Differential Equation: Computational Practicum

Gabdrahimova Lilya
BS17-08
Variant 7

Exact Solution

Дано уравнение:

$$\frac{d}{dx}\,y(x) = -x + \frac{1}{x}\,(2x+1)\,y(x)$$

Это дифф. уравнение имеет вид:
y' + P(x)y = Q(x)

где

$$P(x) = -\frac{1}{x}\,(2x+1)$$

и

$$Q(x) = -x$$

и называется линейным неоднородным
дифф. уравнением 1го порядка:
Решим сначала надо соответствующее линейное однородное ур-ние
y' + P(x)y = 0

с разделяющимися переменными
Данное ур-ние решается следущими шагами:
Из y' + P(x)y = 0 получаем

$$\frac{dy}{y} = -P(x)dx$$

, при y не равным 0

$$\int \frac{1}{y}\,dy = -\int P(x)\,dx$$

$$\log(|y|) = -\int P(x)\,dx$$

Или,

$$|y| = e^{\int P(x)\,dx}$$

Поэтому,

$$y_1 = e^{\int P(x)\,dx}$$

$$y_2 = -e^{-\int P(x)\,dx}$$

Из выражения видно, что надо найти интеграл:

$$\int P(x)\,dx$$

Т.к.

$$P(x) = -\frac{1}{x}\,(2x+1)$$

, то

$$\int P(x)\,dx = \int -\frac{1}{x}\,(2x+1)\,dx =$$

$$= -2x - \log(x) + Const$$

Зн., решение однородного линейного ур-ния:

$$y_1 = xe^{C_1+2x}$$

$$y_2 = -xe^{C_2 + 2x}$$

что соотв. решению
с любой константой C, не равной нулю:

$$y = Cxe^{2x}$$

Мы нашли решение соотв. однородного ур-ния
Теперь надо решить наше неоднородное уравнение
y' + P(x)y = Q(x)

Используем метод вариации произвольной постоянной
Теперь, считаем, что C - это функция от x

$$y = xC(x)e^{2x}$$

И подставим в исходное уравнение.
Воспользовавшись правилами
- дифференцирования произведения;
- производной сложной функции,
находим, что

$$\frac{d}{dx} C(x) = Q(x)e^{\int P(x)\, dx}$$

Подставим Q(x) и P(x) в это уравнение.
Получим простейшее дифф. ур-ние для C(x):

$$\frac{d}{dx} C(x) = -e^{-2x}$$

Зн., C(x) =

$$\int -e^{-2x}\, dx = \frac{1}{2}\, e^{-2x} + Const$$

подставим C(x) в

$$y = xC(x)e^{2x}$$

и получим окончательный ответ для y(x):

$$y(x) = xe^{2x}\left(Const + \frac{1}{2}\, e^{-2x}\right)$$

$$y(x) = x\left(C_1 e^{2x} + \frac{1}{2}\right)$$

x0=1 y0=3

$$3 = C_1 e^{2} + \frac{1}{2}$$

$$C = \frac{3 - 0.5}{e^2} = 2.5e^{-2}$$

$$y(x) = x\left(2.5e^{-2}e^{2x} + \frac{1}{2}\right) = x\left(2.5e^{2x-2} + 0.5\right)$$

Structure of the Program (1)

What? **I have used separate method to represent my variant’s function**

Why? **Not to clutter up my code**

```python
'''
The function from my variant №7
x: value of variable x
y: value of variable y
'''
def funct(x, y):
    return -x + (y * (2 * x + 1)) / x
```

Euler's Method

Code segment of the Euler's method for numerically approximating the solution of a first-order initial value problem

$$y' = f(x, y), \; y(x_0) = y_0$$

$$y_{j+1} := \Delta x f(x_j, y_j) + y_j$$

```
'''
Euler's Method
Variables x0, y0, xf (in moodle's document X) — from the document with requirements
n: number of steps
Return: arrays of x and y variables
'''
def euler(x0, y0, xf, n):
    deltax = (xf - x0) / (n - 1)
    x = np.linspace(x0, xf, n)
    y = np.zeros([n])
    y[0] = y0
    for i in range(1, n):
        y[i] = deltax * funct(x[i - 1], y[i - 1]) + y[i - 1]
    return x, y
```

The table of values:

x	y
x_0	y_0
x_1	y_1
$\vdots$	$\vdots$
x_n	y_n

1. x is array with:
   x0, n-1 points between, xf (X).
   Distance: xi-1-xi = xi-xi+1

2. deltax is distance xi-1-xi

3. y is array, first element is y0

4. xf is maximum value of x n is number of steps

Improved Euler’s Method

This is the iteration formula for the Improved Euler Method, also known as Heun's method. It looks a bit complicated. We would actually compute it in three steps:

1. $m_1 = f(x_j, y_j)$
2. $m_2 = f(x_{j+1}, y_j + hm_1)$
3. $y_{j+1} = y_j + h(m_1 + m_2)/2$

```
'''
Improved Euler's Method
Variables x0, y0, xf (in moodle's document X), n -- from the document with requirements
n: number of steps
Return: arrays of x and y variables
'''
def imp_euler(x0, y0, xf, n):
    deltax = (xf - x0) / (n - 1)
    x = np.linspace(x0, xf, n)
    y = np.zeros([n])
    y[0] = y0
    for i in range(1, n):
        m1 = f(x[i], y[i])
        m2 = f(x[i+1], y[i] + deltax * m1)
        y[i] = y[i - 1] + (deltax / 2) * (m1 + m2)
    return x, y
```

Same actions but other formula for calculating next y

Runge-Kutta Method

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

$y_{n+1} = y_n + (1/6)(del1 + 2del2 + 2del3 + del4)$

where:

$del1 = hf(x_n, y_n)$

$del2 = hf(x_n + h/2, y_n + k1/2)$

$del3 = hf(x_n + h/2, y_n + k2/2)$

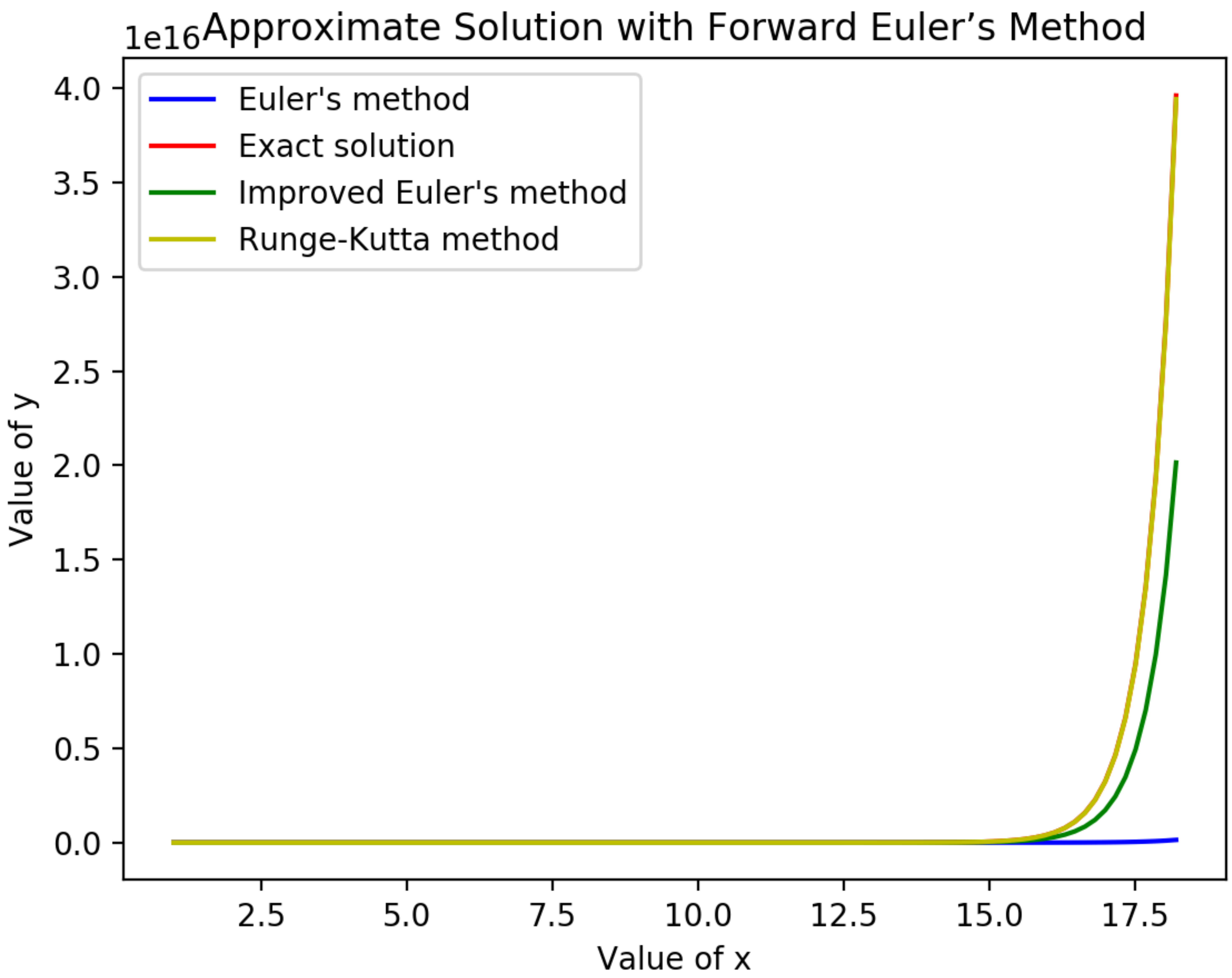
$del4 = hf(x_n + h, y_n + k4)$

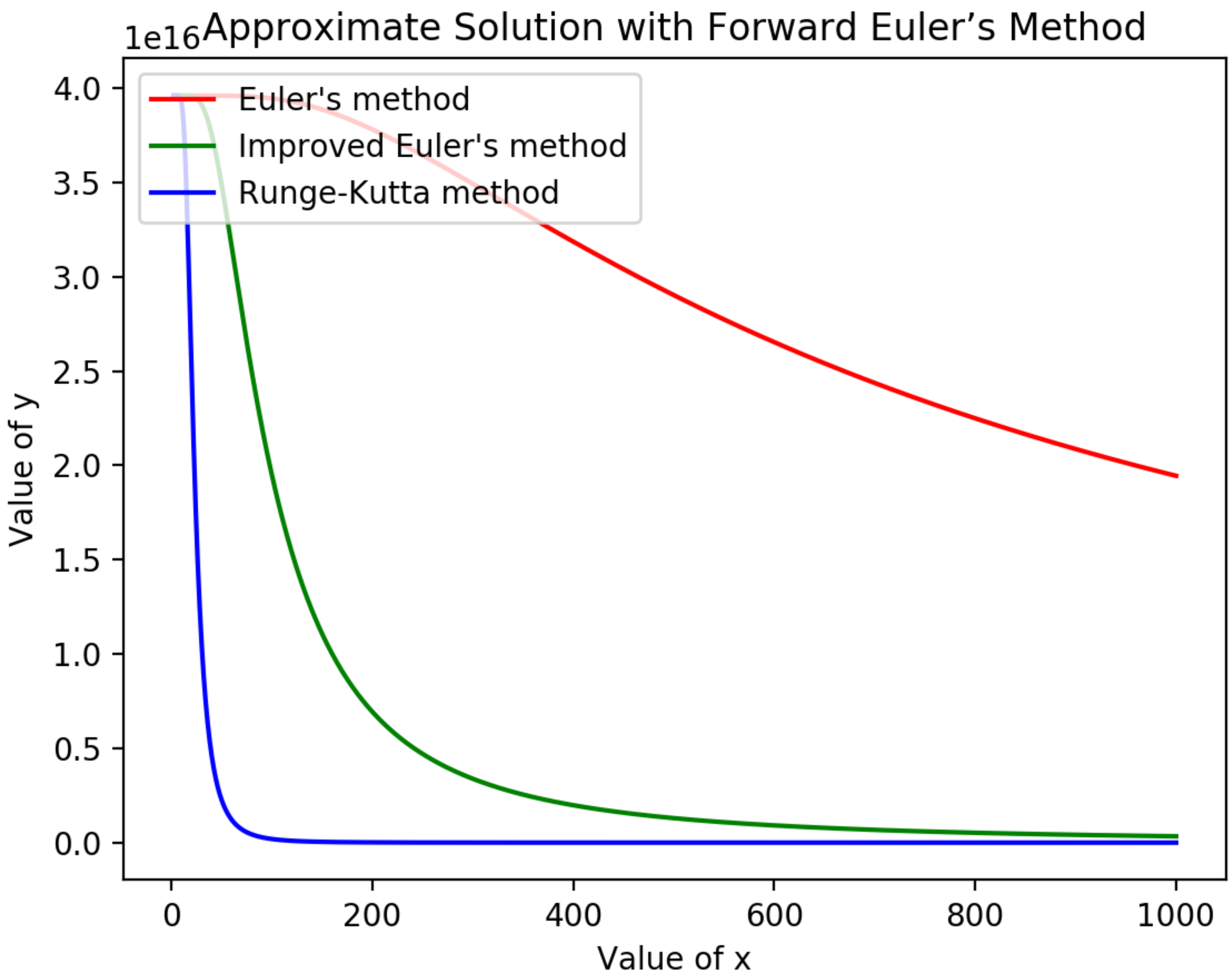
```python
'''
Runge-Kutta Method
Variables x0, y0, xf (in moodle's document X) -- from the document with requirements
n: number of steps
Return: arrays of x and y variables
'''
def runge_kutta(x0, y0, xf, n):
    deltax = (xf - x0) / (n - 1)
