Graphical User Interface (GUI) for visualization of solutions of initial value problems using Runge-Kutta methods

Folaranmi Akinola Davidson

davidsonfolaranmi@gmail.com

https://github.com/F-Akinola/runge-kutta

Abstract In this paper, a successful approach for the visualization of the solutions of initial value problems using different Runge-Kutta methods by developing a new graphical user interface (GUI), has been introduced. The main objective of the paper is to apply the tkinter, numpy[1] and matplotlib[2] modules of the Python programming language in developing a GUI that would enable the user to appreciate the accuracy of each Runge-Kutta method at different step size/increment. The second purpose of this paper is to develop a python module that contains functions that can be imported to .py files when the need to implement Runge-Kutta methods on initial value problems arises. The GUI was used to solve and plot the solutions of an initial value problem. The variations in the graph patterns illustrate the accuracy of the output of each method.

Keywords GUI, Runge-Kutta methods

1. Introduction

Euler and Heun methods

In numerical analysis, the Runge-Kutta methods are a family of iterative methods, which includes the well-known routine called the Euler method, used in temporal discretization for the approximate solution of ordinary differential equations. These methods were developed by the German mathematicians Carl Runge and Wilhelm Kutta[3].

Euler's method is one possible approach for solving differential equations with a given initial value numerically. Euler's method assumes the solution is written in the form of a Taylor's series. For Euler's method, we just take the first two terms only.

$$y(x+h) \approx y(x) + hy'(x) \qquad (1)$$

Euler's method is useful because differential equations appear frequently in physics, chemistry, and economics, but usually cannot be solved explicitly, requiring their solutions to be approximated. The problem with Euler's method is that you have to use a small interval size to get a reasonably accurate result.

The Heun's method is an improvement over the rather simple Euler's method. Although the method uses Euler's method as a basis, it attempts to compensate for the Euler's method failure to take the curvature of the solution curve into account. Heun's method is one of the simplest of a class of methods called the predictor-corrector algorithms.

2. Runge-kutta methods

The table below gives the formulas for calculating values of K_n (n = 1, 2, 3, 4, 5, 6) as well as the new value y after each iteration for different Runge-Kutta methods. [4]

RK method	K_n (n = 1 , 2, 3, 4, 5, 6)	y1

RK2A	k1 = f(x,y)
	
	k2 = f(x + h, y + (h x k1))
		y1 = y + ((h / 2) x (k1 + k2))

RK2B	k1 = f(x,y) k2 = f(x + (0.5 x h), y + (0.5 x h x k1))	y1 = y + (h x k2)
RK2C	k1 = f(x,y) k2 = f(x + ((3 / 4) x h), y + ((3 / 4) x h x k1))	y1 = y + ((h / 3) x (k1 + (2 x k2)))

RK3	k1 = f(x,y)
	
	k2 = f(x + (h / 2), y + ((h x k1) / 2))
	
	k3 = f(x + h, y - (h x k1) + (2 x h x k2))
		y1 = y + ((h / 6) x (k1 + (4 x k2) + k3))

RK4	k1 = h x f(x, y)
	
	k2 = h x f(x + (0.5 x h), y + (0.5 x k1))
	
	k3 = h x f(x + (0.5 x h), y + (0.5 x k2))
	
	k4 = h x f(x + h, y + k3)
		y1 = y + ((1.0/6.0) x (k1 + (2 x k2) + (2 x k3) + k4))

RK5 (Butcher's fifth-order Runge-Kutta method)	k1 = f(x, y)
	
	k2 = f(x + (h / 4), y + (h x k1 / 4))
	
	k3 = f(x + (h / 4), y + (h x k1 / 8) + (h x k2 / 8))
	
	k4 = f(x + (h / 2), y - (h x k2 / 2) + (h x k3))
	
	k5 = f(x + (3 x h / 4), y + (3 x h x k1 /16) + (9 x h x k4 / 16))
	
	k6 = f(x + h, y - (3 x h x k1 / 7) + (2 x h x k2 / 7) + (12 x h x k3 / 7) - (12 x h x k4 / 7) + (8 x h x k5 / 7))
		y1 = y + ((h / 90) x ((7 x k1) + (32 x k3) + (12 x k4) + (32 x k5) + (7 x k6)))

RKF45 (Runge-Kutta-Fehlberg method)	k1 = f(x,y) k2 = f(x + (h / 5), y + (h x k1 / 5)) k3 = f(x + (3 x h / 10), y + (3 x h x k1 / 40) + (9 x h x k2 / 40)) k4 = f(x + (3 x h / 5), y + (3 x h x k1 / 10) - (9 x h x k2 / 10) + (6 x h x k3 / 5)) k5 = f(x + h, y - (11 x h x k1 /54) + (5 x h x k2 / 2) - (70 x h x k3 / 27) + (35 x h x k4 / 27))	y1 = y + (h x ((37 x k1 / 378) + (250 x k3 / 621) + (125 x k4 / 594) + (512 x k6 / 1771)))

	k6 = f(x + (7 x h / 8), y + (1631 x h x k1 / 55296) + (175 x h x k2 / 512) + (575 x h x k3 / 13824) - (44275 x h x k4 / 110592) + (253 x h x k5 / 4096))	

3. rungeKutta module

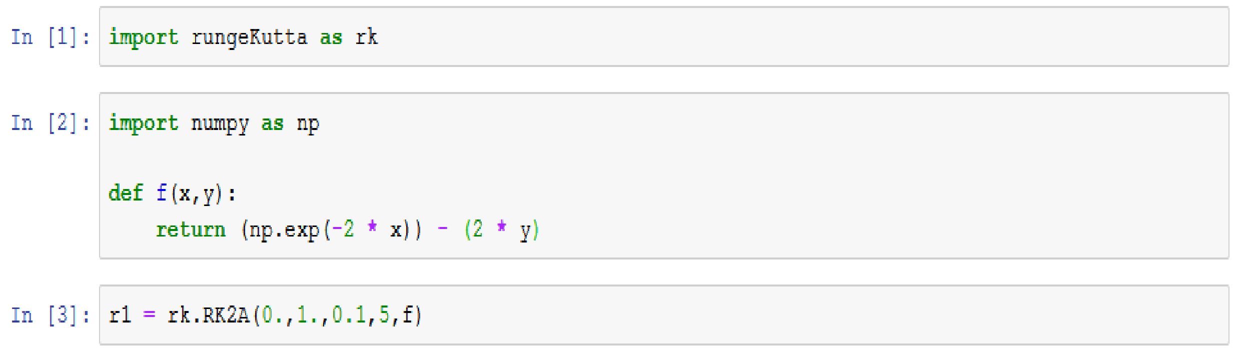
The rungeKutta module contains functions that helps to compute the solutions of differential equations using the Runge-Kutta methods. Contained in the module are the second-, third- and fourth-order methods, as well as Butcher's fifth order method and Runge-Kutta-Fehlberg method.

