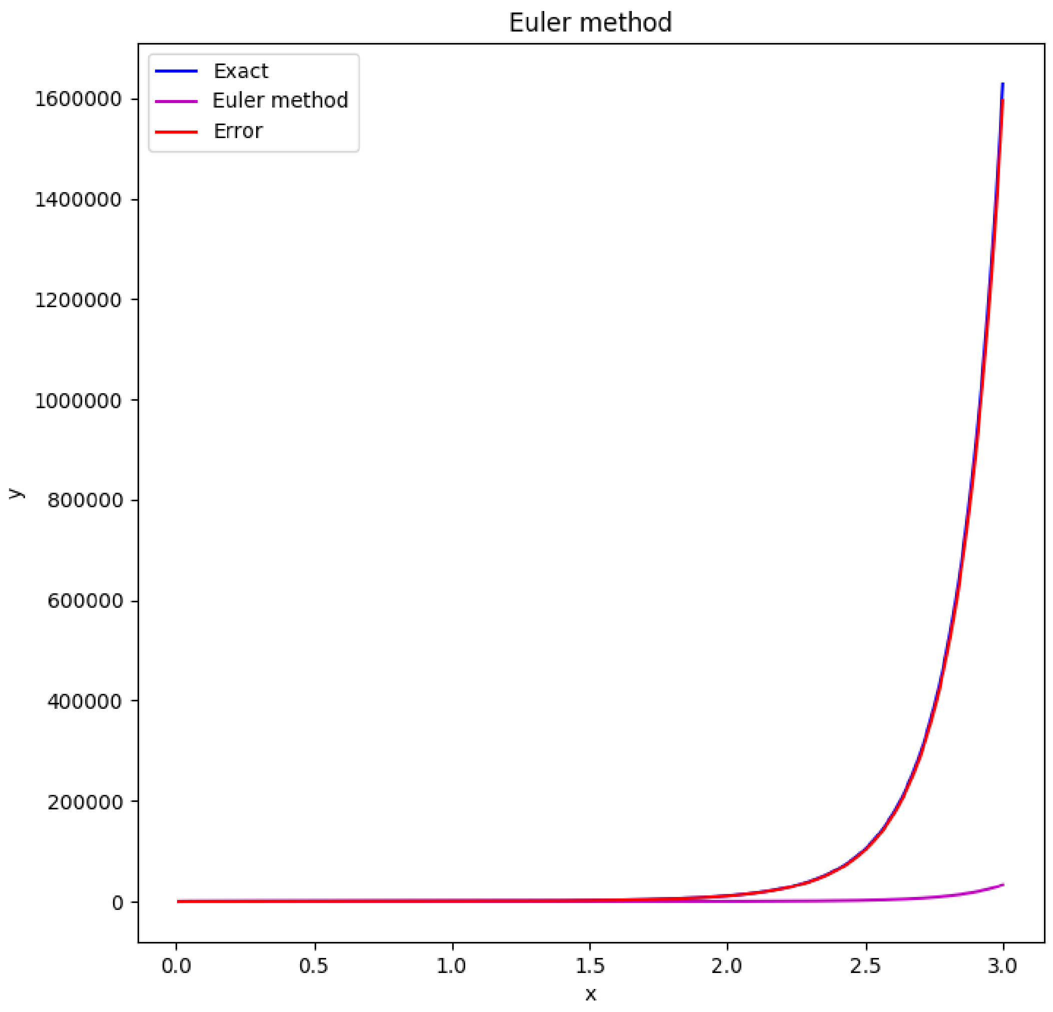
we build the solution.

```
def euler_method(y0, x0, xn, step):
    y = [y0]
    x_arr = np.arange(x0, xn + step, step)
    error = []
    for x in x_arr:
        y_n = y[-1] + step * f(x, y[-1])
        error.append(y_ivp(x, x0, y0) - y[-1])
        y.append(y_n)
    return x_arr, y, error
```

$step = 0.5$


Euler method
Exact
Euler method
Error
30000
25000
20000
15000
10000
5000
0
y
0.5
1.0
1.5
2.0
2.5
3.0
x

step = 0.01



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 closer 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.

