Jafar Badour

Differential Equations

Programming assignment: variant 18

Oct 29 – 2018

https://github.com/JafarBadour/Differential_Equations_Computational_Practicum

Exact Solution:

The differential equation is of the form

$y' = f(x, y)$ where $f(x, y) = 2\sqrt{y} + 2y$

Lets substitute $z = \sqrt{y}$ thus,

$$2zdz = dy$$

$2zdz = (2z + 2z^2)dx$ assuming that $z \neq 0$

$$\frac{2dz}{2 + 2z} = dx$$

$$\ln(1 + z) = x + c$$

$$|1 + z| = e^{x+c}$$

Note that here $z \geq 0$, thus $1 + z$ is always positive thus,

$$1 + z = e^{x+c}$$

$$z = (De^x - 1): D \in]0, \infty[$$

$$y = (De^x - 1)^2$$

Now for $y(0) = 1$, $1 = (D - 1)^2$,

Here D has two solutions $|D - 1| = 1 \rightarrow D - 1 = \pm 1$

$$D = 0,2$$

Well $D = 0$ is not accepted because it violates the range of D

Thus

$$D = 2$$

The exact solution when y(0) = 1

$$y = (2e^x - 1)^2$$

Structure of the program

```
class DrawOpt:
    def __init__(self):
        self.option = ""
        self.num_seg = 10
        self.INITIAL_X = 0
        self.INITIAL_Y = 1

    def update_option(self, event):
        l = event.widget
        sel = l.curselection()
        self.option = l.get(sel[0])

    def update_num_seg(self, event):
        self.num_seg = int(event)
```

this class is for xml file and UI.

We have getD which is to get the constant after integration from the initial conditions.

```
def exact_sol(X, opt):
    D = getD(opt)
    return np.square(1-D*np.exp(X))
```

this method will get us the exact solution of the initial value problem.

```
def f(x, y):
    return 2 * math.sqrt(y) + 2 * y
```

the function f(x,y) where y’ = f(x,y)

```
def approx_method(opt):
```

To chose which approximation method to use.

```
def get_name_of_method(opt):
```

to transform the string of the name of the methods to numbers for easier manipulations.

```
def Eulers(opt):
```

Gives us the approximate solution using euler method.

```
def ImprovedEulers(opt):
```

Gives us the approximate solution of the improved euler method which is basically the same but adding to the loop in line 105 k2,k1 and take the average.

```
def RungeKutta(opt):
```

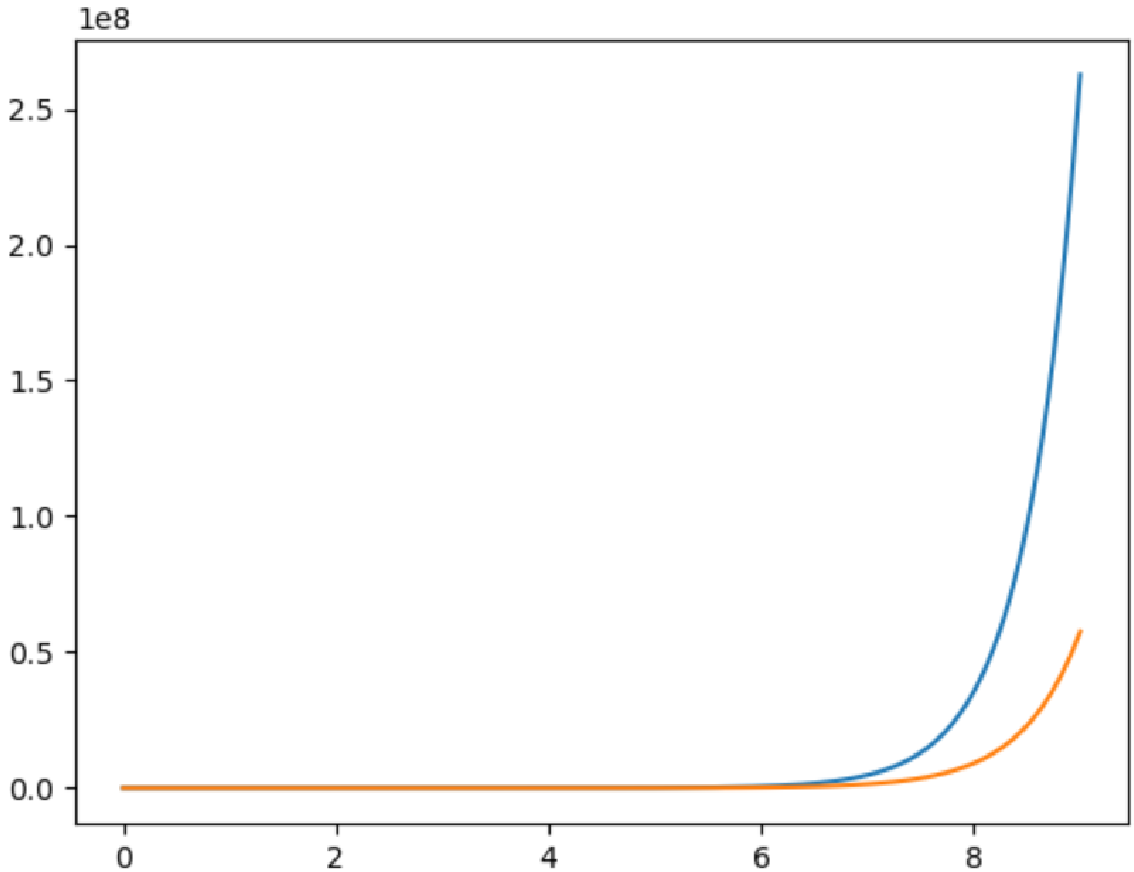
Gives us the approximate solution according to RungeKutta method,