The functions in the module can be accessed from another .py file with the command:

import rungeKutta as rk

To utilize this module, the user has to define the differential equation to be passed as an argument to the functions. The parameters of the functions in this module are the initial values of x and y, the increment at the end of each iteration i.e. step size, the desired number of iterations and the user-defined differential equation. For example, to utilize the RK2A method, the syntax is:

rk.RK2A (initial x, initial y, step size, number of iterations, differential function)



```
In [1]: import rungeKutta as rk

In [2]: import numpy as np

        def f(x,y):
            return (np.exp(-2 * x)) - (2 * y)

In [3]: r1 = rk.RK2A(0.,1.,0.1,5,f)
```

Figure 1: Importing and utilizing the rungeKutta module

In Figure 2, the function, f(x,y) is the differential equation with variables x and y passed as the parameters.

4. rkgui.py

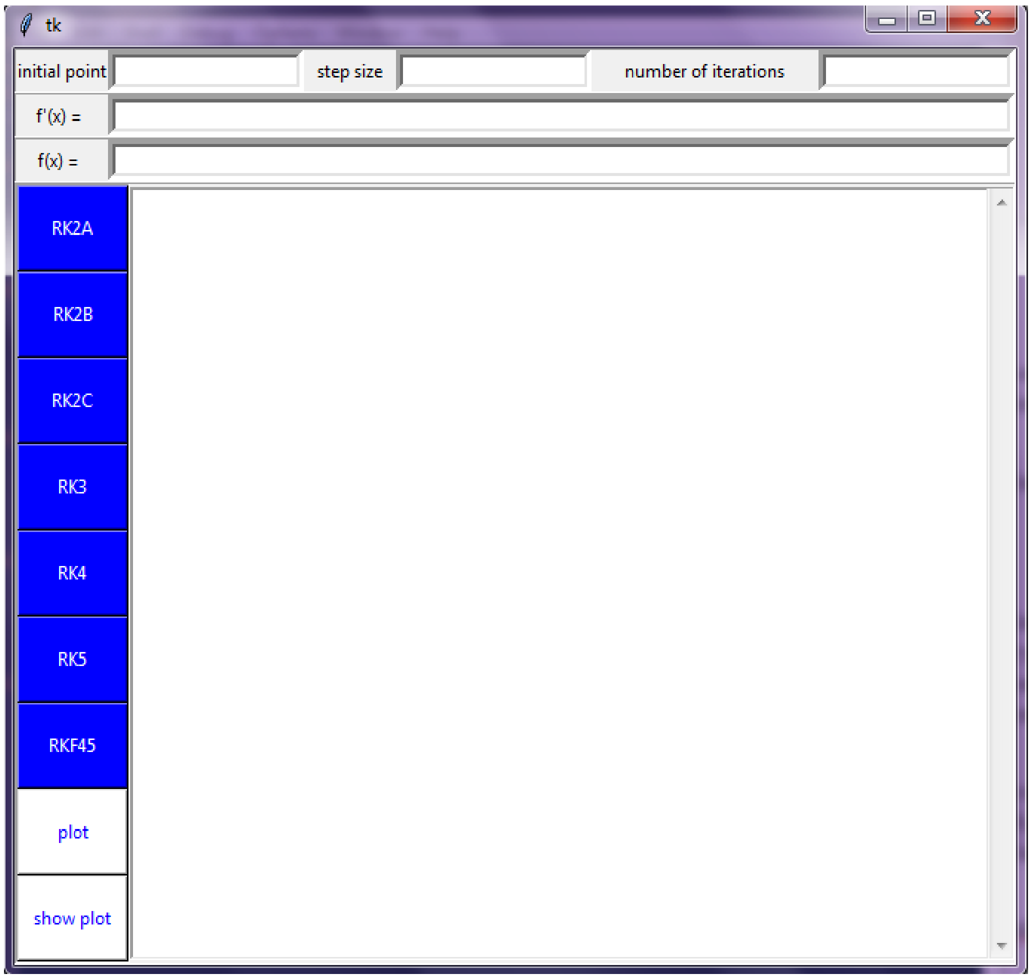
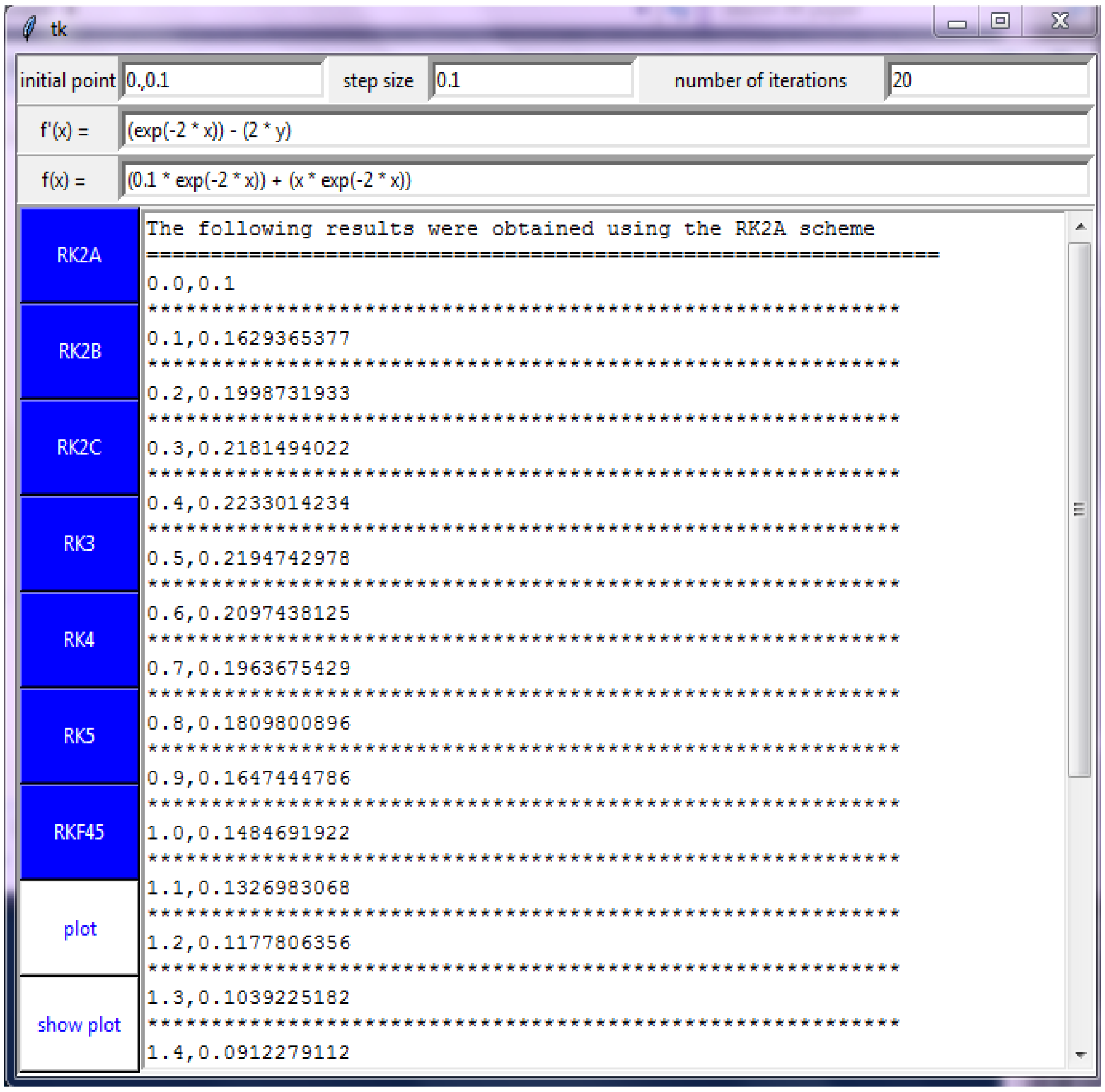


