Project 2: Runge Kutta for ODE

Diego Guerra & Ryan Scott

CST-305

Dr. Ricardo Citro

Part 1

Solving Runge Kutta by Hand:

METHOD: RUNGE KUTTA			
Problem: $y' = \frac{(y)}{e^x - 1}$; x_0 = 1; y_0 =5			
n	h = 0.02		True Solution
	X_n	Y_n	
0	1	5	5
1	1.02	5.0576	5.0576
2	1.04	5.1141	5.1141
3	1.06	5.1695	5.1695
4	1.08	5.2237	5.2237
5	1.10	5.2769	5.2769

Runge Kutta

$y' = \frac{y}{e^x - 1}$ $y_0 = 5$ $x_0 = 1$ $h = 0.02$

$K_1 = \frac{5}{e^1 - 1} = 2.9099$

$K_2 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{hk_1}{2}\right) = f(1.01, 5.0291)$

$= \frac{5.0291}{e^{1.01} - 1} = 2.8810$

$K_3 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{hk_2}{2}\right) = f(1.01, 5.0288)$

$= \frac{5.0288}{e^{1.01} - 1} = 2.8808$

$K_4 = f(x_0 + h, y_0 + hk_3) = f(1.02, 5.0576)$

$= \frac{5.0576}{e^{1.02} - 1} = 2.8523$

$y_1 = y_0 + \frac{h}{6}(K_1 + 2k_2 + 2k_3 + K_4) =$

$5.0576 = y_1$

Step 2:

$y' = \frac{y}{e^x - 1}$ $\quad x_1 = 1.02 \quad y_1 = 5.0576$

$h = 0.02$

$k_1 = f(x_1, y_1) = f(1.02, 5.0576)$

$= \frac{5.0576}{e^{1.02} - 1} = 2.8523$

$k_2 = f\left(x_1 + \frac{h}{2}, y_1 + \frac{hk_1}{2}\right) = f(1.03, 5.0861)$

$= \frac{5.0861}{e^{1.02} - 1} = 2.8240$

$k_3 = f\left(x_1 + \frac{h}{2}, y_1 + \frac{hk_2}{2}\right) = f(1.03, 5.0859)$

$= \frac{5.0859}{e^{1.03} - 1} = 2.7958$

$k_4 = f(x_1 + h, y + hk_3) = f(1.04, 5.1141)$

$= \frac{5.1141}{e^{1.04} - 1} = 2.7958$

$y_2 = y_1 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) =$

$5.1141 = y_2$

Step 3: $x_2 = 1.04 \quad y_2 = 5.1141$

$h = 0.02$

$k_1 = f(x_2, y_2) = f(1.04, 5.1141)$

$= \frac{5.1141}{e^{1.04} - 1} = 2.7958$

$k_2 = f\left(x_2 + \frac{h}{2}, y_2 + \frac{k_1 h}{2}\right) = f(1.05, 5.1421)$

$= \frac{5.1421}{e^{1.05} - 1} = 2.7680$

$k_3 = f\left(x_2 + \frac{h}{2}, y_2 + \frac{k_2 h}{2}\right) = f(1.05, 5.1418)$

$= \frac{5.1418}{e^{1.05} - 1} = 2.7679$

$k_4 = f(x_2 + h, y_2 + k_3 h) = f(1.06 \; 5.1695)$

$= \frac{5.1695}{e^{1.06} - 1} = 2.7404$

$y_3 = y_2 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) =$

$5.1695 = y_3$

Part 4:

$x_3 = 1.06 \quad y_3 = 5.1695$

$h = 0.02$

$y_1 = f(x_3, y_3) = f(1.06, 5.1695)$

$= \frac{5.1695}{e^{1.06} - 1} = 2.74104$

$k_1 = f(x_3 + h, y_3 + \frac{hk_1}{2}) = f(1.07, 5.1969)$

$\frac{5.1969}{e^{1.07} - 1} = 2.7132$

$k_2 = f(x+h, y_3 + \frac{hk_2}{2}) = f(1.07, 5.1966)$

$= \frac{5.1966}{e^{1.07} - 1} = 2.7131$

$k_4 = f(x+h, y_3 + hk_3) = f(1.08, 5.2237)$

$= \frac{5.2237}{e^{1.08} - 1} = 2.6862$

$y_4 = y_3 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) =$

$5.2237 = y_4$

Step 5 $x_4 = 1.08$ $y_{4} = 5.2237$

$h = 0.02$

$K_1 = f(x_4, y_4) = f(1.08, 5.2237)$

$= \frac{5.2237}{e^{1.08} - 1} = 2.6862$

$K_2 = f(x_4 + \frac{h}{2}, y_4 + \frac{hK_1}{2}) = f(1.09, 5.2506)$

$= \frac{5.2506}{e^{1.09} - 1} = 2.6595$

$K_3 = f(x_4 + \frac{h}{2}, y_4 + \frac{hK_2}{2}) = f(1.09, 5.2503)$

$= \frac{5.2503}{e^{1.09} - 1} = 2.6593$

$K_4 = f(x_4 + h, y_4 + hK_3) = f(1.1, 5.2769)$

$= \frac{5.2769}{e^{1.1} - 1} = 2.6330$

$y_5 = y_4 + \frac{h}{2}(K_1 + 2K_2 + 2K_3 + K_4) =$

$5.2769 = Y_3$

Step 6 $x_5 = 1.1 \quad y_5 = 5.2769$

$h = 0.02$

$k_1 = f(x_5, y_5) = f(1.1, 5.2769)$

$\frac{5.2769}{e^{1.1} - 1} = 2.6330$

$k_2 = f(x_5 + \frac{h}{2}, y_5 + \frac{hk_1}{2}) = f(1.11, 5.3032)$

$\frac{5.3032}{e^{1.11} - 1} = 2.6068$

$k_3 = f(x_5 + \frac{h}{2}, y_5 + \frac{hk_2}{2}) = f(1.11, 5.3030)$

$\frac{5.3030}{e^{1.11} - 1} = 2.6067$

$k_4 = f(x_5 + h, y_5 + \frac{hk_3}{2}) = f(1.12, 5.3290)$

$\frac{5.3290}{e^{1.12} - 1} = 2.5808$

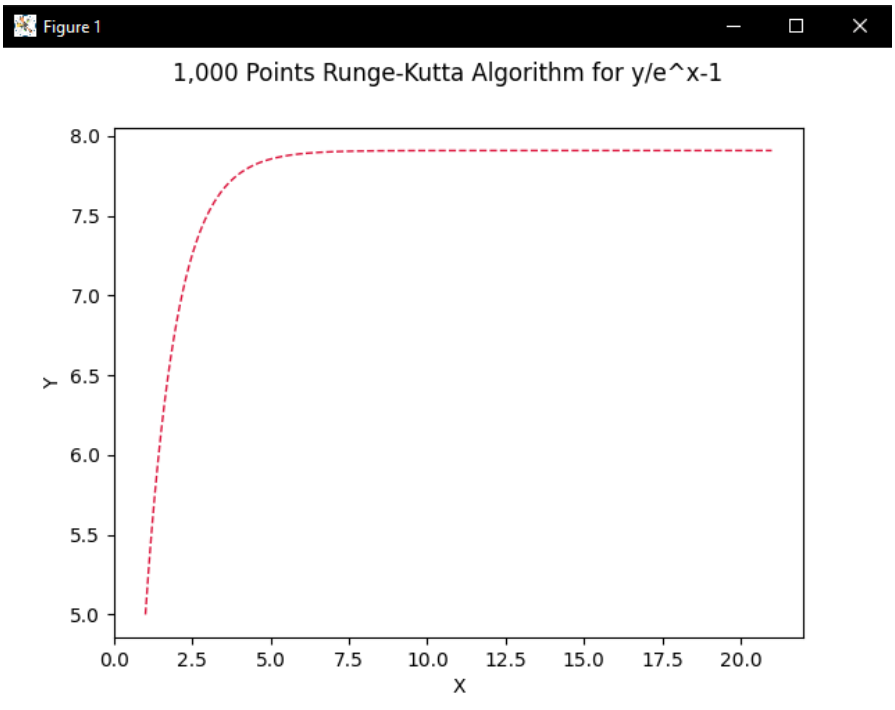
$y_6 = y_5 + \frac{h}{2}(k_1 + 2k_2 + 2k_3 + k_4) =$

$5.3290 = y_6$

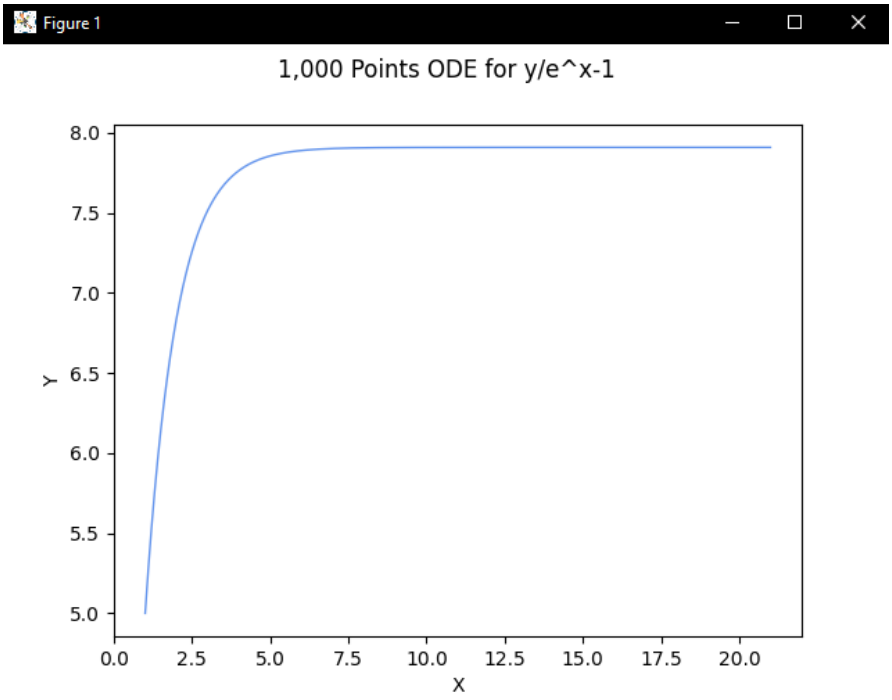
Part 2:

Python Program:

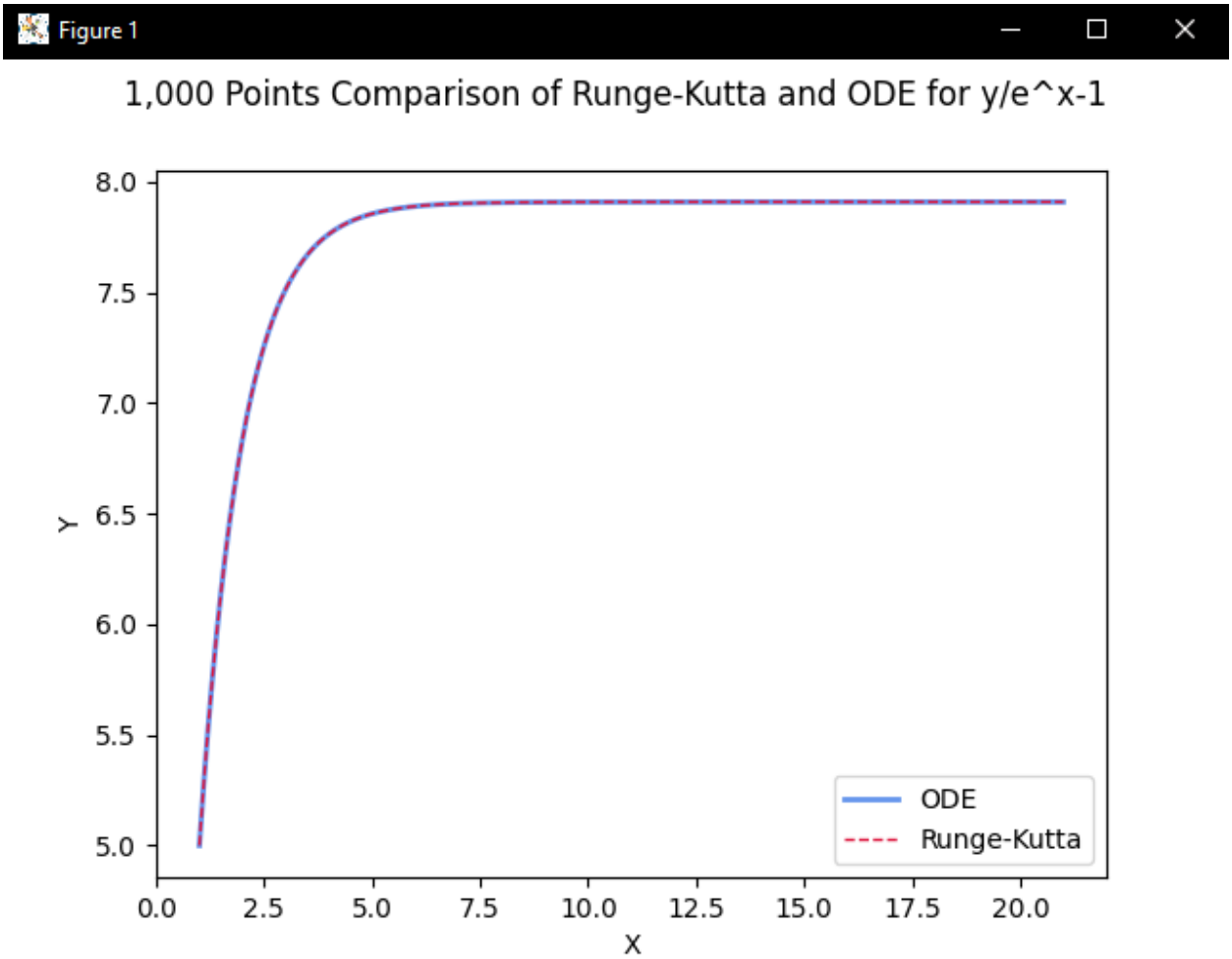
Runge-Kutta Method Results:



ODEINT Method Results:



Results of both:



Precision of Complex Computer Calculation:

There were definitely ways that we could have improved the code, and one way that we could have done that was by using the round function in Python. We weren't too concerned about the code's efficiency; we were more concerned with its successful execution. Using the round function in Python is one of the simpler things we could have done. In this function, you can round the number to any decimal place value that you want. If we had rounded it up to four decimal places, it would have sacrificed accuracy but would have increased efficiency since fewer values would have to be handled.

We could have improved the code's efficiency by using Numba as well. Python is notoriously a sluggard language, so we should take advantage of any opportunity to make the code run faster. The Numba Python library takes advantage of high speed coding languages, such as c++, and compiles the code in that language, thus decreasing the overall time involved in compiling the code and running it. Our code will be more efficient than with rounding if we use Numba, as we won't have to sacrifice accuracy. The only drawback with Numba is that Ryan and Diego have very limited knowledge of Numba, so we would have to take some time to learn how to properly utilize the library..

Code Execution:

Among the first changes we made to the code was creating a function that could solve the Runge-Kutta equation with any equation and any value. Several attempts were made to code this function correctly, and finally we were able to achieve it. It contains several variables, but its primary component is to obtain each ‘k’ value and return the value of:

$$y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Our next step was to make the ODEInt function of the equation. This was quite easy, all we had to do was to make an ODEInt function that contains the equation.

$$y' = \frac{y}{e^x - 1}$$

To complete the process, we used MatPlotLib to graph the equation they created and the odeint function. Additionally, we created a comparison graph showing both equations. Additionally, we researched a lot about how to stylize our graph to make it look very appealing to the eye, as well as how to change it to make it more attractive. You can visualize graphs more easily with MatPlotLib's line designs, and graph backgrounds.

Speed of Code Execution:

With the help of importing time, we measured the speed of their code execution by storing the time before the computation begins and then storing the end time of that computation, finally finding the difference between the two to get a final result.

```
Runge-Kutta Algorithm ran for 0.38483691215515137 seconds!

Odeint ran for 0.0037178993225097656 seconds!
```

Unsurprisingly, Odeint beat out our custom recursive Runge-Kutta function by a longshot, as it is optimized to handle ODEs such as these.