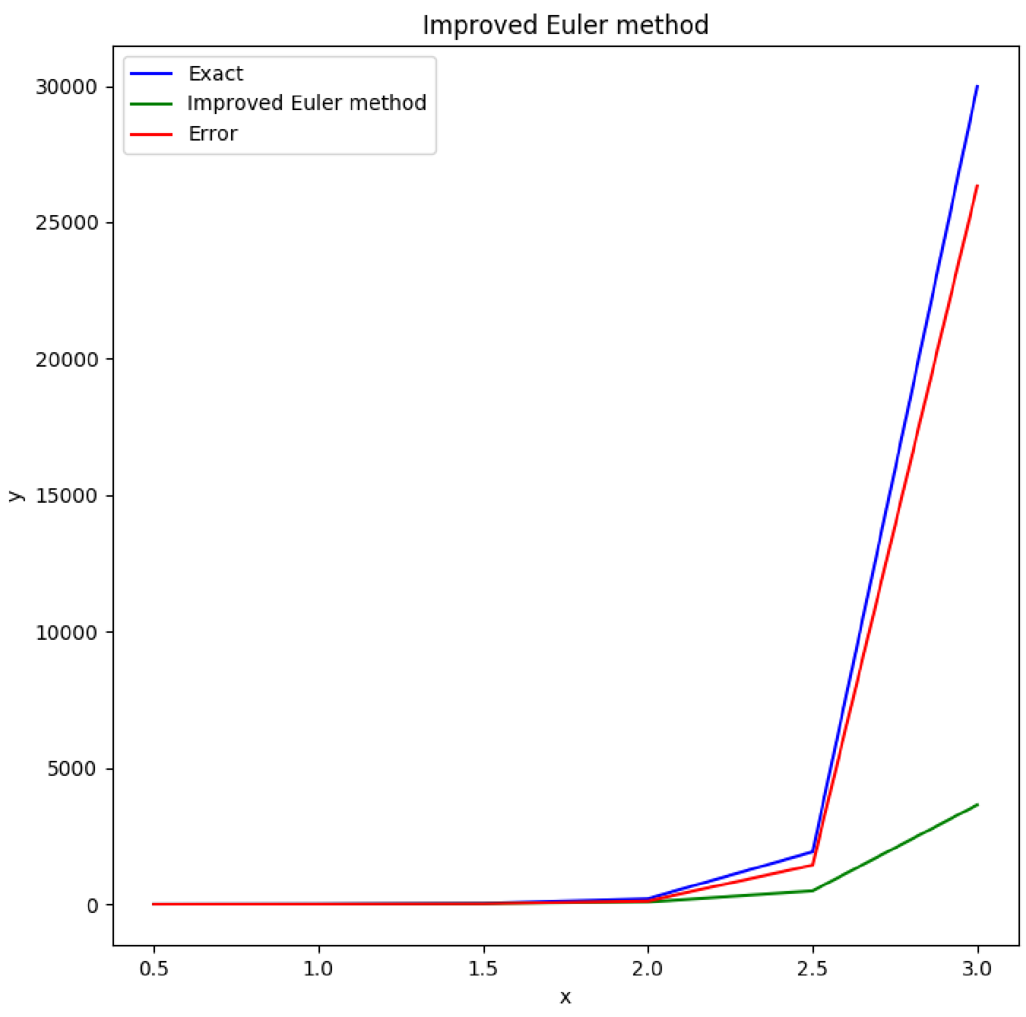
$k_{1i} = f(x_i, y_i),$

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

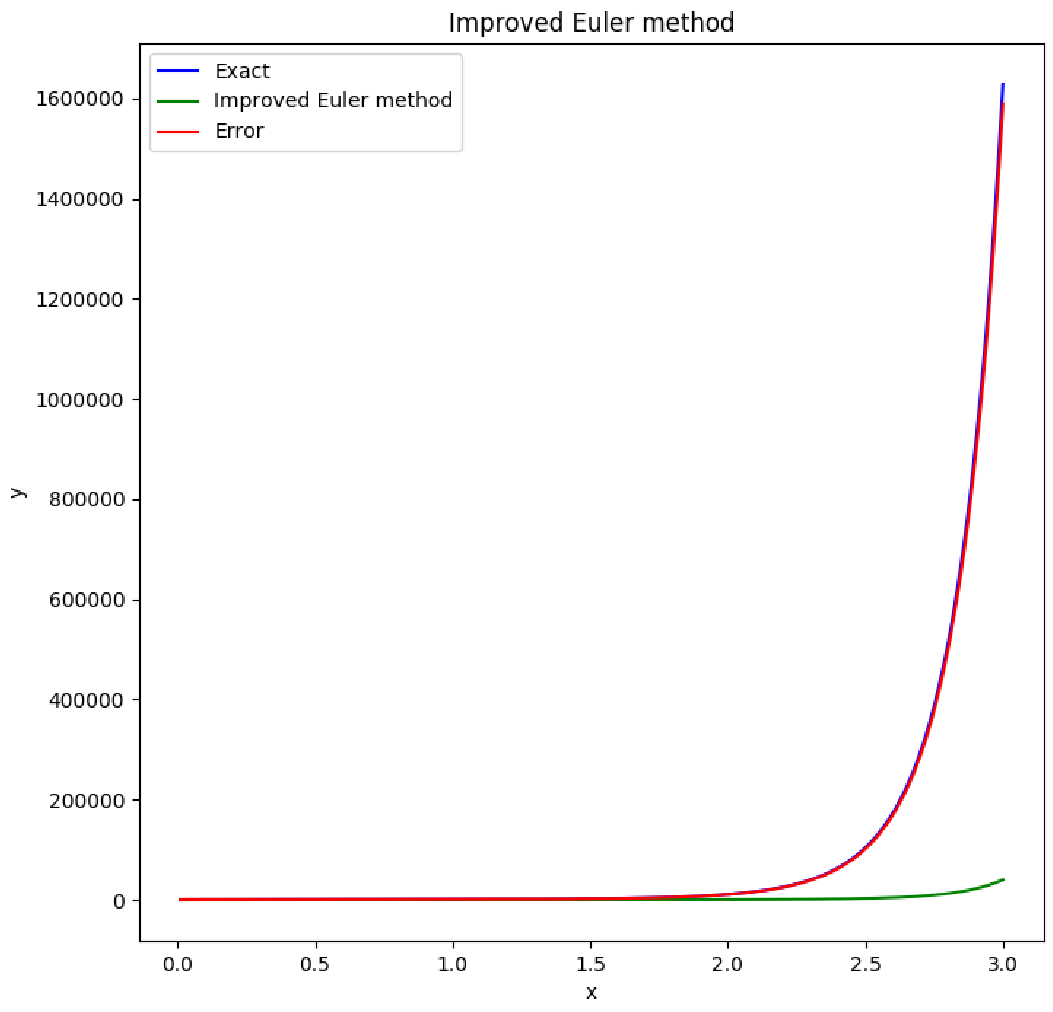
$y_{i+1} = y_i + \frac{h}{2}(k_{1i} + k_{2i})$

```
def imp_euler(y0, x0, xn, step):
    y = [y0]
    x_arr = np.arange(x0, xn + step, step)
    error = []
    for x in x_arr:
        k1 = f(x, y[-1])
        k2 = f(x + step, y[-1] + step * k1)
        y_n = y[-1] + step * (k1 + k2) / 2
        error.append(y_ivp(x, x0, y0) - y[-1])
        y.append(y_n)
    return x_arr, y, error
```

step = 0.5



step = 0.01



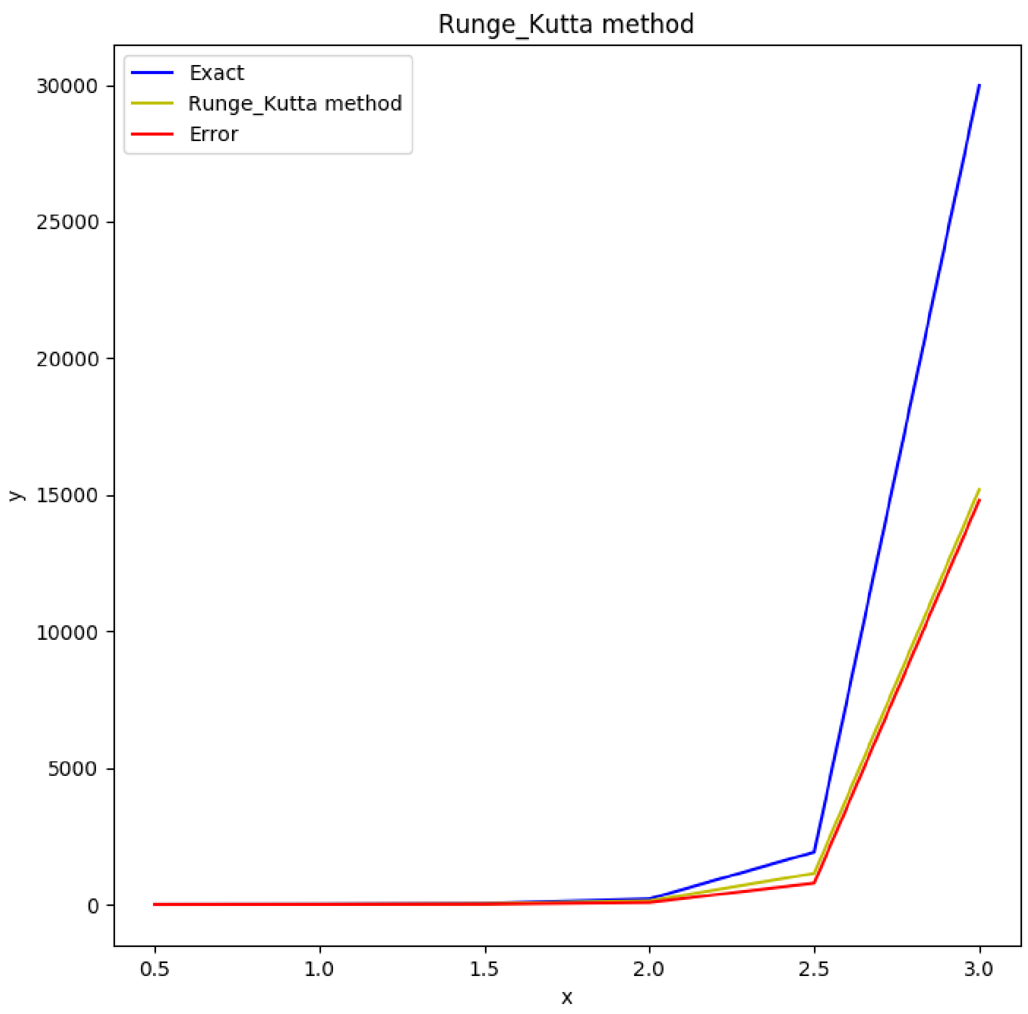
We can see that the Improved Euler's method is more accurate than the Euler's method.

