Homework01

February 8, 2021

Victor Cruz Ramos
Your Woskshop section here.

1 Homework 1: Python basics and a little plotting

** Submit this notebook to bCourses to receive a credit for this assignment.**

Please complete this homework assignment in code cells in the iPython notebook. Include comments in your code when necessary. Enter your name in the cell at the top of the notebook, and rename the notebook [email_name]_HW01.ipynb, where [email_name] is the part of your UCB email address that precedes "@berkeley.edu". Please also submit a PDF of the jupyter notebook to bcourses.

1.1 Problem 1: Satellite Altitudes

[Adapted from Newman, Exercise 2.2] A satellite is to be launched into a circular orbit around the Earth so that it orbits the planet once every T seconds. The altitude h above the Earth's surface that the satellite must have is

$$h = \left(\frac{GMT^2}{4\pi^2}\right)^{1/3} - R,$$

where $G = 6.67 \times 10^{-11}$ m^3 kg^{-1} s^{-2} is Newton's gravitational constant, $M = 5.97 \times 10^{24}$ kg is the mass of the Earth, and $R = 6371$ km is its radius.

1a. Write a program that, for a given value of T (entered as a variable T in a cell), calculates and prints out the correct altitude in meters, kilometers, and miles, with one decimal place for each result.

Output for 1a: When the code cell for this part of the problem is entered, it should specify (in the comments or the Markdown cell above) what units of T are assumed. The program should print out the correct altitude in meters, kilometers, and miles, with one decimal place for each result.

[1]:
```
import numpy as np
PI = np.pi  #the numpy library has a lot of useful mathematical constants

def find_altitude(T):
    #"""This function calculates and prints the altitude above Earth's surface
    #for a satellite with orbital period T (input in seconds)"""

    G = 6.67e-11 #Gravitational constant, units: m^3/kg/s^2
    M = 5.97e24  #mass of Earth, units: kg
    R = 6371e3    #radius of Earth, units: m

    h = ((G*M*(T)**2)/(4*PI**2))**(1/3)-R
    meters = h / 1000
    kilometers = h
    miles = h * 0.621371
    print ("The altitude would be {0:1.1f} in meters, {1:1.1f} in kilometers,␣
 ↪and {2:1.1f} in miles".format(meters, kilometers, miles) )
    return  h
find_altitude(6000)
```

```
The altitude would be 763.2 in meters, 763238.8 in kilometers, and 474254.5 in
miles
```

[1]:
```
763238.8150119493
```

Output for 1b and 1c: Use code cells to carry out the desired calculations, and Markdown cells to present and discuss your results.

1b. Use your program to calculate the altitudes of satellites that orbit the Earth once a day (so-called "geosynchronous" orbit), once every 90 minutes, and once every 45 minutes. What do you conclude from the last of these calculations?

[2]:
```
find_altitude(5400)

find_altitude(2700)

# I can conclude from my calculations that the longer the time, the higher the␣
 ↪altitude;
# this is trend with all of the inputs I've tried.
```

```
The altitude would be 279.3 in meters, 279321.6 in kilometers, and 173562.4 in
miles
The altitude would be -2181.6 in meters, -2181559.9 in kilometers, and
-1355558.1 in miles
```

[2]:
```
-2181559.8978108233
```

1c. Technically a geosynchronous satellite is one that orbits the Earth once per *sidereal day*, which is 23.93 hours, not 24 hours. Why is this? And how much difference will it make to the altitude of the satellite?

I think this is becuase the satellite appears to stay in the same spot, in a sense, it serves to keep both the satellite and earth facing each othe properly.

Hints Be very careful of units! Unlike wolframalpha.com, python is oblivious to what units your various numbers represent. You need to carefully keep track of them to make sure that your units work out. Not converting units properly can lead to very expensive disasters or just wrong scientific results …

Remember to comment your code and use descriptive variable names so others (and future you) can understand what you're doing!

1.2 Problem 2: Perfect Hardboiled Eggs

[Adapted from Langtangen, Exercise 1.12. You may derive the formula in Physics 112 or Physics 89] As an egg cooks, the proteins first denature and then coagulate. When the temperature exceeds a critical point, reactions begin and proceed faster as the temperature increases. In the egg white the proteins start to coagulate for temperatures above 63 C, while in the yolk the proteins start to coagulate for temperatures above 70 C. For a soft boiled egg, the white needs to have been heated long enough to coagulate at a temperature above 63 C, but the yolk should not be heated above 70 C. For a hard boiled egg, the center of the yolk should be allowed to reach 70 C.

The following formula expresses the time t it takes (in seconds) for the center of the yolk to reach the temperature T_y (in Celsius degrees):

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \ln\left[0.76 \frac{T_0 - T_w}{T_y - T_w}\right].$$

Here, M, ρ, c, and K are properties of the egg: * M is the mass, * ρ is the density, * c is the specific heat capacity, and * K is the thermal conductivity.

