Assignment 4

June Knauth | CMPT360 | 20211005

In this assignment, I created a Python program which solves for the motion of a simple pendulum using the fourth-order Runge-Kutta method. It also animates the motion of the pendulum

I used the following sources:

https://en.wikipedia.org/wiki/Runge–Kutta_methods

https://matplotlib.org/stable/gallery/animation/double_pendulum.html (for graphics)

And the provided RK4 example. The code is well documented, so I'll let it speak for itself here:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  4 20:14:34 2021

@author: june
"""

from numpy import sin, cos, pi
import numpy as np
import matplotlib.animation as animation
import matplotlib.pyplot as plt

"""
Runge-Kutta method for single pendulum, animated.

Here are the equations:

theta' = omega
omega' = -g/r * sin(theta)

"""

l = 1 # length of pendulum in m
g = 9.81 # gravitational constant in m/s^2
n = 300 # number of iterations
theta_0 = pi/4 # initial angle in rad, counterclockwise from negative y axis
omega_0 = 1 # initial angular velocity in rad/sec
tfinal = 10 # integrate up to this time
```

```
# animation fps will be n/tfinal (iterations/second)

h = tfinal/n # time step

t = np.arange(0, tfinal, h) # time values


def runge_kutta(n, theta, t, omega, tfinal, h):
    theta_arr = np.zeros(n) # np arrays to store values
    omega_arr = np.zeros(n)

    """
    Adapted from the given material on RK
    """

    for i in range(n):
        theta_arr[i] = theta
        omega_arr[i] = omega


        k1_omega = -1*(g/l)*sin(theta)
        k1_theta = omega

        k2_omega = -1*(g/l)*sin(theta + k1_theta*h/2)
        k2_theta = omega + k1_omega*h/2

        k3_omega = -1*(g/l)*sin(theta + k2_theta*h/2)
        k3_theta = omega + k2_omega*h/2

        k4_omega = -(g/l)*sin(theta + k3_theta*h)
        k4_theta = omega + k3_omega*h

        theta += (k1_theta+2*k2_theta+2*k3_theta+k4_theta)*h/6

        omega += (k1_omega+2*k2_omega+2*k3_omega+k4_omega)*h/6

        t += h

    return theta_arr, omega_arr # return thetas, one for each time t

thetas_t, omegas_t = runge_kutta(n, theta_0, 0, omega_0, tfinal, h) # get values for each t
# using fourth-order runge-kutta method

"""
At this point, what we have is two arrays, one of t values, the
other of their respective theta values. That's what RK4
gives us. Everything after this is for graphing.

(also there's an array of omegas for the display)
"""

x = l*sin(thetas_t) # convert thetas into cartesian coordinates
y = -l*cos(thetas_t) # yay physics 1
```

```
# setup the graph
fig = plt.figure(figsize=(5, 4))
ax = fig.add_subplot(autoscale_on=True, xlim=(-l-1, l+1), ylim=(-l-1, l+1))
ax.set_aspect('equal')
ax.grid()

line, = ax.plot([], [], 'o-', lw=2) # sets up the line to be graphed

time_template = 'time = %.1fs' # display values
theta_template = 'theta = %.1f rad'
omega_template = 'omega = %.1f rad/s'

time_text = ax.text(0.05, 0.88, '', transform=ax.transAxes)
theta_text = ax.text(0.05, 0.83, '', transform=ax.transAxes)
omega_text = ax.text(0.05, 0.78, '', transform=ax.transAxes)


"""
Here's how this works:

The animation.FuncAnimation() function calls animate(i) with increasing
i values. animate(i) indexes into the position and time arrays to return
their respective values at those indices.
Then matplotlib uses those values to graph the figure at each index.
"""


def animate(i):
    thisx = [0, x[i]] # the 0 is because the line is from 0 to the point
    thisy = [0, y[i]]

    line.set_data(thisx, thisy) # plot line
    time_text.set_text(time_template % (i*h)) # plot data values
    theta_text.set_text(theta_template % thetas_t[i])
    omega_text.set_text(omega_template % omegas_t[i])

    return line, time_text, theta_text, omega_text


ani = animation.FuncAnimation(
    fig, animate, len(y), interval=h*1000, blit=True)
plt.show()

# # Here's the code to save the animation:
# writervideo = animation.FFMpegWriter(fps=30)
# ani.save('animation.mp4', writer=writervideo)
```