Figure 2: The GUI

The rkgui.py file imports the rungeKutta module in the design of a GUI for the visualization of results obtained using any of the Runge-Kutta schemes. It comprises entry boxes for specifying the initial values of the function we wish to evaluate, the desired increment per step, the number of steps, the first derivative of the function of interest and a final entry box for the exact function for differential equations whose exact solutions can be obtained.

The GUI also includes a set of buttons. These buttons allows users to specify the desired scheme to evaluate the differential equation typed into the appropriate text box. The plot button passes the command to the GUI to plot the x and f(x) values but does not show the plot. The show plot button displays the the plot of f(x) against x. After computing the values of the specified function over the specified number of iterations, the results are displayed on the scrolled text widget of the GUI.

The GUI utilizes the tkinter module, while the plot utilizes the matplotlib and numpy modules.



initial point 0.,0.1 step size 0.1 number of iterations 20

f'(x) = (exp(-2 * x)) - (2 * y)

f(x) = (0.1 * exp(-2 * x)) + (x * exp(-2 * x))

```
The following results were obtained using the RK2A scheme
==============================================================
0.0,0.1
***********************************************************
0.1,0.1629365377
***********************************************************
0.2,0.1998731933
***********************************************************
0.3,0.2181494022
***********************************************************
0.4,0.2233014234
***********************************************************
0.5,0.2194742978
***********************************************************
0.6,0.2097438125
***********************************************************
0.7,0.1963675429
***********************************************************
0.8,0.1809800896
***********************************************************
0.9,0.1647444786
***********************************************************
1.0,0.1484691922
***********************************************************
1.1,0.1326983068
***********************************************************
1.2,0.1177806356
***********************************************************
1.3,0.1039225182
***********************************************************
1.4,0.0912279112
```

Figure 3: The GUI with output

5. Case study and Results

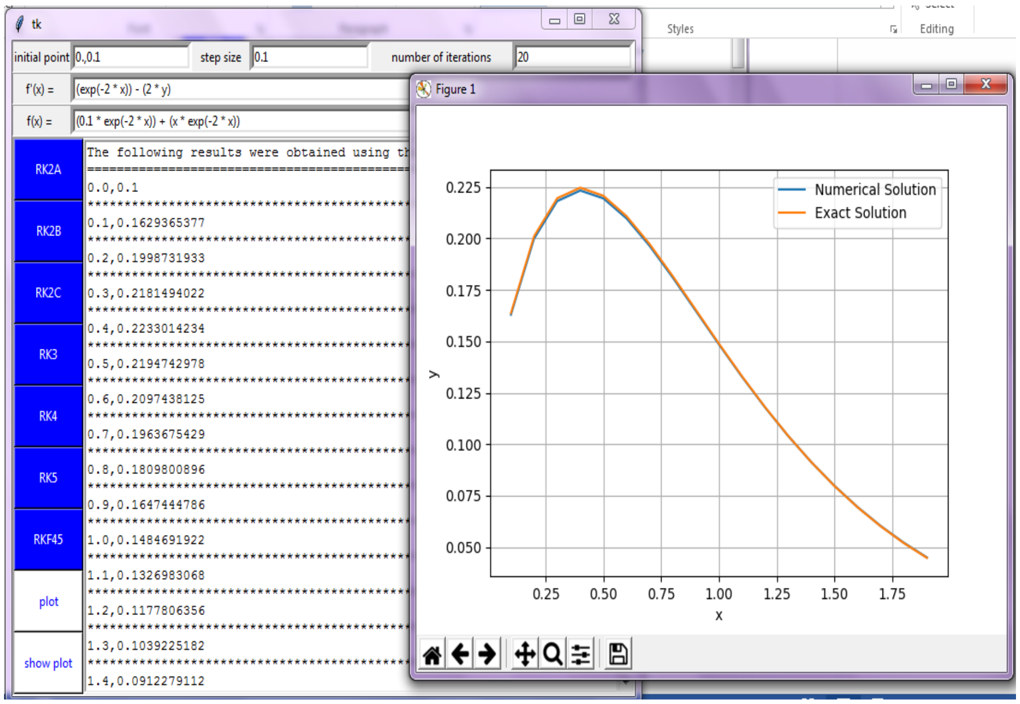
The GUI was used to solve and produce the plots of the numerical and exact solutions of the following initial value problem:

$$y' = e^{-2x} - 2y \qquad (2)$$

With y(0) = 0.1 on [0,2] and step size h = 0.2 and h = 0.1.