    x = np.linspace(x0, xf, n)
    y = np.zeros([n])
    y[0] = y0
    for i in range(1, n):
        del1 = deltax * funct(x[i - 1], y[i - 1])
        del2 = deltax * funct(x[i - 1] + 0.5 * deltax, y[i - 1] + 0.5 * del1)
        del3 = deltax * funct(x[i - 1] + 0.5 * deltax, y[i - 1] + 0.5 * del2)
        del4 = deltax * funct(x[i - 1] + deltax, y[i - 1] + del3)
        y[i] = y[i - 1] + 1 / 6 * (del1 + 2 * del2 + 2 * del3 + del4)

    return x, y
```

Same actions but other formula for calculating next y

Let’s run it

```
/Users/lilyarmand/PycharmProjects/de_assignment/venv/bin/python /Users/lilyarmand/PycharmProjects/de_assignment/main.py
x0 = 1
y0 = 3
xf = 18.2
№ of steps = 100
min № of steps = 2
max № of steps = 1000
```


Approximate Solution with Forward Euler's Method
1e16
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Value of y
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Value of x
Euler's method
Exact solution
Improved Euler's method
Runge-Kutta method



Approximate Solution with Forward Euler's Method
1e16
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Value of y
0
200
400
600
800
1000
Value of x
Euler's method
Improved Euler's method
Runge-Kutta method

GitHub Repository

Link