**Runge-Kutta-Fehlberg (RKF) for ODE : Project 2**

CST-305: Principles of Modeling and Simulation

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

WF1100A Dr. Citro

Dayana Gonzalez Cruz

October 1st, 2023

**Runge-Kutta 4th Order**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The Runge-Kutta 4th order (RK4) is a method to solve a ordinary differential equation (ODE) given its initial values (i.e. and ) and a step size h- a value by which to increment x per y (LibreTexts, 2020). To elaborate, the mathematical algorithm involves calculating to calculate where the formulas for each calculation are as follows:

*Problem Solved :* Given the ODE , with initial values and and a step size of , a python program can be written to automate the calculation steps for a given m number of y values. Moreover, the SciPy library for python offers an ODEINT function that solves a given ordinary differential equation. The first 5 values of the given ODE were manually solved by using the RK4 method by hand and compared with computations run by a python program.

Calculations:

Below is the manual calculation for the first y, .

Below are the results of manually calculated against the computational results of the written python function RKSolve (which are labeled “True Solution”) within the source code .py file that accompanies this document. The results show that the manual calculations coincide with the computations of the program accurately.

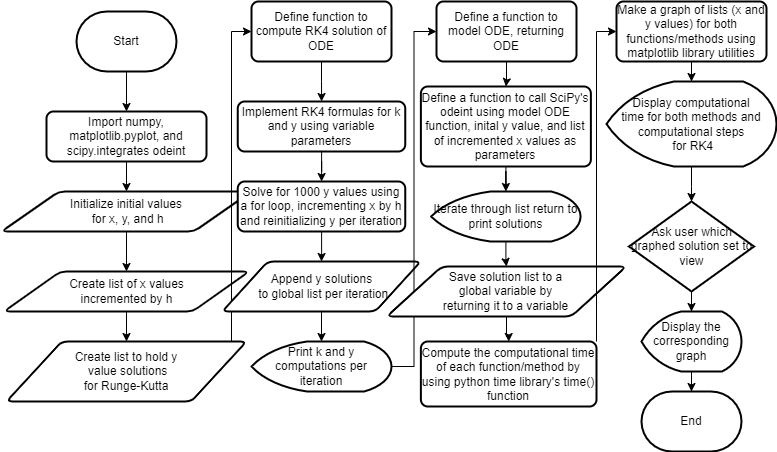
|  |  |  |  |
| --- | --- | --- | --- |
| Method: RUNGE-KUTTA METHOD | | | |
| Problem: | | | |
|  |  | | True Solution |
|  |  |
|  |  |  | 1 |
|  | 2.3 | 0.9399 | 0.9399 |
|  | 2.6 | 0.9292 | 0.9292 |
|  | 2.9 | 0.9511 | 0.9511 |
|  | 3.2 | 0.9941 | 0.9942 |
|  | 3.5 | 1.05028 | 1.0503 |

**Approach**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Mathematical:* As depicted in the above calculations, the mathematical approach to solving the given ODE is to identify the initial values and use a method such as RK4 to formulaically calculate the solution per x for y.

*Code:*



The program was written in python using notepad text editor and run in an Ubuntu wsl terminal.

The algorithm defines two functions to solve the ODE. One function, defined RKSolve(xm,ym,h,m) takes in parameters of the initial x and y values of the problem, as well as step size h and count m of y values. The function sequentially computes the values for the k according to formulaic statements, given the parameters. It replaces y and increments x by the step size for m loops.

On the other hand, the ODESolve function calls the odeint function from SciPy integrate with the parameters of a function ODE that returns the definition of the given ODE, the initial y value, and a list of x values (incremented by step size), globally and pre-initialized with Numpy’s arange() function which can take parameters of a start, an end that is not included, and a step size to create a list of values (Numpy Developers, n.d. ). The ODE returns the solution of y for each value of x and stores them in a list (SciPy Community, n.d.).