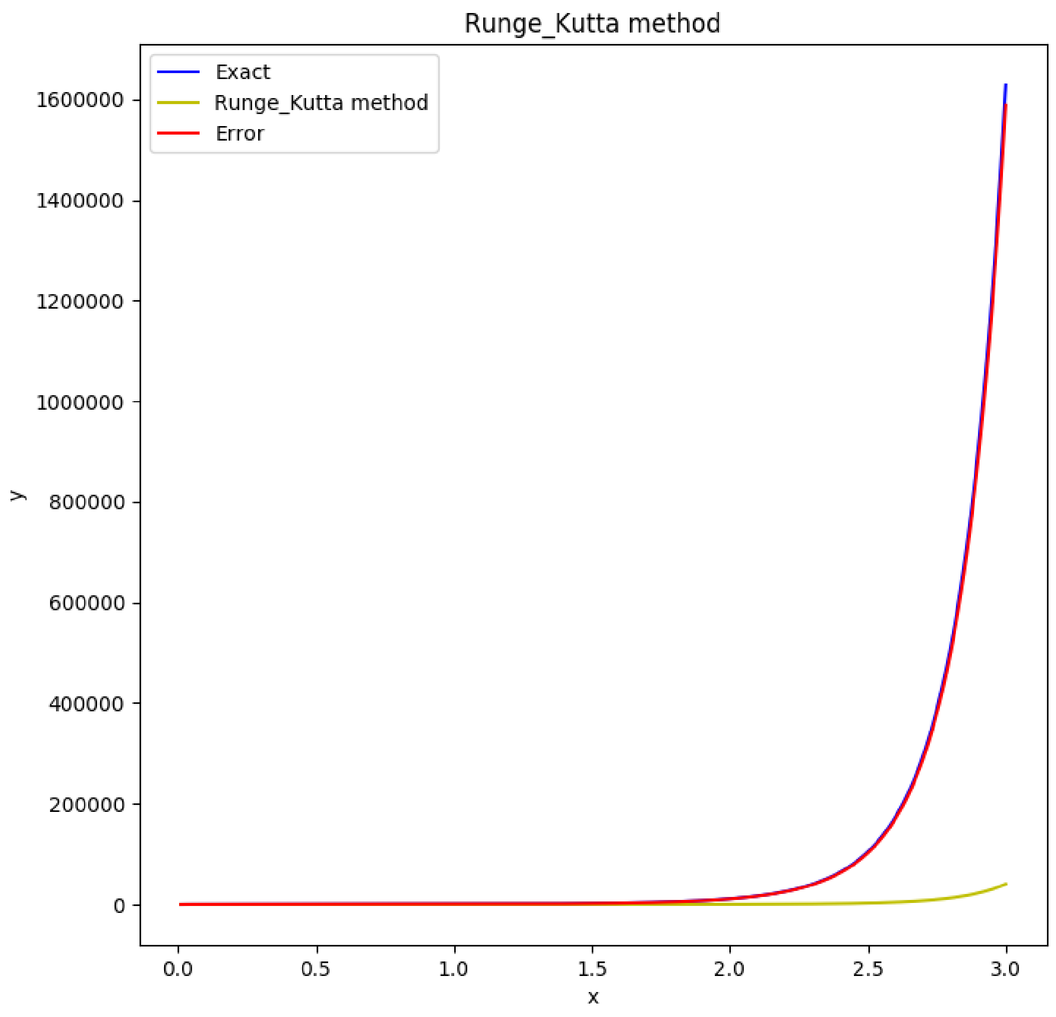
Runge-Kutta method

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.

$k_{1i} = f(x_i, y_i)$
$k_{2i} = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_{1i})$
$k_{3i} = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_{2i})$
$k_{4i} = f(x_i + h, y_i + hk_{3i})$
$y_{i+1} = yi + \frac{h}{6}(k_1 i + 2k_2 i + 2k_3 i + k_4 i)$

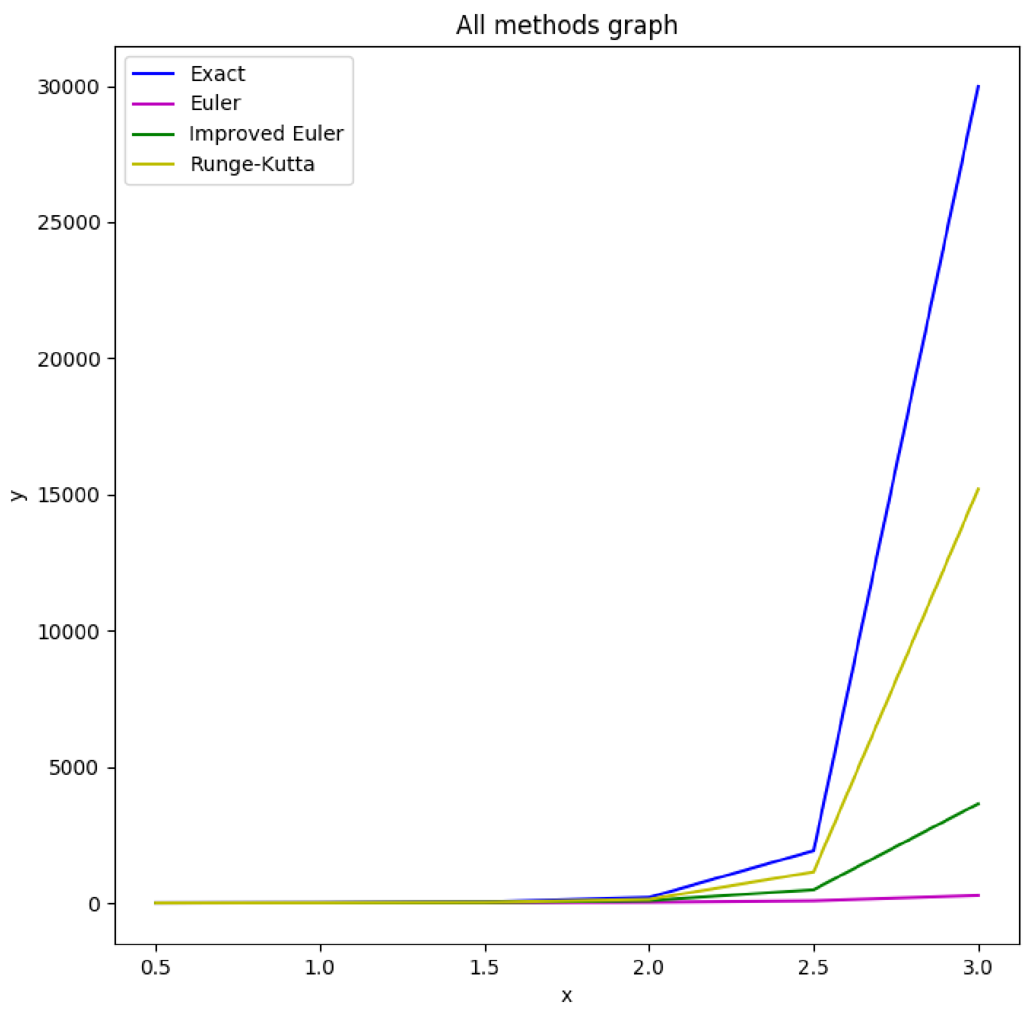
```
def runge_kutta(y0, x0, xn, step):
    y = [y0]
    x_arr = np.arange(x0, xn + step, step)
    error = []
    for x in x_arr:
        k1 = f(x, y[-1])
        k2 = f(x + step / 2, y[-1] + step * k1 / 2)
        k3 = f(x + step / 2, y[-1] + step * k2 / 2)
        k4 = f(x + step, y[-1] + step * k3)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



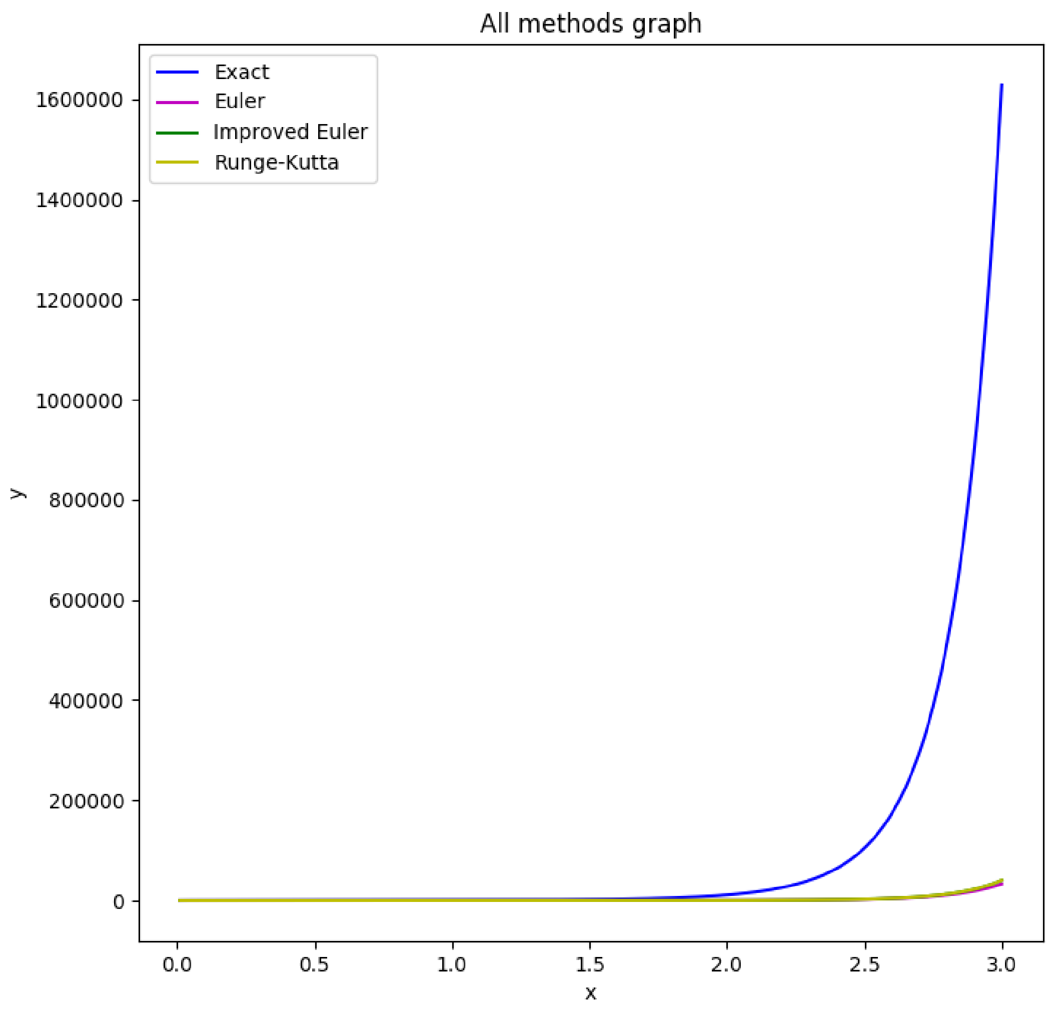


All methods plot

step = 0.5


All methods graph
Exact
Euler
Improved Euler
Runge-Kutta
30000
25000
20000
15000
10000
5000
0
y
0.5