The exact solution is:

$$y(x) = \frac{1}{10}e^{-2x} + xe^{-2x} \qquad (3)$$



initial point: 0.,0.1
step size: 0.1
number of iterations: 20
f'(x) = (exp(-2 * x)) - (2 * y)
f(x) = (0.1 * exp(-2 * x)) + (x * exp(-2 * x))

```
The following results were obtained using th
============================================
0.0,0.1
*******************************************
0.1,0.1629365377
*******************************************
0.2,0.1998731933
*******************************************
0.3,0.2181494022
*******************************************
0.4,0.2233014234
*******************************************
0.5,0.2194742978
*******************************************
0.6,0.2097438125
*******************************************
0.7,0.1963675429
*******************************************
0.8,0.1809800896
*******************************************
0.9,0.1647444786
*******************************************
1.0,0.1484691922
*******************************************
1.1,0.1326983068
*******************************************
1.2,0.1177806356
*******************************************
1.3,0.1039225182
*******************************************
1.4,0.0912279112
```

Figure 3: After calling the plot function via the plot and the show plot buttons

h = 0.1

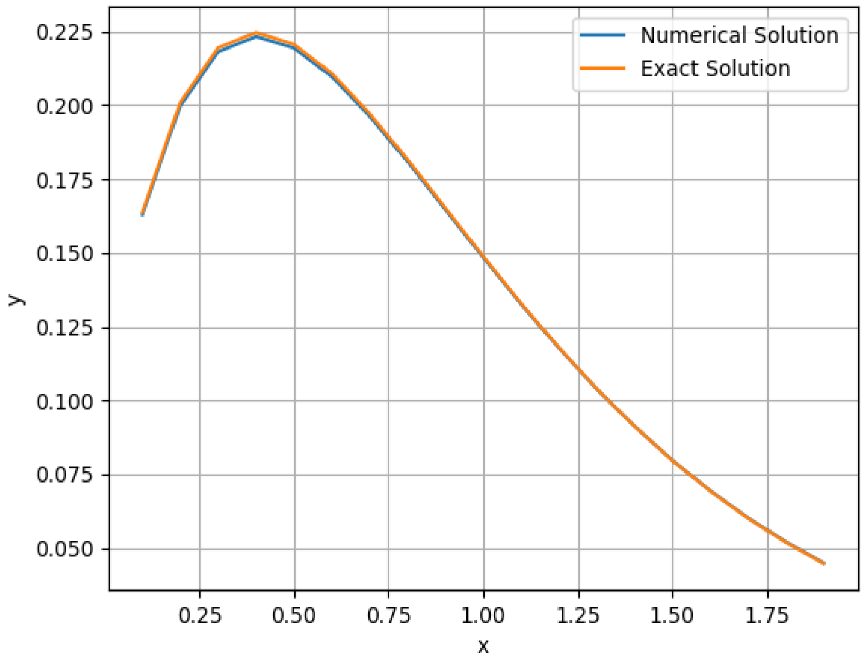


Figure 4: Second-order Runge-Kutta method RK2A

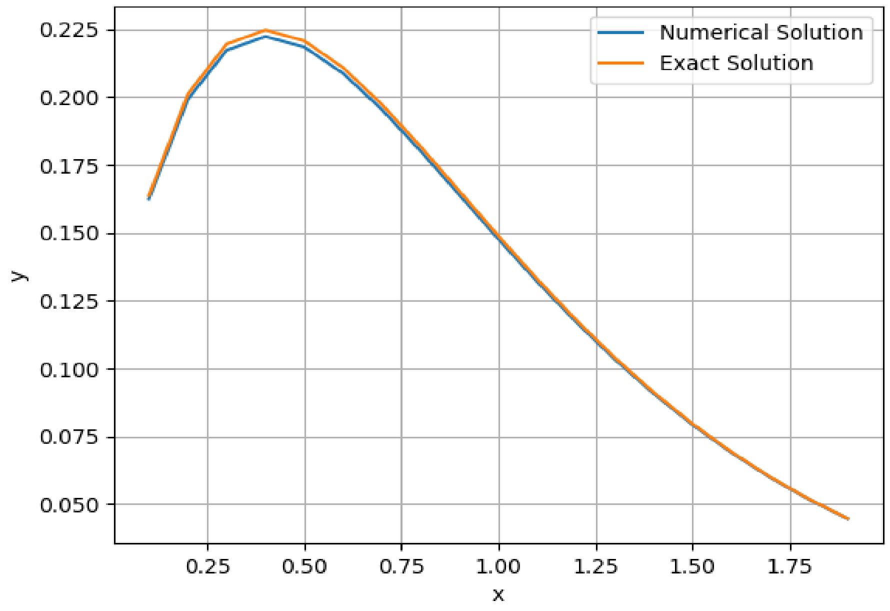


Figure 5: Second-order Runge-Kutta method RK2B

