Initial Value Problem

$$y' + y\left(1 - \frac{1}{x}\right) = 3xe^x$$

$$vr + r'v + rv\left(1 - \frac{1}{x}\right) = 3xe^x$$

$$r' + r\left(1 - \frac{1}{x}\right) = 0$$

$$\frac{dr}{r} = \left(\frac{1}{x} - 1\right)dx$$

$$\ln|r| = \ln|x| - x$$

$$r = xe^{-x}$$

$$v'xe^{-x} = 3xe^x$$

$$dv = 3e^{2x}dx$$

$$v = \frac{3e^{2x}}{2} + C$$

$y(1) = 0$

$$y = vr = \frac{3e^x}{2} \cdot x + \frac{Cx}{e^x} = \frac{3xe^{2x} + 2Cx}{2ex}$$

$$0 = \frac{3}{2}e + \frac{C}{e} = \frac{3e^2 + 2C}{2e} \qquad C = -\frac{3}{2}e^2$$

Link to github:

https://github.com/Flamel001/de_programming_assignment/tree/master/programming_assignment_1

Functions for calculating

```python
import math


# function according to my variant
def f(x, y):
    return 3 * x * math.e ** x - y * (1 - 1 / x)


# function according to my solution of IVP
def f_exact(x):
    return (3 / 2 * x * math.e ** x) + (-3 / 2 * math.e ** 2) * x / math.e ** x


# function for improved euler method
def delta_y(x, y, h):
    return h * f(x + h / 2, y + h / 2 * f(x, y))


# function and subfunctions for runge kutta method
def runge_kutta_delta_y(x, y, h):
    return h / 6 * (k1(x, y) + 2 * k2(x, y, h) + 2 * k3(x, y, h) + k4(x, y, h))


def k1(x, y):
    return f(x, y)


def k2(x, y, h):
    return f(x + h / 2, y + h * k1(x, y) / 2)


def k3(x, y, h):
    return f(x + h / 2, y + h * k2(x, y, h) / 2)


def k4(x, y, h):
    return f(x + h, y + h * k3(x, y, h))
```

All the functions of ‘method name’ look the same, difference only in method called for “y”(79 line on screenshot), initial values taken from task description

```
70      def runge_kutta(steps_amount):
71          # Initial values
72          x = [1.0]
73          y = [0.0]
74          # step size
75          h = (5 - x[0]) / steps_amount
76          for i in range(steps_amount):
77              # here graph is created point by point
78              x.append(x[i]+h)
79              y.append(y[i] + runge_kutta_delta_y(x[i],y[i],h))
80          return x,y
```

Calculating and building graphs for methods and local errors

```
# accuracy
steps_amount = 3
# calculating every graph
x_euler, y_euler = functions.euler(steps_amount)
x_exact, y_exact = functions.exact(steps_amount)
x_euler_improved, y_euler_improved = functions.euler_improved(steps_amount)
x_runge_kutta, y_runge_kutta = functions.runge_kutta(steps_amount)

# local error calculating
euler_err, euler_improved_err, runge_kutta_err = [0.0], [0.0], [0.0]
for i in range(steps_amount):
    euler_err.append(fabs(y_exact[i] - y_euler[i]))
    euler_improved_err.append(fabs(y_exact[i] - y_euler_improved[i]))
    runge_kutta_err.append(fabs(y_exact[i] - y_runge_kutta[i]))
```

```
# here function graph is plotted
plt.title("Methods")
plt.plot(x_euler, y_euler, label="euler")
plt.plot(x_exact, y_exact, label="exact")
plt.plot(x_euler_improved, y_euler_improved, label="euler_improved")
plt.plot(x_runge_kutta, y_runge_kutta, label="runge_kutta")
plt.ylabel("y")
plt.xlabel("x")
plt.legend()
plt.show()

# here local error graph is plotted
plt.title("Local error")
plt.plot(x_euler, euler_err, label="euler")
plt.plot(x_euler_improved, euler_improved_err, label="euler_improved")
plt.plot(x_runge_kutta, runge_kutta_err, label="runge_kutta")
plt.ylabel("error")
plt.xlabel("x")
plt.legend()
plt.show()
```

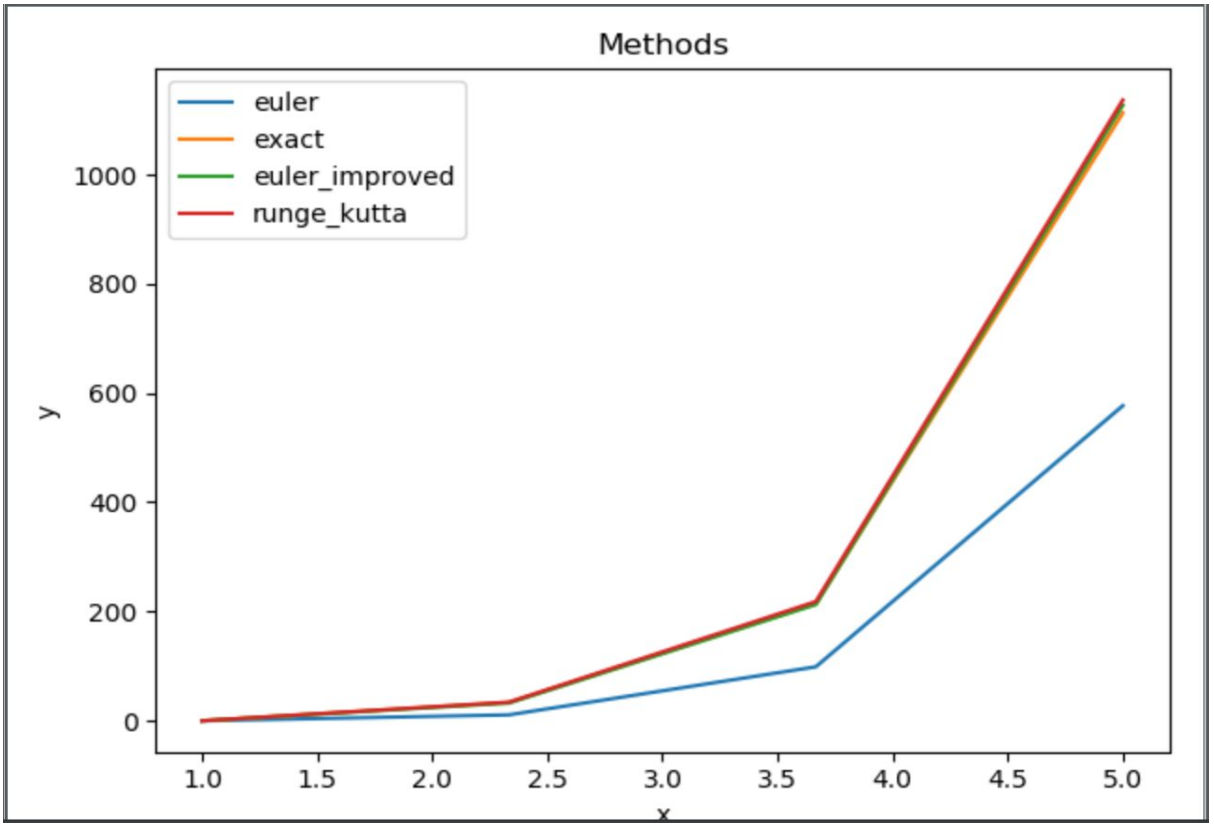
Calculating and building graph for global error

```
# accuracy
start = 20
finish = 100
# just arr for x axis in global error graph
arr = []
# global error calculating
euler_global_err, euler_improved_global_err, runge_kutta_global_err = [],[],[]
for i in range(start,finish):
    arr.append(i)
    # calculating every graph with 'i' accuracy
    x_euler, y_euler = functions.euler(i)
    x_exact, y_exact = functions.exact(i)
    x_euler_improved, y_euler_improved = functions.euler_improved(i)
    x_runge_kutta, y_runge_kutta = functions.runge_kutta(i)
    # calculating global error
    euler_max_err, euler_improved_max_err, runge_kutta_max_err = 0,0,0
    for j in range(i):
        if fabs(y_exact[j] - y_euler[j]) > euler_max_err:
            euler_max_err = fabs(y_exact[j] - y_euler[j])
        if fabs(y_exact[j] - y_euler_improved[j]) > euler_improved_max_err:
            euler_improved_max_err = fabs(y_exact[j] - y_euler_improved[j])
        if fabs(y_exact[j] - y_runge_kutta[j]) > runge_kutta_max_err:
            runge_kutta_max_err = fabs(y_exact[j] - y_runge_kutta[j])
    euler_global_err.append(euler_max_err)
    euler_improved_global_err.append(euler_improved_max_err)
    runge_kutta_global_err.append(runge_kutta_max_err)
```

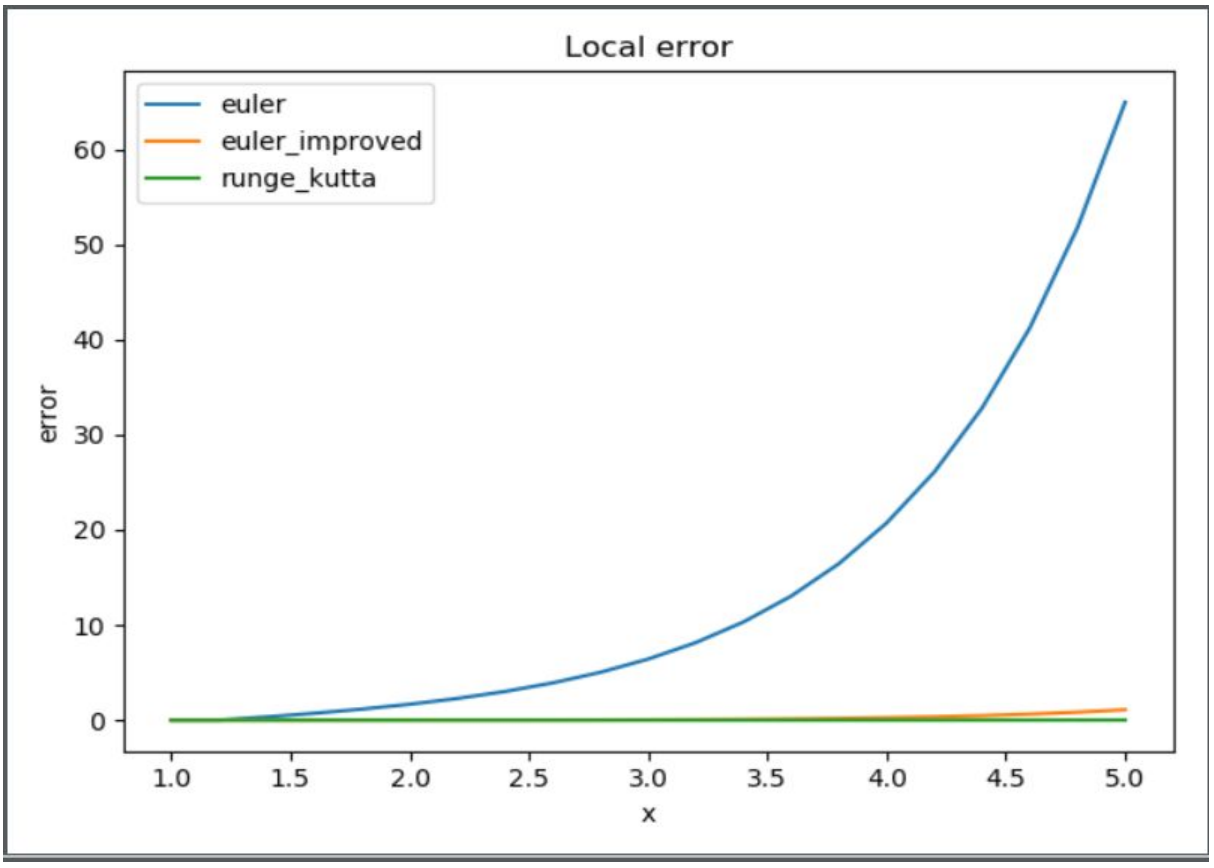
```
# here global error graph is plotted

plt.title("Global error")
plt.plot(arr, euler_global_err, label="euler")
plt.plot(arr, euler_improved_global_err, label="euler_improved")
plt.plot(arr, runge_kutta_global_err, label="runge_kutta")
plt.ylabel("error")
plt.xlabel("N")
plt.legend()
plt.show()
```

