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

Jack Utzerath

College of Technology, Grand Canyon University

CST-305: Principles of Modeling and Simulation

Ricardo Citro

10/1/23

Responsibilities and completed tasks by each team member

All tasks are done by Jack Utzerath

Specific Problem Solved

I am using the 4th-order Runge-Kutta-Fehlberg (RKF) method to solve an ODE manually and through programming. The ODE is $f(x,y) = -y + \ln(x)$.

The mathematical approach to solving it

Cst 305
Jack Utzerath

Runge-Kutta-~~Ke~~ Fehlberg (RKF) for ODE

Givens

ODE: $f(x,y) = -y + \ln(x)$ $\quad (x_0, y_0) = (2, 1)$

$y_0 = 1 \quad x_0 = 2 \quad h = 0.3$

$K_1 = f(x_0, y_0)h = (-1 + \ln(2)) \times .3 = -0.0920558$
$K_2 = f(x_0 + \frac{h}{2}, y_0 + \frac{K_1}{2}) \cdot h = hf(2.15, .953972) = (-.953972 + \ln(2.15)) \cdot .3 = -.0565513$
$K_3 = f(x_0 + \frac{h}{2}, y_0 + \frac{K_2}{2}) \cdot h = hf(2.15, .9717244) = (-.9717244 + \ln(2.15)) \cdot .3 = -.061876\,96$
$K_4 = f(x_0 + h, y_0 + K_3) \cdot h = hf(2.3, .938123) = (-.938123 + \ln(2.3)) \cdot .3 = -0.031564$
$y_1 = y_0 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) = \underline{0.939921}$

$(x_1, y_1) = (2.3, 0.939921)$

$K_1 = f(x_1, y_1) \cdot h = (-.939921 + \ln(2.3)) \cdot .3 = -0.032103$
$K_2 = f(x_1 + \frac{h}{2}, y_1 + \frac{K_1}{2}) \cdot h = f(2.45, .923869) \cdot .3 = (-.923869 + \ln(2.45)) \cdot .3 = -.0083344$
$K_3 = f(x_1 + \frac{h}{2}, y_1 + \frac{K_2}{2}) \cdot h = f(2.45, .935754) \times .3 = (-.935754 + \ln(2.45)) \times .3 = -.0118997$
$K_4 = f(x_1 + h, y_1 + K_3) \cdot h = f(2.6, .928021) \times .3 = (-.928021 + \ln(2.6)) \times .3 = .0082471$
$y_2 = y_1 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) = 0.9292$

$(x_2, y_2) = (2.6, .929200)$

$K_1 = f(x_2, y_2) \cdot h = (-.9292 + \ln(2.6)) \cdot .3 = .007893$
$K_2 = f(x_2 + \frac{h}{2}, y_2 + \frac{K_1}{2}) \cdot h = f(2.75, .93315) \times .3 = (-.93315 + \ln(2.75)) \times .3 = .023536$
$K_3 = f(x_2 + \frac{h}{2}, y_2 + \frac{K_2}{2}) \cdot h = f(2.75, .940968) \times .3 = (-.940968) + \ln(2.75)) \times .3 = .0211898$
$K_4 = f(x_2 + h, y_2 + K_3) \cdot h = f(2.9, .9503898) \times .3 = (-.9503898 + \ln(2.9)) \times .3 = .034\cancel{2}$
$y_3 = y_2 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) = 0.95114$

$(x_3, y_3) = (2.9, .95114)$

$K_1 = f(x_3, y_3) \cdot h = (-.95114 + \ln(2.9)) \cdot .3 = 0.034071$
$K_2 = f(x_3 + \frac{h}{2}, y_3 + \frac{K_1}{2}) \cdot h = f(3.05, 0.968176) \times .3 = (-.968176 + \ln(3.05)) \times .3) = .04408979$
$K_3 = f(x_3 + \frac{h}{2}, y_3 + \frac{K_2}{2}) \cdot h = f(3.05, .973185) \times .3 = (-.973185 + \ln(3.05)) \times .3 = .042587$
$K_4 = f(x_3 + h, y_3 + K_3) \cdot h = f(3.2, .993727) \times .3 = (-.993727 + \ln(3.2)) \times .3 = .05082714$
$y_4 = y_3 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) = 0.9941818$

$(x_4, y_4) = (3.2, .9941818)$

$K_1 = f(x_4, y_4) \cdot h = (-.9941818 + \ln(3.2)) \times .3 = .0506907$
$K_2 = f(x_4 + \frac{h}{2}, y_4 + \frac{K_1}{2}) \cdot h = f(3.35, 1.019527) \times .3 = (-1.019527 + \ln(3.35)) \times .3 = 0.05682995$
$K_3 = f(x_4 + \frac{h}{2}, y_4 + \frac{K_2}{2}) \cdot h = f(3.35, 1.022597) \times .3 = (-1.022597 + \ln(3.35)) \times .3 = 0.0559090\overline{7}$
$K_4 = f(x_4 + h, y_4 + K_3) \cdot h \quad f(3.5, 1.05009087) \times .3 = (-1.05009087 + \ln(3.5)) \times .3 = 0.0668016$
$y_5 = y_4 + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) = 1.0503435$

$(x_5, y_5) = (3.5, 1.0503435)$

Here I manually calculated (x0,y0) through (x5, y5). These points were calculated by using the RKF method given in class. Since this a 4-order solving method, k1 through k4 need to be found before calculating the next point. To find the next y, this equation is used:

$$Y_{n+1} = Y_n + (1/6)(K1 + 2 *K2 + 2*K3 + K4)$$

The next x is just Xn + the step size which is 0.3 in this case.

The approach for implementation in code

For the implementation in Code, I used three packages. Numpy, matplotlib, and scipy. I first used odeint to solve the ODE. This is a function in scipy that is used to calculate the next point for differentiable equations. Then I used the runge kutta method to solve the ODE. I stored the variables in an array and used a for loop to update the x and y values. In the for loop, I also printed the values to the console. With these two methods, I was able to use matplotlib to plot these functions and compare them both on the same graph.

References for theory and code sources

Fehlberg, E. (1968). Classical fifth-, sixth-, seventh-, and eighth-order Runge–Kutta–Nystrom formulae. Computer Journal, 11(2), 125-132. https://doi.org/10.1093/comjnl/11.2.125

GeeksforGeeks. (2023, January 17). *Runge-Kutta 4th order method to solve differential equation*. GeeksforGeeks. https://www.geeksforgeeks.org/runge-kutta-4th-order-method-solve-differential-equation/

Readme Document written in Markdown detailing how to install and run the program

https://github.com/utzerath/Solving-ODE-Using-RKF/tree/main

Full code submitted to GitHub

https://github.com/utzerath/Solving-ODE-Using-RKF/tree/main

Github SC

utzerath Update README.md ... 3 minutes ago 3

README.md	Update README.md	3 minutes ago
RKF.py	Add files via upload	5 minutes ago

README.md

Solving-ODE-Using-RKF

#download python file and use ide or linux to run program #Packages Used: Matplotlib, numpy, and scipy

#For the implementation in Code, I used three packages. Numpy, matplotlib, and scipy. I first used odeint to solve the ODE. This #is a function in scipy that is used to calculate the next point for differentiable equations. Then I used the runge kutta #method to solve the ODE. I stored the variables in an array and used a for loop to update the x and y values. In the for loop, #I also printed the values to the console. With these two methods, I was able to use matplotlib to plot these functions and #compare them both on the same graph.