1.0
1.5
2.0
2.5
3.0
x

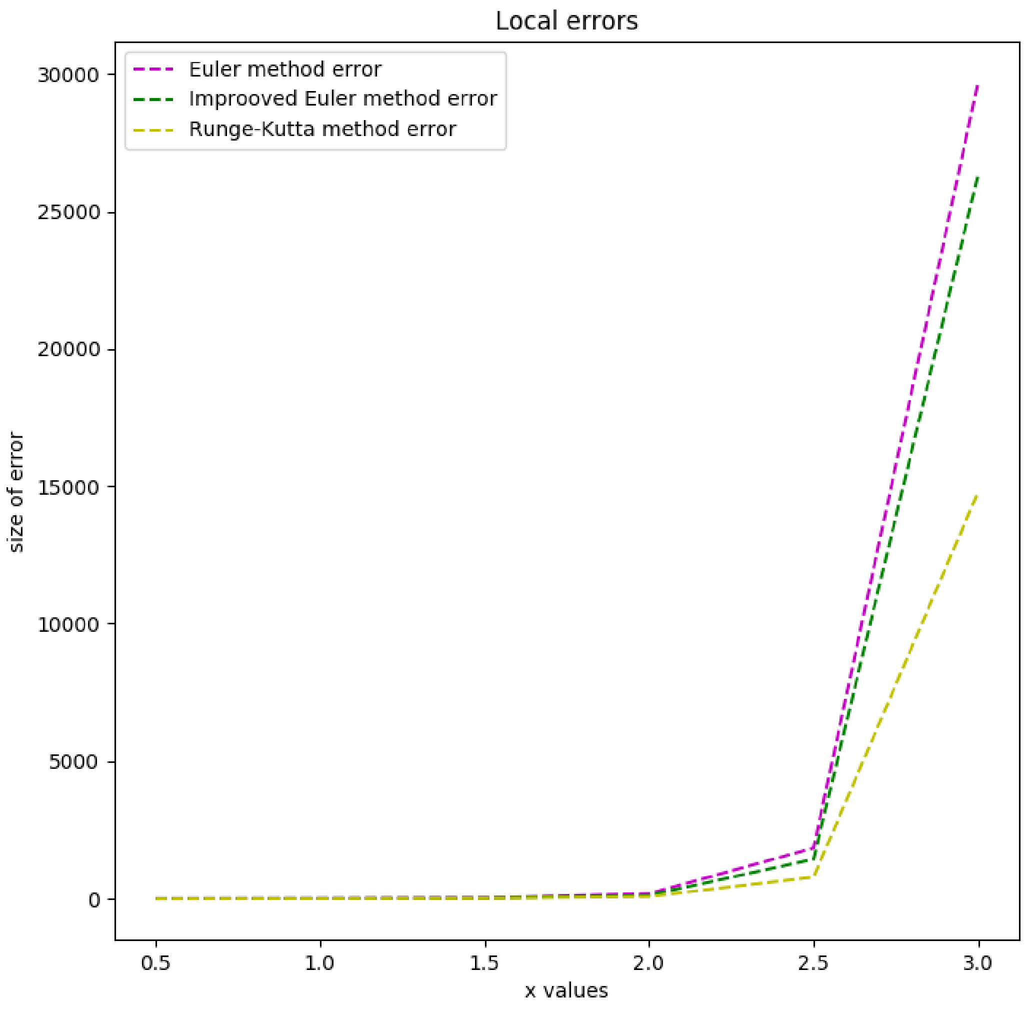
step = 0.01



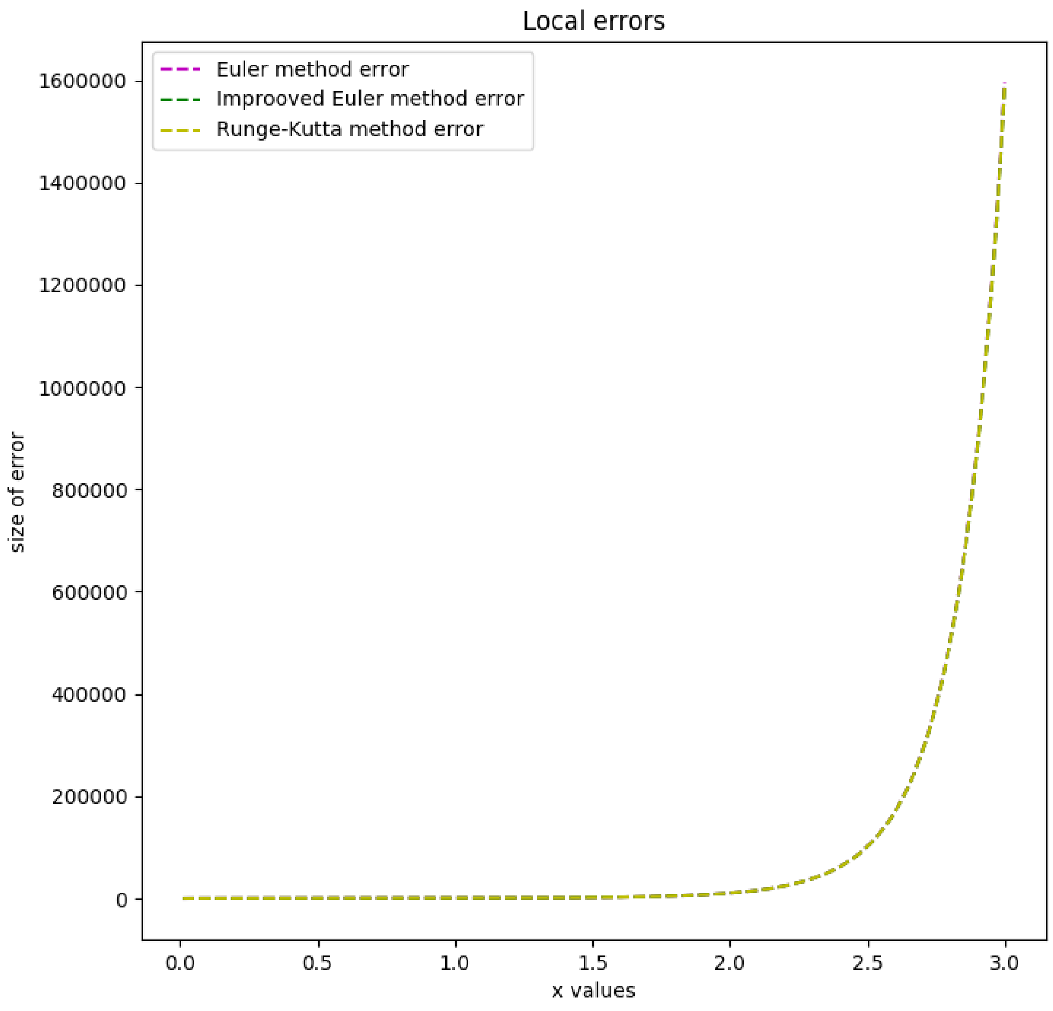
On this plot we can see the difference between all of this methods. So, we see that the Euler's method is rough and less accurate and precise than two others. Improved Euler's method is closer to the exact solution, it means that it is more accurate than ordinary Euler's method but worse than Runge-Kutta.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

