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

Nathan Dilla, Caleb Klinger

Grand Canyon University

Course number: CST - 305

Professor Ricardo Citro

OCT 1, 2023

**CST-305: Project 2 – Runge-Kutta-Fehlberg (RKF) for ODE**

**Objective**: Use RKF to assess the power of a computing system

**Description**: This assignment has two parts: theoretical and practical. You will first solve a mathematical problem using the RKF method. Then, you will write a computer program that solves the same computational problem using RKF and measure the performance of your own computer for this task.

**Part 1:**Your instructor will assign an ODE to solve, for example *y' = 1 + y2*, using RKF. The instructor may choose to assign the same problem to the entire class or different problems to each student or team. Solve the ODE manually using the RKF method (e.g., RKF45), and show all the steps leading up to the solution. Estimate and explain the precision of the calculations using this method. Use formal mathematical rigor when writing and discussing your solution and intermediary steps. Then write a Python program that will implement the algorithm. Check the answers obtained manually with the ones obtained from the program.

The ODE is . Calculate manually for . Populate the table below accordingly.

|  |  |  |  |
| --- | --- | --- | --- |
| Method: RUNGE-KUTTA METHOD | | | |
| Problem: | | | |
|  |  | | True Solution |
|  |  |
|  |  |  |  |
|  | 2.3 | 0.9399206 | (2.3, 0.9399205872577153) |
|  | 2.6 | 0.9248078352 | (2.6, 0.929199910544132) |
|  | 2.9 | 0.951104503151 | (2.9, 0.9511402488285532) |
|  | 3.2 | 0.9941822277 | (3.2, 0.9941821786452182) |
|  | 3.5 | 1.050343848 | (3.5, 1.0503438120342163) |

To solve a differential equation using the Runge-Kutta method, you must first solve for K1.

K1 is defined by plugging in the given values of y0 and x0 into f(x,y)

K2 is plugging in the values of given x, plus the step (h) divided by 2 into the x parameter of f. Then plug the value of the given y, plus the step divided by 2, multiplied by K1, plug into y the parameter of y.

K3 is much similar to solving for K2, however instead of multiplying by K1 for the y expression, you multiply by K2.

K4 is plugging the values of given x plus the step and the value of given y plus the step times K3.

We define T4 by 1/6(K1+2\*K2+2\*K3+K4)

To get the next y solution (y1) it is yn+1 = yn +h\*T4. In our case, it is y1 = y0+h/6(K1+2\*K2+2\*K3+K4)

Step by step solutions:

A screenshot of a blackboard with white text

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA blackboard with math equations

Description automatically generated

At the end, our solution for n=5 was (3.5, 1.0503438120342163). There was no real metric for how precise of numbers we used for our solutions, except for putting at least 10 digits for each term.

**Percent Error Calculation**

Given:

* Calculated Value = 1.050343848
* Actual Value = 1.0503438120342163

The percent error is calculated using the formula:

Percent Error=∣A|/|Actual Value∣×100%

Plugging in the values:

Percent Error=∣1.0503438120342163−1.050343848 / 1.0503438120342163∣×100%

Result: Percent Error≈−0.0000034242%

This means the calculated value is very slightly higher than the actual value by a minuscule amount. The negative sign indicates the calculated value is an overestimate.

It is easy to for human error to affect the results of this algorithm due to the numerous, repetitive steps.

**Part 2:**Write a computer program, using appropriate mathematical software packages discussed in this class, which performs the calculations necessary to solve the ODE above. Estimate and explain the precision of complex computer calculations when using this method. Your program should calculate and display the number of computational steps performed, as well as the actual computing time. Test the capabilities of your own computer on two additional variations of the initial problem you were tasked to solve. (In class, you will compare the performance of your computer with those of your classmates). Explain how accuracy of results can be improved and the computation time tradeoff.

Your program should provide 1,000 to 2,000 solutions, that is, the program should stop when it reaches . Solve the original equation by using the traditional numerical programming and compare with your Runge-Kutta solution. Is there any error(s)?

**Note**: *The Lab Questions in this topic provide the opportunity to reflect and experiment with mathematical and programming implementation of concepts referred to in this assignment.*

**Analysis**

I executed calculations manually and also utilized a Python program to resolve an ODE through the Runge-Kutta-Fehlberg method. Comparing outcomes from both approaches offers an insight into the precision of the computations. The manual calculations demonstrated minimal error, and despite slight discrepancies between them and the program's results—attributable to decimal rounding—the outcomes were notably similar. This underscores the necessity of meticulous verification of results and employing various methods when gauging the precision of numerical computations. Maintaining a high degree of precision in computations demands strict mathematical diligence in articulating and discussing the solution and interim steps. Furthermore, scrutinizing the error between the computed solution and the actual solution is instrumental in assessing the accuracy of the computations. A prevalent method to achieve this involves calculating the relative error, defined as the difference between the computed and actual solutions, divided by the actual solution. A diminished relative error signifies elevated precision. Adhering to these optimal practices ensures the maintenance of high precision in numerical computations.