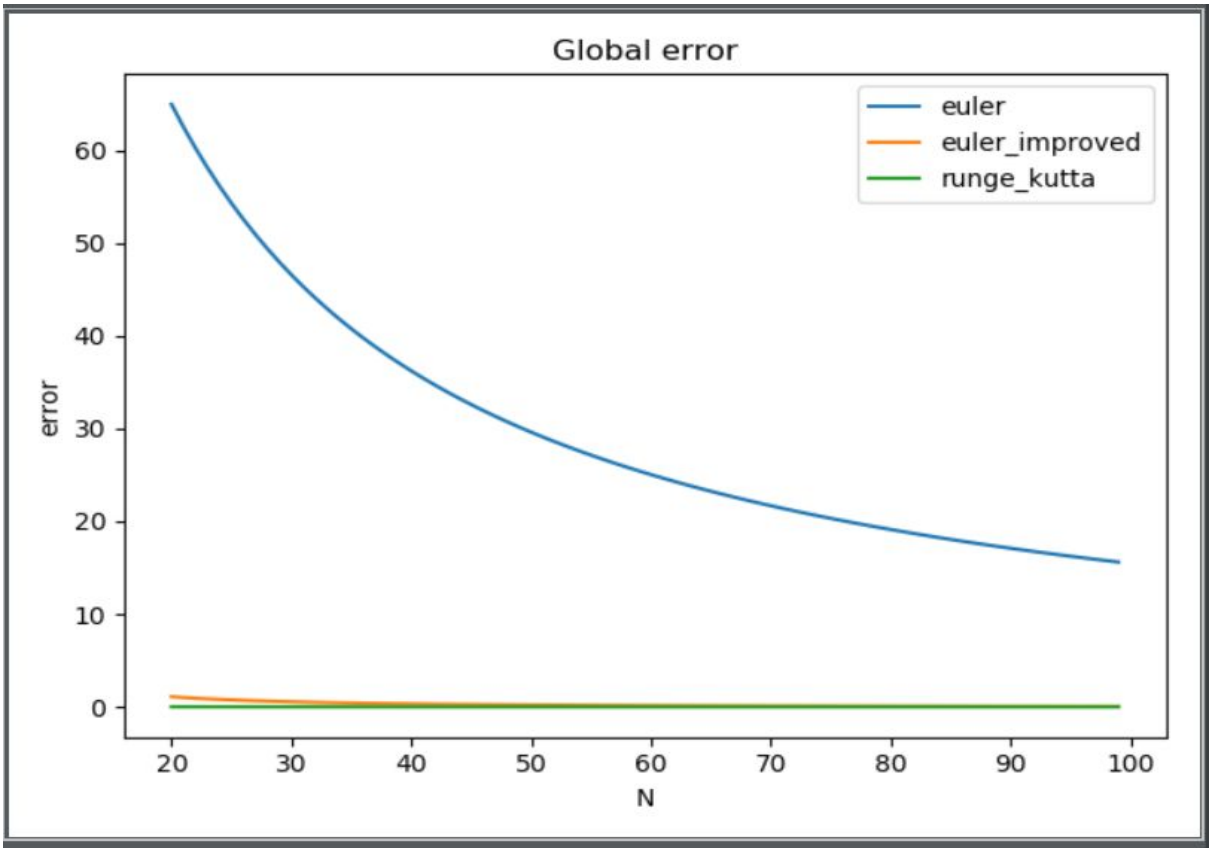
Graphs



Here amount of steps = 3, because if it is bigger, than all the graphs, except euler are merged in one line and difference can't be seen



Local error, as it should, increasing



and global is decreasing with increasing accuracy