Local errors

step = 0.5


Local errors
Euler method error
Improoved Euler method error
Runge-Kutta method error
size of error
x values
30000
25000
20000
15000
10000
5000
0
0.5
1.0
1.5
2.0
2.5
3.0

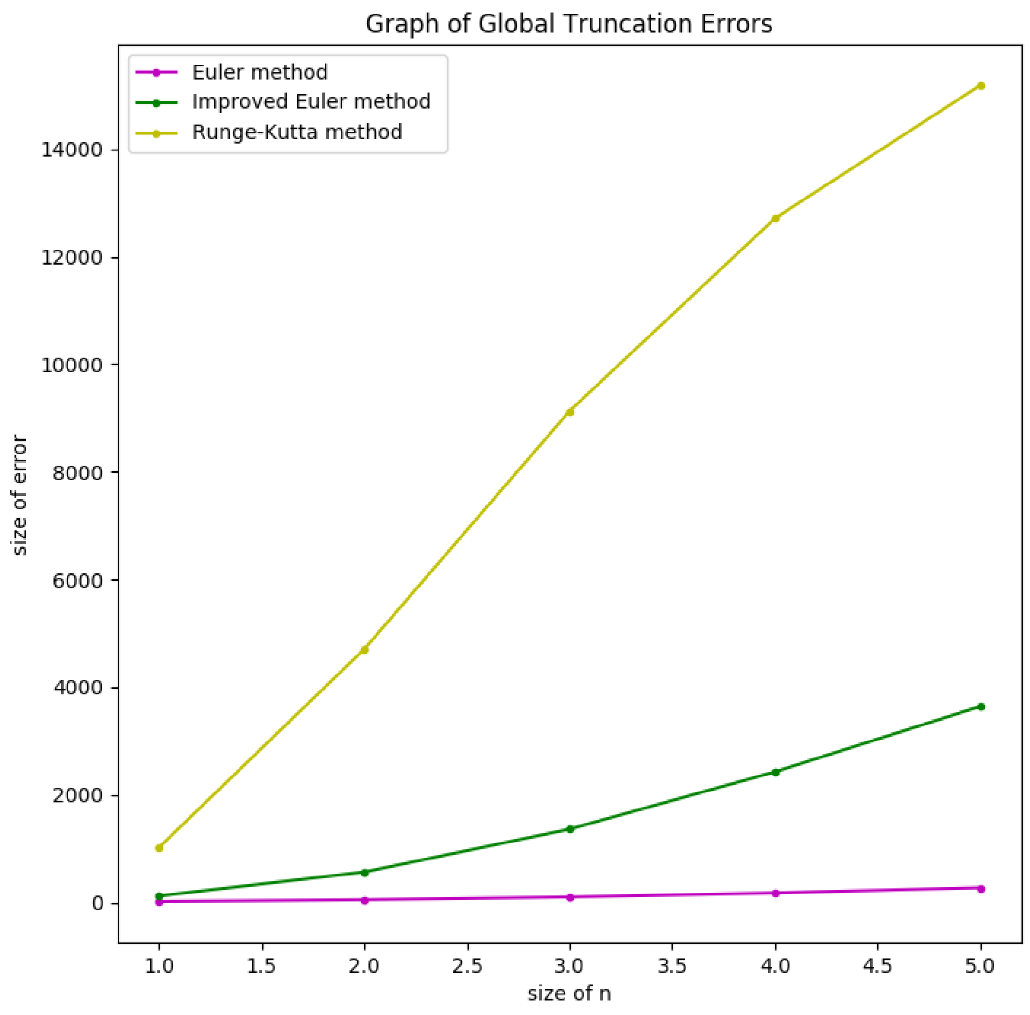
step = 0.01



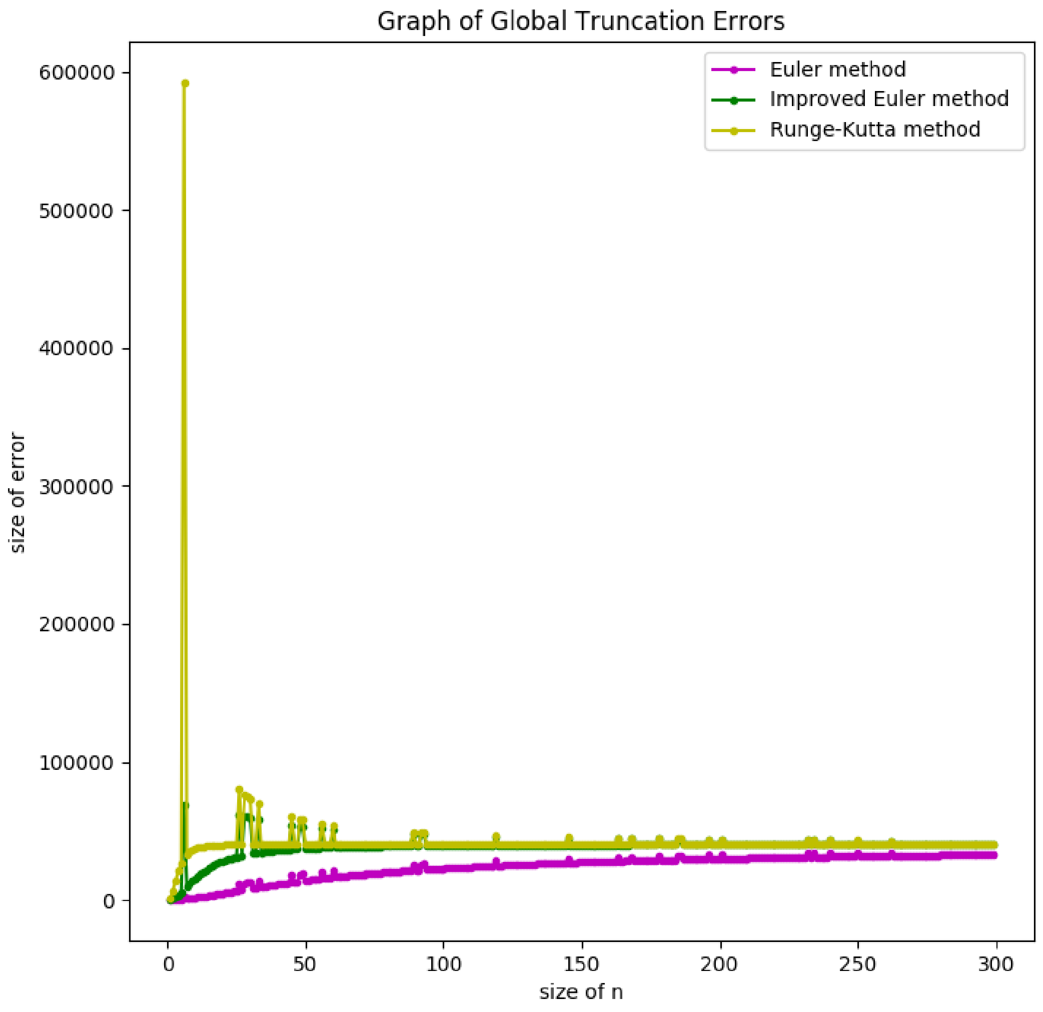
There we can see that errors of this methods define its accuracy and precision. So, the Euler's has the the biggest error and it is the most inaccurate method. Improved Euler's has less error but it is bigger than Runge-Kutta.

Global errors

step = 0.5


Graph of Global Truncation Errors
Euler method
Improved Euler method
Runge-Kutta method
14000
12000
10000
8000
6000
4000
2000
0
size of error
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
size of n

step = 0.01


Graph of Global Truncation Errors
Euler method
Improved Euler method
Runge-Kutta method
size of error
size of n
0
100000
200000
300000
400000
500000
600000
0
50
100
150
200
250
300