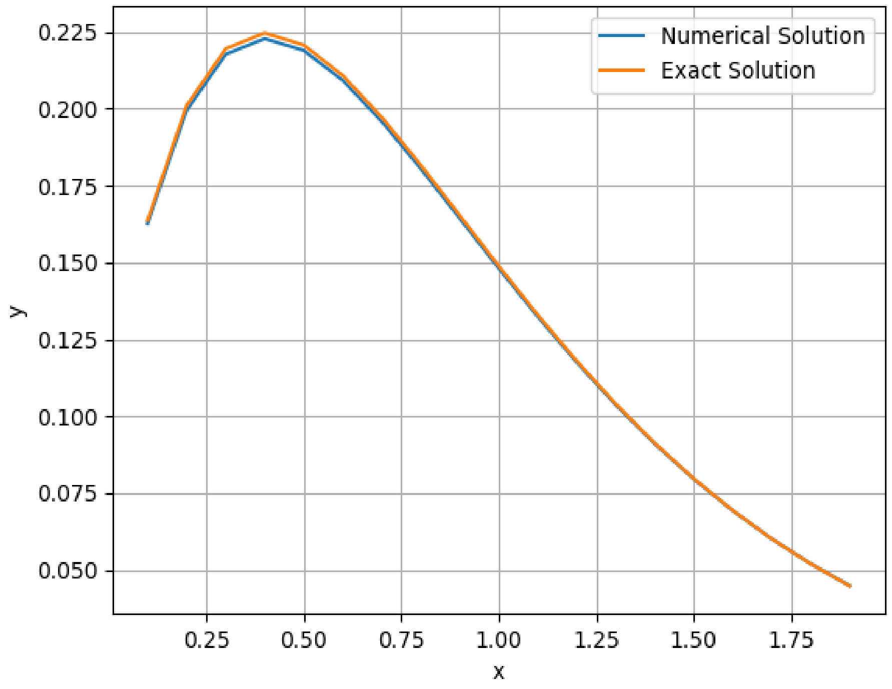


Figure 6: Second-order Runge-Kutta method RK2C

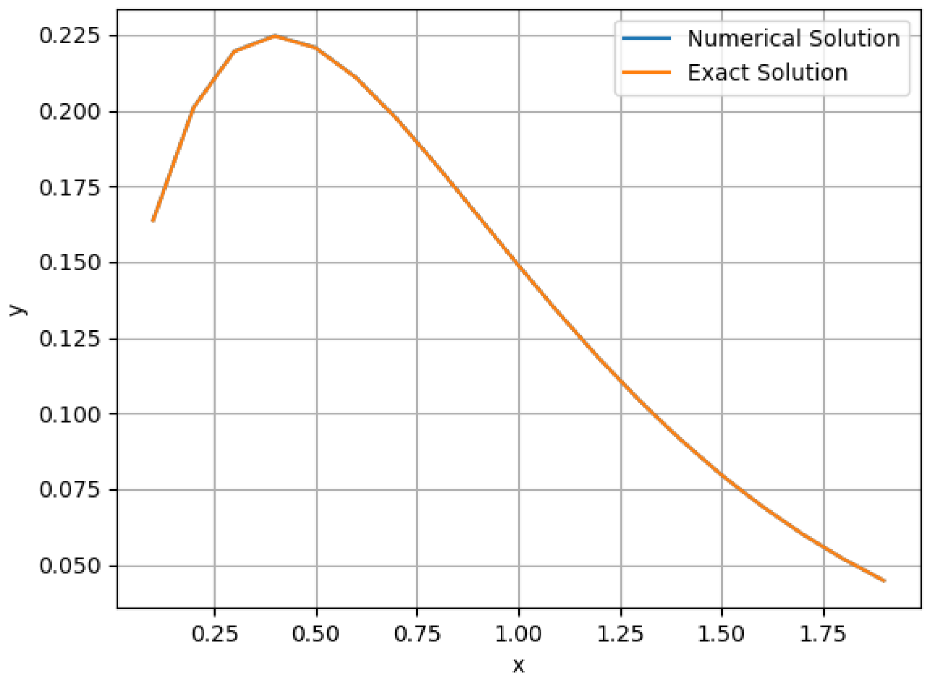


Figure 7: Third-order Runge-Kutta method RK3

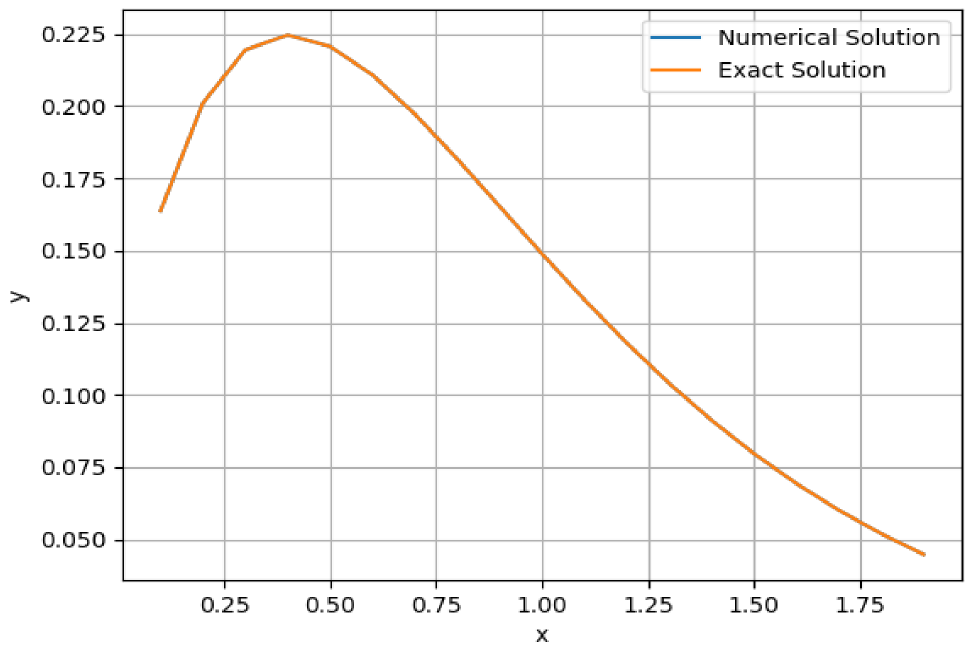


Figure 8: Fourth-order Runge-Kutta method RK4

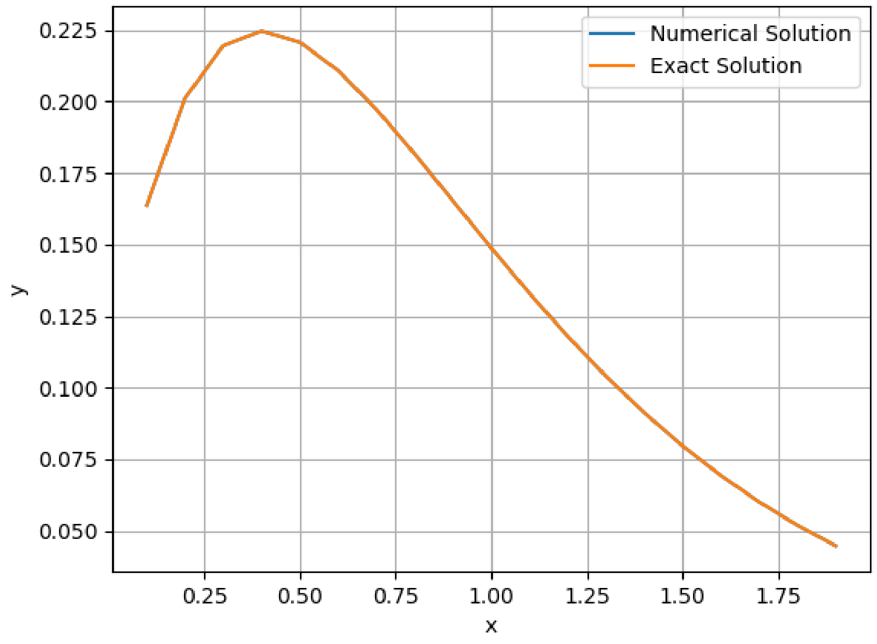


Figure 9: Butcher's fifth-order Runge-Kutta method RK5

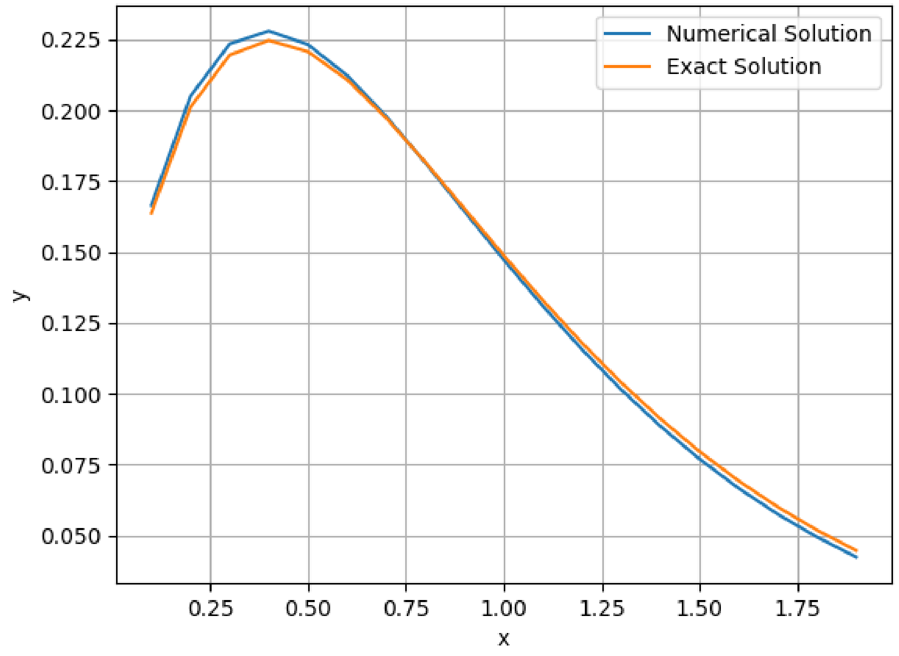


Figure 10: Runge-Kutta-Fehlberg method RKF45

h = 0.2

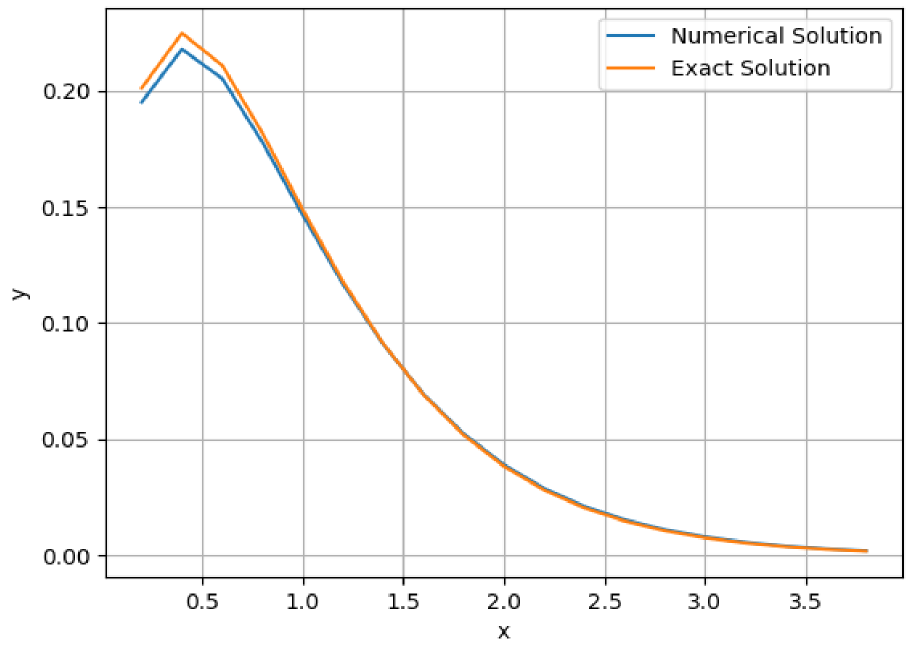


Figure 11: Second-order Runge-Kutta method RK2A

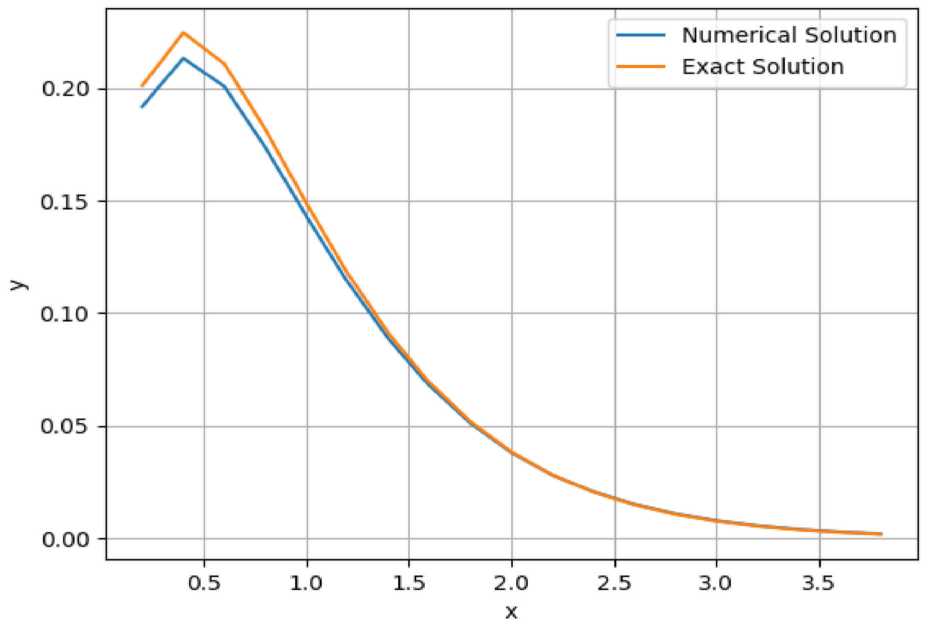


Figure 12: Second-order Runge-Kutta method RK2B

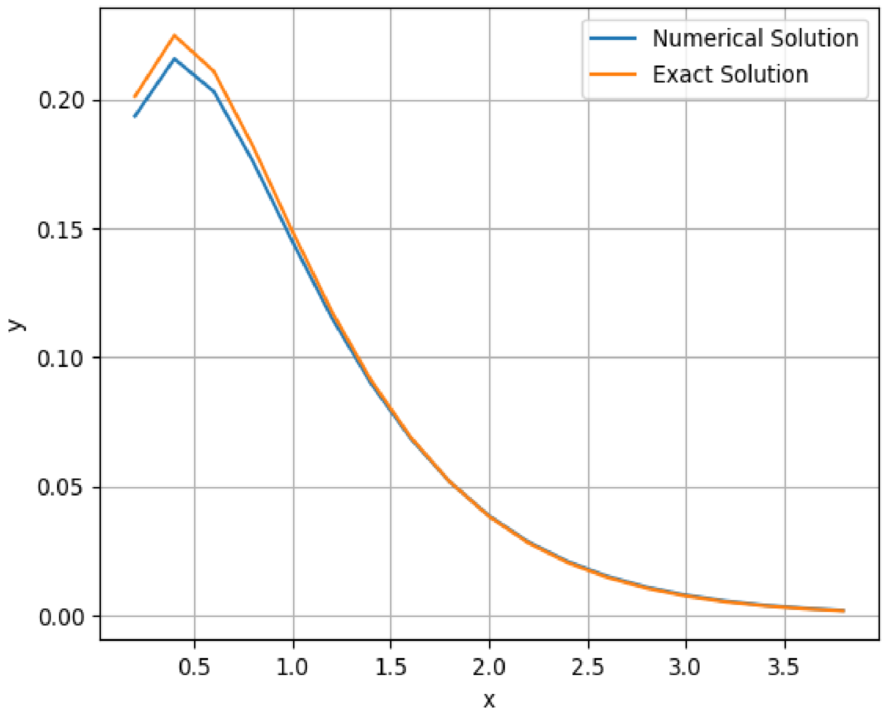


Figure 13: Second-order Runge-Kutta method RK2C