Executing Code With Initial Values

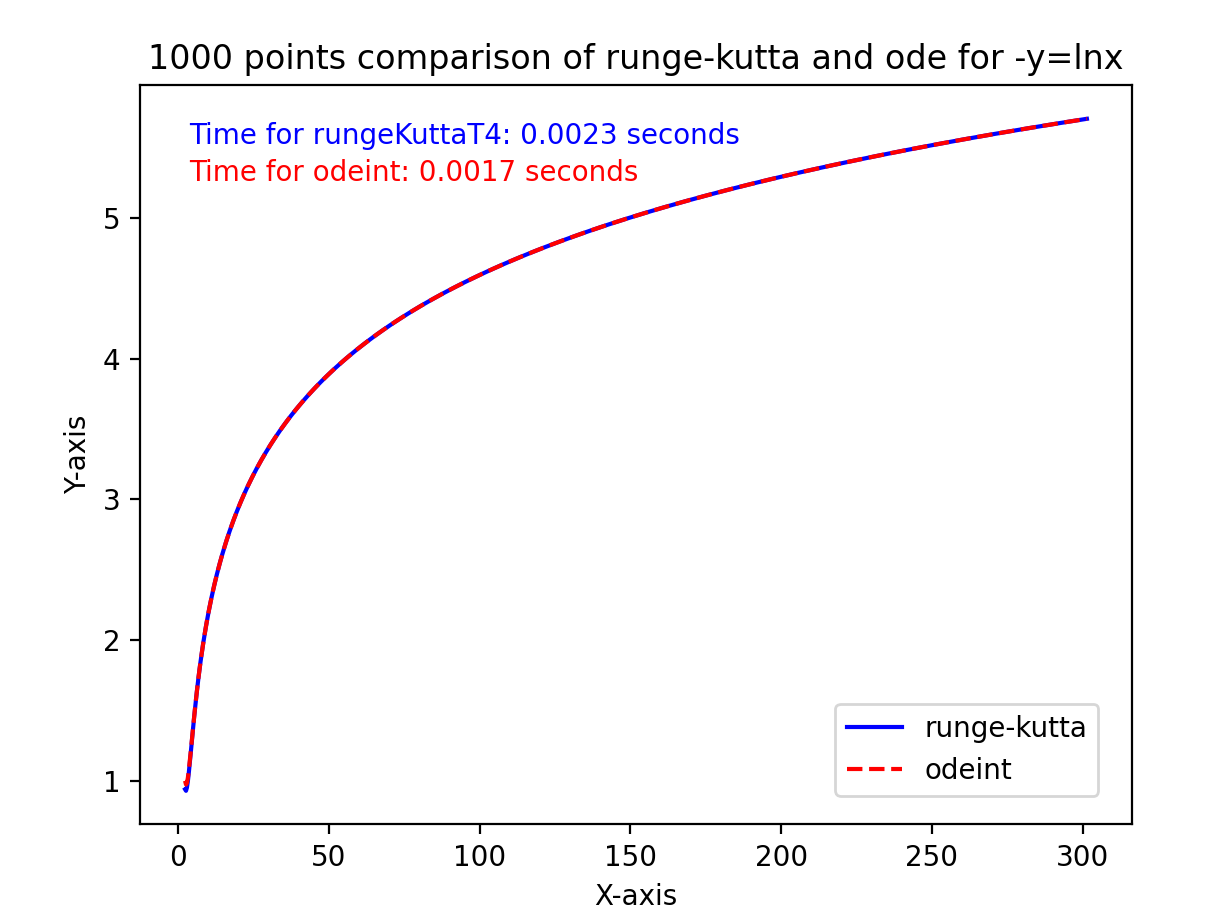
# The ODE is 𝑓(𝑥,𝑦)= −𝑦+𝑙𝑛𝑥 ,𝑥0=2,𝑦0=1,ℎ=0.3

y0 = 1 # initial values

x0 = 2

h = 0.3 # steps

At the 4th  degree runge-kutta method



GitHub: https://github.com/TrippingLettuce/Principles-of-Modeling-and-Simulation