Relevant values are * $M = 64$ g for a large egg (USA size XL: en.wikipedia.org/wiki/Chicken_egg_sizes), * $\rho = 1.0378$ g cm^{-3}, * $c = 3.7$ J g^{-1} K^{-1}, and * $K = 5.4 \cdot 10^{-3}$ W cm^{-1} K^{-1}.

Furthermore, * T_w is the temperature (in C degrees) of the boiling water, and * T_0 is the original temperature (in C degrees) of the egg before being put in the water.

Suppose we want our eggs hard-boiled. Implement the formula in a program, set $T_w = 100$ C and $T_y = 70$ C, and compute t for a large egg taken from the fridge ($T_0 = 4$ C) and from room temperature ($T_0 = 20$ C). Also compute the results for a small egg ($M = 42$ g).

Output for 2: When you run your code cell, it should produce the following text, with your numbers instead of the TTT, MMM, and SSS placeholders:

```
To hard-boil a large egg taken directly from the fridge, cook it for TTT minutes (MMM min, SSS
To hard-boil a small egg taken directly from the fridge, cook it for TTT minutes (MMM min, SSS
To hard-boil a large egg starting from room temperature, cook it for TTT minutes (MMM min, SSS
To hard-boil a small egg starting from room temperature, cook it for TTT minutes (MMM min, SSS
```

The TTT placeholders should be values in minutes to two decimal places. The MMM and SSS placeholders should be rounded to the nearest minute/second, with no decimal places. For example,

```
To hard-boil a large egg taken directly from the fridge, cook it for 56.78 minutes (56 min 47 
```

[3]:
```python
import math
import numpy as np
import datetime
PI = np.pi  #the numpy library has a lot of useful mathematical constants
```

```python
def find_time(T_initial, M_egg):

    T_water = 100 #temperature of boiling water, units: degrees Celsius
    T_yolk  = 70  #temperature of cooked yolk, units: degrees Celsius
    rho = 1.0378  #density of egg, units: grams/cm^3
    c = 3.7       #specific heat capacity, units: Joules/grams/(degrees Kelvin)
    K = 5.4e-3    #thermal conductivity, units: Watts/cm/(degrees Kelvin)

    n = (M_egg**(2/3))*c*(rho**(1/3))
    v = K*(PI**2)*((4*(PI))/3)**(2/3)

    time = ((n)/(v))*math.log(.76*((T_initial - T_water)/(T_yolk - T_water)))

    min = time

    print ("To hard-boil a large egg taken directly from the fridge, cook it␣
↪for {0:1.1f} minutes ({1:2.1f} min/sec).".format(time, min ))
    print ("To hard-boil a small egg taken directly from the fridge, cook it␣
↪for {0:1.1f} minutes ({1:2.0f} min/sec).".format(float(time), float(min)))
    print ("To hard-boil a large egg starting from room temperature, cook it␣
↪for {:1.1f} minutes ({:2.0f} min/sec).".format(float(time), float(min)))
    print ("To hard-boil a small egg starting from room temperature, cook it␣
↪for {:1.1f} minutes ({:2.0f} min/sec).".format(float(time), float(min)))
    return (time)
find_time(4, 7)
#print ("To hardboil a small egg taken directly from the fridge, cook it for␣
↪time_egg minutes (MMM min, SSS sec)"
```

```
To hard-boil a large egg taken directly from the fridge, cook it for 88.0
minutes (88.0 min/sec).
To hard-boil a small egg taken directly from the fridge, cook it for 88.0
minutes (88 min/sec).
To hard-boil a large egg starting from room temperature, cook it for 88.0
minutes (88 min/sec).
To hard-boil a small egg starting from room temperature, cook it for 88.0
minutes (88 min/sec).
```

```
[3]: 87.96590982848521
```

Hints Writing the entire formula in one line is difficult to type, difficult to read, difficult to debug—and you have to retype the entire calculation just to change one piece of it. Try breaking it down in smaller chunks assigned to variables, and combine those chunks to produce the final result.

Beware of integer division!

Remember to comment your code and use descriptive variable names so others (and future you) can understand what you're doing!