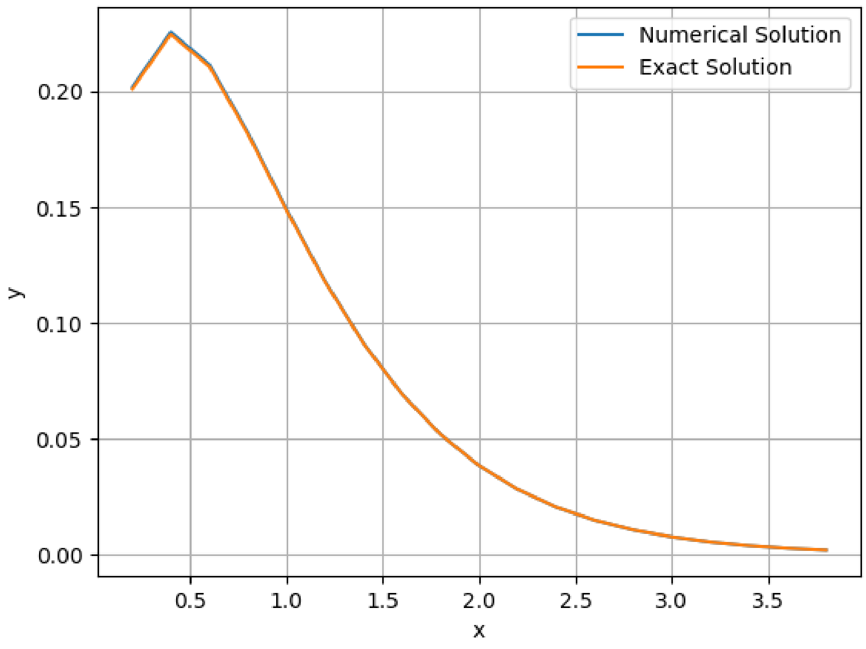


Figure 14: Third-order Runge-Kutta method RK3

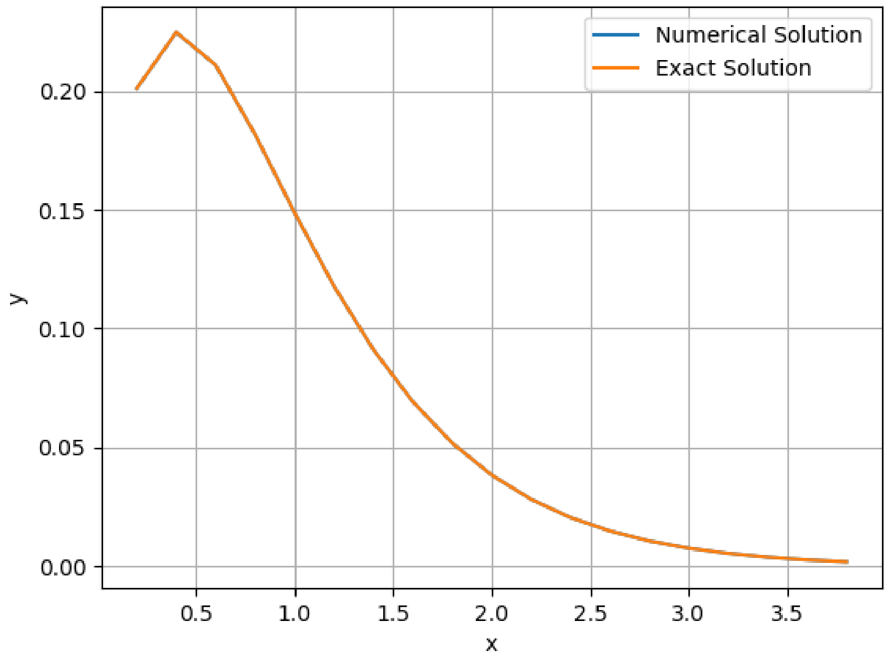


Figure 15: Fourth-order Runge-Kutta method RK4

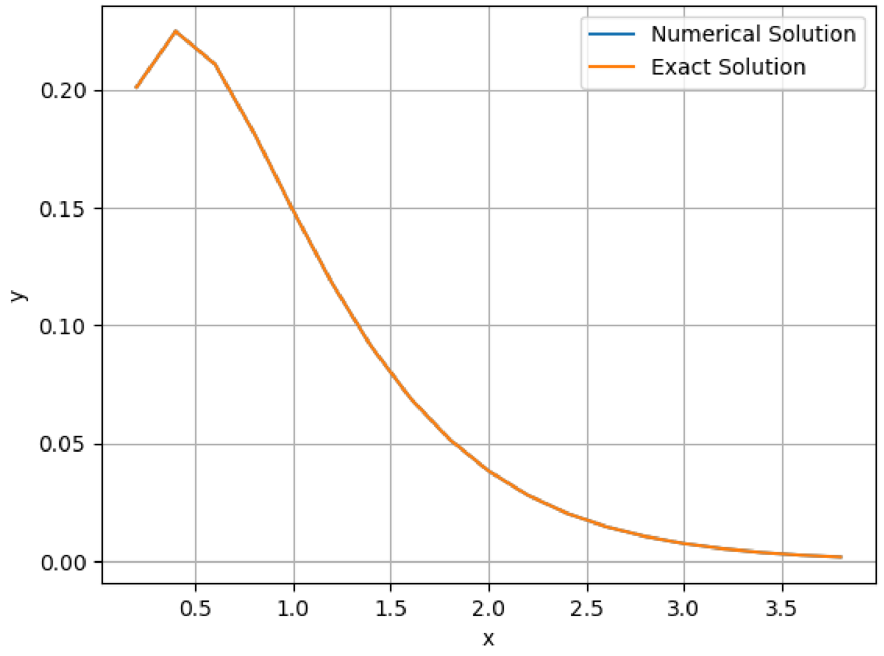


Figure 16: Butcher's fifth-order Runge-Kutta method RK5

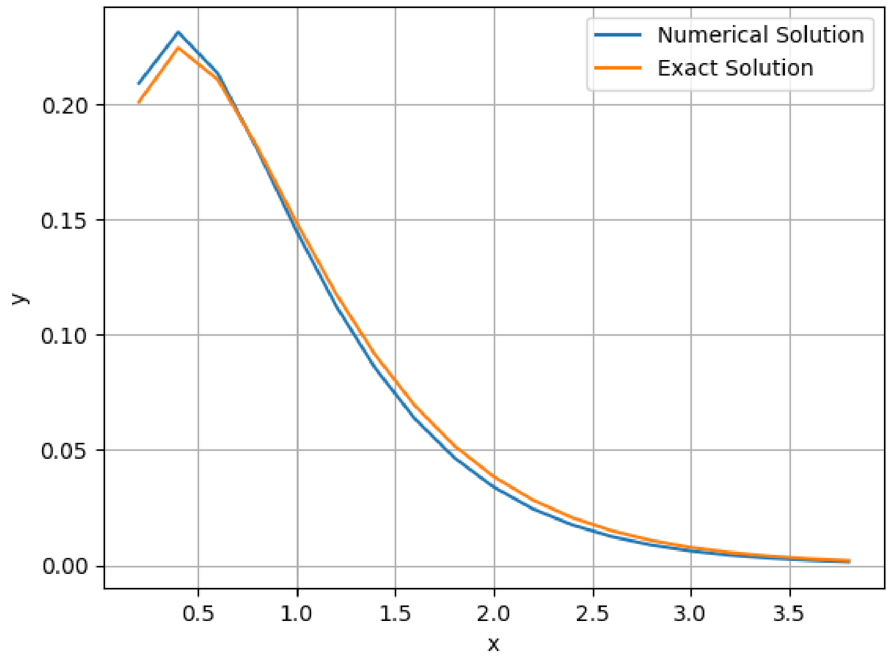


Figure 17: Runge-Kutta-Fehlberg method RKF45

6. Conclusions/Discussions

The main objective of the paper is to develop a GUI that would enable the user appreciate the accuracy of the Runge-Kutta methods. A major issue of the GUI is the need for users to supply the differential functions in terms of lowercase ‘x’ and lowercase ‘y’. This is partly as a result of the case-sensitivity of the Python programming language and partly because the source code for this portion of the GUI wasn’t written in such a way as to make it flexible enough to understand other symbolic representation besides ‘x’ and ‘y’. Added to this is the requirement that the equations must be typed using the Python programming language syntax. The GUI was applied to an initial value problem and the results obtained shows the efficiency, accuracy and reliability of the GUI, despite its simplicity and the aforementioned issues.

REFERENCES

[1] https://numpy.org

[2] https://matplotlib.org

[3] https://en.m.wikipedia.org/wiki/Runge-Kutta_methods

[4] Lecture 9,http://people.bu.edu/andasari/courses/numericalpython/python.html