Code:

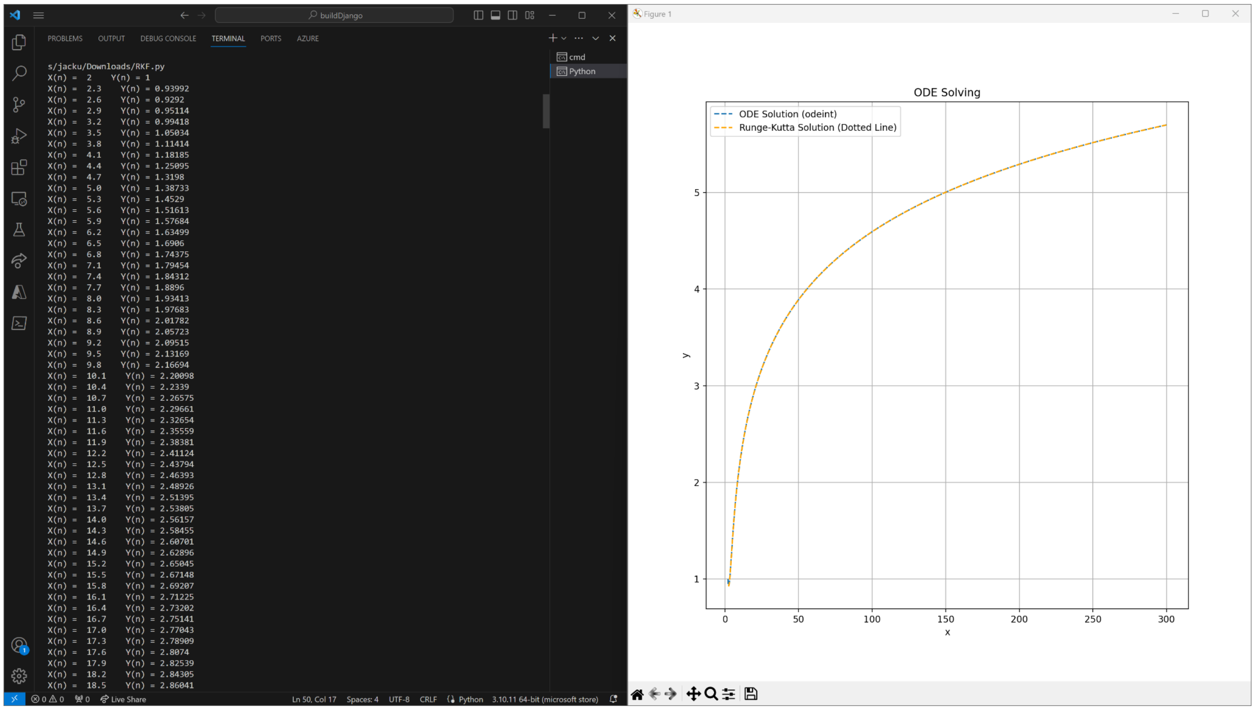
This screenshot depicts the ODE solved through Odeint

```python
#Jack Utzerath
#Packages: Numpy, matplotlib, scipy
#using 4th order RKF method to solve ODE

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# Define the ODE function: f(x, y) = -y + ln(x)
def f(y, x):
    return -y + np.log(x)

# Initial conditions and parameters
x0 = 2.0
y0 = 1.0
x_target = 300
x_values = np.linspace(x0, x_target, 1000)  # Create evenly spaced x values
y_values = odeint(f, y0, x_values)[:, 0]

# Create a matplotlib graph of the solution
plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, '--', label="ODE Solution (odeint)")
plt.xlabel("x")
plt.ylabel("y")
plt.title("ODE Solving")
plt.grid(True)
```

This screenshot depicts the ODE being solved through RKF

```python
#Reading ODE as an input
def dydx(x, y):
    return -y + np.log(x)

#solving the ode using the runge kutta fehlburg method
def rungeKutta(x0, y0, x, h):
    #initialize array
    x_values = []
    y_values = []

    #set up n and y
    n = (int)((x - x0) / h)
    y = y0

    #for loop for rkf
    for i in range(1, n + 1):

        #displays x and y to console
        print(f"n = {n}     X(n) =  {round(x0, 5)}     Y(n) = {round(y, 5)} ")
        k1 = h * dydx(x0, y)
        k2 = h * dydx(x0 + 0.5 * h, y + 0.5 * k1)
        k3 = h * dydx(x0 + 0.5 * h, y + 0.5 * k2)
        k4 = h * dydx(x0 + h, y + k3)

        #update y and x
        y = y + (1.0 / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
        x0 = x0 + h

        #add variables to array
        x_values.append(x0)
        y_values.append(y)

    #return values
    return x_values, y_values

# Driver method
x0 = 2
y0 = 1
x = 1000
h = 0.3

x_values, y_values = rungeKutta(x0, y0, 300, h)

# Create a matplotlib graph with a dotted line up to x=300
plt.plot(x_values, y_values, '--', label="Runge-Kutta Solution (Dotted Line)", linestyle='--',
plt.legend()
plt.show()
```

Code Output


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
AZURE
buildDjango
cmd
Python
s/jacku/Downloads/RKF.py
X(n) = 2 Y(n) = 1
X(n) = 2.3 Y(n) = 0.93992
X(n) = 2.6 Y(n) = 0.9292
X(n) = 2.9 Y(n) = 0.95114
X(n) = 3.2 Y(n) = 0.99418
X(n) = 3.5 Y(n) = 1.05034
X(n) = 3.8 Y(n) = 1.11414
X(n) = 4.1 Y(n) = 1.18185
X(n) = 4.4 Y(n) = 1.25095
X(n) = 4.7 Y(n) = 1.3198
X(n) = 5.0 Y(n) = 1.38733
X(n) = 5.3 Y(n) = 1.4529
X(n) = 5.6 Y(n) = 1.51613
X(n) = 5.9 Y(n) = 1.57684
X(n) = 6.2 Y(n) = 1.63499
X(n) = 6.5 Y(n) = 1.6906
X(n) = 6.8 Y(n) = 1.74375
X(n) = 7.1 Y(n) = 1.79454
X(n) = 7.4 Y(n) = 1.84312
X(n) = 7.7 Y(n) = 1.8896
X(n) = 8.0 Y(n) = 1.93413
X(n) = 8.3 Y(n) = 1.97683
X(n) = 8.6 Y(n) = 2.01782
X(n) = 8.9 Y(n) = 2.05723
X(n) = 9.2 Y(n) = 2.09515
X(n) = 9.5 Y(n) = 2.13169
X(n) = 9.8 Y(n) = 2.16694
X(n) = 10.1 Y(n) = 2.20098
X(n) = 10.4 Y(n) = 2.2339
X(n) = 10.7 Y(n) = 2.26575
X(n) = 11.0 Y(n) = 2.29661
X(n) = 11.3 Y(n) = 2.32654
X(n) = 11.6 Y(n) = 2.35559
X(n) = 11.9 Y(n) = 2.38381
X(n) = 12.2 Y(n) = 2.41124
X(n) = 12.5 Y(n) = 2.43794
X(n) = 12.8 Y(n) = 2.46393
X(n) = 13.1 Y(n) = 2.48926
X(n) = 13.4 Y(n) = 2.51395
X(n) = 13.7 Y(n) = 2.53805
X(n) = 14.0 Y(n) = 2.56157
X(n) = 14.3 Y(n) = 2.58455
X(n) = 14.6 Y(n) = 2.60701
X(n) = 14.9 Y(n) = 2.62896
X(n) = 15.2 Y(n) = 2.65045
X(n) = 15.5 Y(n) = 2.67148
X(n) = 15.8 Y(n) = 2.69207
X(n) = 16.1 Y(n) = 2.71225
X(n) = 16.4 Y(n) = 2.73202
X(n) = 16.7 Y(n) = 2.75141
X(n) = 17.0 Y(n) = 2.77043
X(n) = 17.3 Y(n) = 2.78909
X(n) = 17.6 Y(n) = 2.8074
X(n) = 17.9 Y(n) = 2.82539
X(n) = 18.2 Y(n) = 2.84305
X(n) = 18.5 Y(n) = 2.86041
Live Share
Ln 50, Col 17
Spaces: 4
UTF-8
CRLF
Python
3.10.11 64-bit (microsoft store)
Figure 1
ODE Solving
ODE Solution (odeint)
Runge-Kutta Solution (Dotted Line)
y
x
0
50
100
150
200
250
300
1
2
3
4
5