Note after each method we transform the array into numpy array for plotting and easier for manipulation.

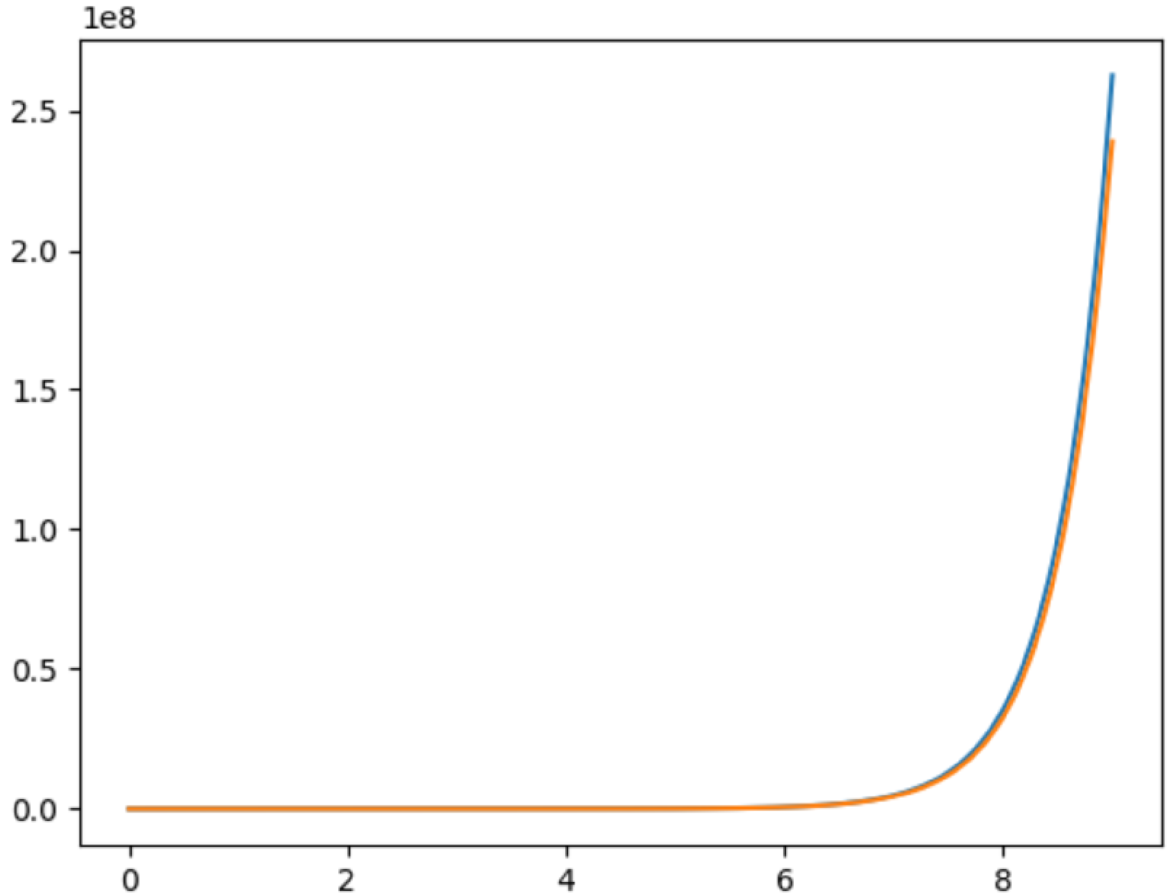
Graphics

For x = 0, y = 1 and steps = 100