1.3 Problem 3: Estimating Half-Life

[Adapted from Ayars, Problem 0-3] The data in file Ba137.txt is actual data from a radioactive decay experiment (you should already have the file from the Workshop). The first column is the number of decays N, the second is the time t in seconds. We'd like to know the half-life $t_{1/2}$ of ^{137}Ba. It should follow the decay equation

$$N = N_0 e^{-\lambda t}$$

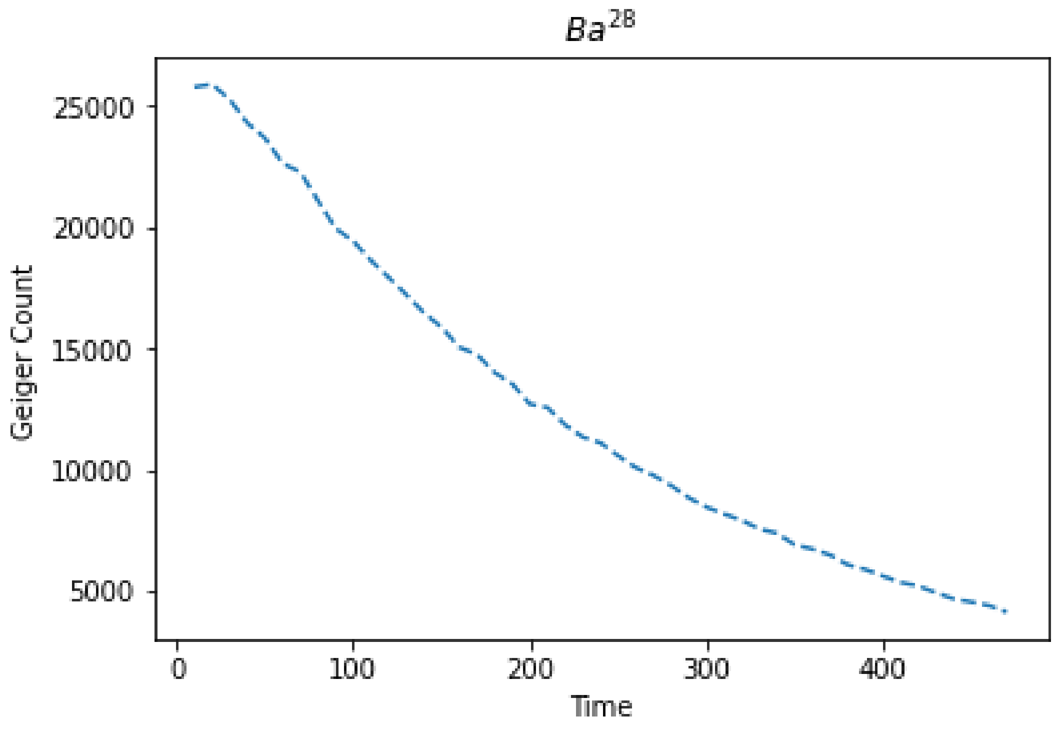
where $\lambda = \frac{\log 2}{t_{1/2}}$. Using the techniques you've learned from the lecture and workshop, load the data from the file Ba137.txt into appropriately-named variables. Experiment with different values of N and λ and plot the resulting equation on top of the data. (Python uses `exp()` to calculate the exponential function: i.e. `y = A*exp(-L*time)`) Don't worry about automating this process yet (unless you *really* want to!) just try adjusting things by hand until the equation matches the data pretty well. What is your best estimate for $t_{1/2}$?

Output for 3: When you run your code cell, it should produce a well-labeled plot with both the data and your curve of best fit. It should also print a message to the terminal which says, "My best estimate for the half life is x", where x is your estimate with units.

[9]:
```
import numpy as np
import matplotlib.pyplot as plt

# Starting by reading in the data from the text file
counts, times = np.loadtxt("Ba137.txt", unpack = True)

def lamda(times):
    k_value = np.log(2)/times
    return k_value

def half_life(counts):
    N = counts*np.exp(-lamda(times))
    return N

y = half_life(counts)

x = times

plt.plot(x, y, '--')

plt.title('${Ba}^{28}$')

plt.xlabel('Time')

plt.ylabel('Geiger Count')

plt.figure()

plt.show
```

```
print("I think an accurate estimate of the half life for Barium is about 200␣
 ↪seconds or so. ")
# 'counts' and 'times' are both Python lists; you can try printing to get an␣
 ↪idea of what values they contain

# The numpy library has an exponential function; use np.exp(x) to return e^x
```

```
I think an accurate estimate of the half life for Barium is about 200 seconds or
so.
```



```
<Figure size 432x288 with 0 Axes>
```

Hints Remember to comment your code and use descriptive variable names so others (and future you) can understand what you're doing!

[6]:
```
type(counts)
```

[6]:
```
numpy.ndarray
```

[7]:
```
print(times)
```

```
[ 10.      19.999  30.      39.999  50.      59.999  70.      80.      89.999
 100.     109.999 120.     129.999 140.     150.     159.999 170.     179.999
 190.     199.999 210.     220.     229.999 240.     249.999 260.     269.999
 280.     290.     299.999 310.     319.999 330.     339.999 350.     360.
 369.999 380.     389.999 400.     409.999 420.     430.     439.999 450.
 459.999 470.    ]
```