Furthermore, the program provides the computational time and computational steps needed within the graphical displays. The computational time is calculated using python’s time library’s time() function by subtracting the time before each function is called from that after it is called. The time function provides a means to measure the execution time between two points as two instantiations can be subtracted to give a time difference (Python Software Foundation, n.d.). The computational steps are calculated according to the formula steps = 5 \* m where m is the number of y values computed as 5 formulas must be computed for each y value computed.

**Screenshots of Runge-Kutta.py**

Github repository:

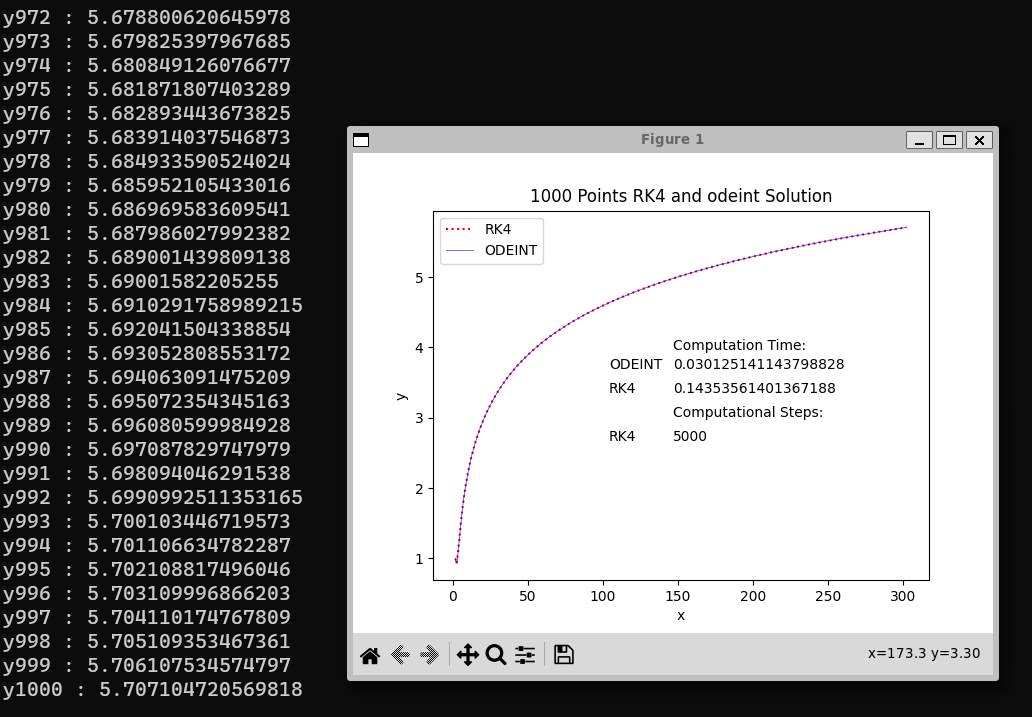
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*A screenshot of a computer

Description automatically generatedDepicts the graph for 1000 y solutions per x for the given ODE solved by the RK4 method as well as computational time and steps.*

*A screenshot of a computer

Description automatically generated*

*Depicts the graph for 1000 y solutions per x for the given ODE solved by the SciPy odeint method as well as computational time and steps.*

*Depicts the graph for 1000 y solutions per x for the given ODE solved by both the RK4 and the SciPy odeint method as well as computational time and steps.*

**References**

Libretexts. (2020). *3.3: The Runge-Kutta method*. Mathematics LibreTexts. https://math.libretexts.org/Courses/Monroe\_Community\_College/MTH\_225\_Differential\_Equations/3%3A\_Numerical\_Methods/3.3%3A\_The\_Runge-Kutta\_Method

Numpy Developers. (n.d.). *Numpy.arange*. NumPy v1.26 Manual. <https://numpy.org/doc/stable/reference/generated/numpy.arange.html#numpy.arange>

SciPy Community. (n.d.). *Scipy.integrate.odeint*. SciPy v1.11.3 Manual. https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

Python Software Foundation. (n.d.). *Time - Time Access and conversions*. Python Documentation. https://docs.python.org/3/library/time.html