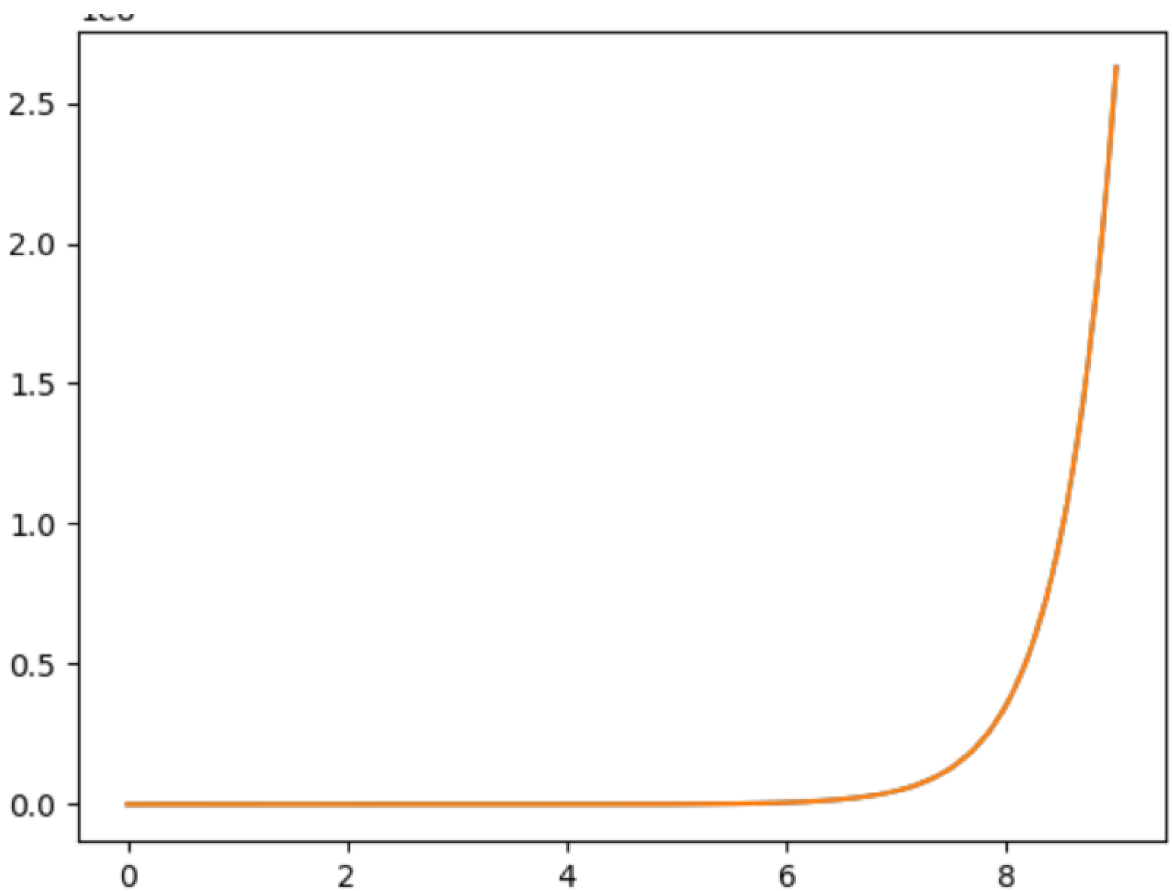
Euler method, Exact Solution



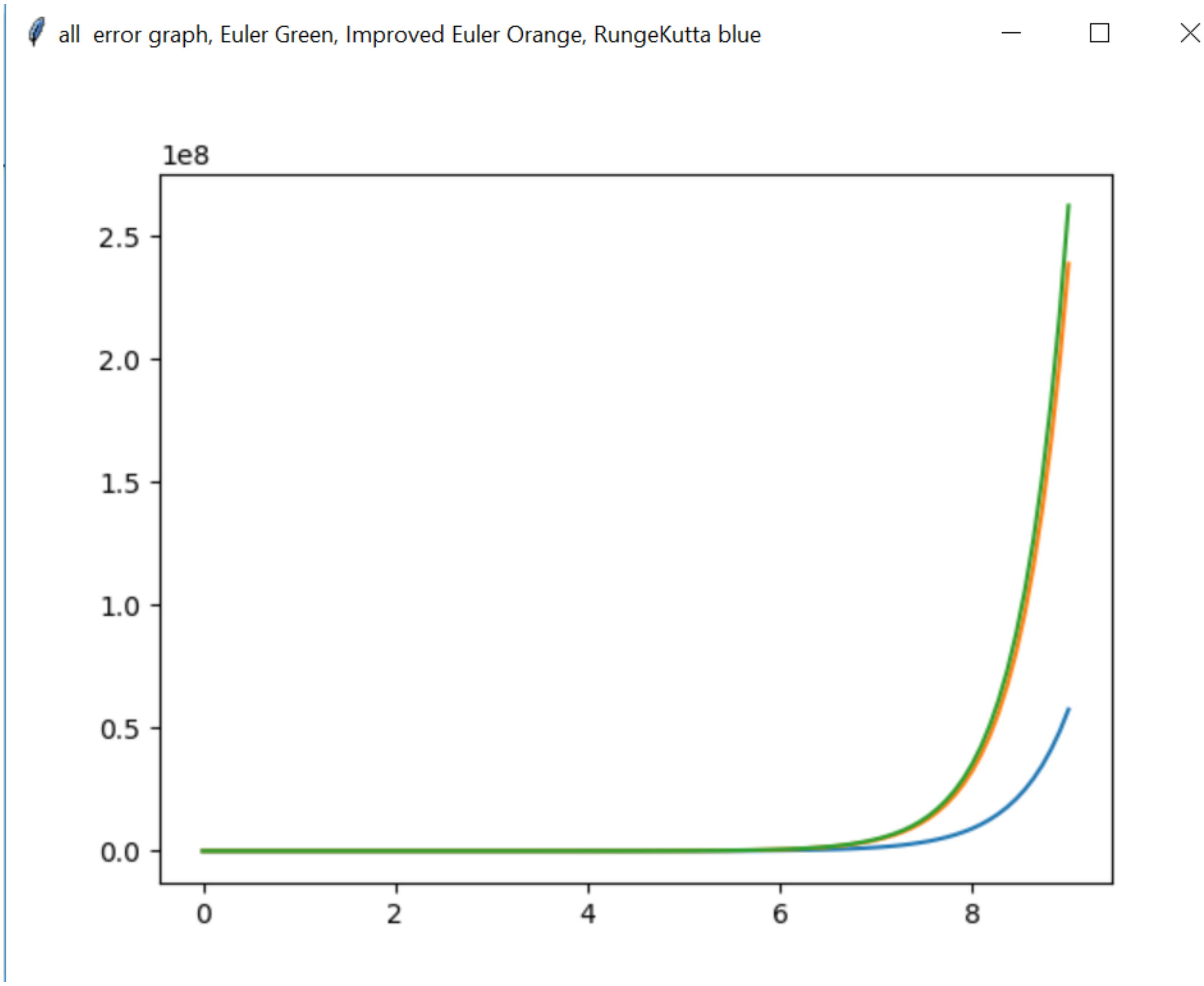
Improved Euler method, Exact Solution



RungeKutta method, Exact Solution



All errors of these methods can be viewed in one graph



We Notice that RungeKutta is the best method to calculate the solution and improved euler come in the second place. And the worst is euler method.

Errors as a number of steps: