homework4

February 1, 2024

Introduction to Computer Programming for the Physical Sciences

Joseph F. Hennawi

Winter 2024

Open a new Jupyter notebook

Name your notebook with your name and Homework 1

Open a Markdown cell at the top and write your name and Homework 1

Open a Markdown cell before each problem and write e.g. Problem 1, Problem 2(a), etc.

Please abide by the Policy and Guidelines on Using AI Tools

Once you finish the problems: 1) Restart the Python kernel and clear all cell outputs. 2) Rerun the notebook from start to finish so that all answers/outputs show up. 3) Save your notebook as a single .pdf file and upload it to Gradescope on Canvas by the deadline. No late homeworks will be accepted except for illness accompanied by a doctor's note.

For parts of problems that require analytical solutions you can perform your calculations using a pencil and paper. Then photograph your work and convert the photograph to a .pdf file using an online tool. Homework assignments can only be submitted as a single .pdf file, so you will also need to figure out how to concatenate your photo .pdf file and your notebook .pdf file into a single .pdf file that you can submit. Online websites can do this for you. Alternatively, you can code up the analytical solution to your problems in a notebook Markdown cell using the LaTeX mathematical rendering language. This is harder but a chatbot can help you learn it.

1 Homework 4

1.1 Problem 1: Benford's Law

Benford's law refers to the observation that in many real-life datasets, the frequency of the first digits of each number is not uniform. If the digits were distributed uniformly, they would each occur as the first digit with equal probability, i.e. 1/9 per digit, or 11.1% of the time. Instead, there tend to be many more numbers with the first significant (decimal) digit of 1 as compared to 9. This applies best to datsets where the numbers span several orders of magnitude. Familiarize yourself with Benford's law by reading the Wikipedia page here. Incidentally, Benford's law is used in forensic accounting to detect fraud, see here.

Download the file census_data.csv at this link: https://github.com/enigma-igm/Phys29/blob/main/Phys29/homework/HW4/data/census_data.csv This is a comma-

separated values or **csv** file containing data from the United States Census Bureau. It lists the number of inhabitants for every town in the USA. As we discussed in the Week4 lecture on Reading and Writing Files, a CSV file is like a simple spreadsheet, where cells in each row are separated by a specific delimiter (in this case, a comma). As is indicated in the first header line of the CSV, the census data has three columns: State,Town,Population.

Note that Problem 2 will require you to reuse some of the code that you develop for this problem. Make sure that you write general functions for this problem that you can then use for both problems.

a) Write a Python program to *brute force* read the census data file, i.e., only using native Python file handling and string parsing commands. Store the data that you read in in lists **You are not allowed to import any Python packages into your notebook for executing this part of the problem.**

b) Now read the census data file using the `pandas` package.

c) Write code to verify that the contents of the dataset read in via step **a)** and step **b)** are exactly the same for all elements of the dataset (i.e. the State, Town, and Population columns are the same for all rows).

d) Extract the most significant digit from the Population column data for each town. For example for `California,Santa Barbara city,86353`, the most significant digit of the Population is 8. Store the most significant digits for each town in a `numpy` array. This array should have integer type and shape `shape=(ntown,)` where `ntown` is the number of towns in the dataset.

e) Construct a numpy array of shape `(9,)` that contains the integer number of times that each of the most significant digits 1-9 appears in the dataset. For this part of the problem, you are allowed to loop over the numbers (1,2,...9), **but no other loops are allowed!!, i.e. you cannot loop over the towns in the dataset** Hint: Use `numpy` Boolean indexing.

f) Write a program to plot a histogram of the **number of times** that each significant digit appears. The x-axis should be the digits (1,2,...9), and the y-axis should be the number of times that each digit appears. Plot the expectation for the case where the distribution across the digits is uniform, i.e. 11.1% per digit, as a horizontal line on the same plot. I realize we have not covered plotting yet in the lectures, but you can use Google or AI to assist you.

g) Using your results from **e)** and **f)** print to the screen the **percentage** of the time that each of the most significant digits 1-9 appears in the dataset.

h) Do the populations in this dataset obey Benford's law? Explain your reasoning.

i) Make a histogram of the town Populations using the following code:

```
from matplotlib import pyplot as plt
log10_bins = np.logspace(np.log10(np.min(populations)), np.log10(np.max(populations)), 100)
plt.hist(populations, bins=log10_bins)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Population')
plt.ylabel('Number of Towns')
plt.show()
```

Note that we have chosen to our histogram bins to be linearly spaced in the $\log_{10}$ of the population and made our histogram plot with a log scale on the x-axis to better illustrate the full dynamic range of the populations. We similarly chose a log scale on the y-axis to illustrate the large range of the number of cities in each histogram bin.

Based on the *distribution* of populations illustrated by the histogram, comment on why or why not you would expect the population data to obey Benford's law.

1.2 Problem 2: Munich Temperatures

Download the file munich_temperatures.txt at this link: https://github.com/enigma-igm/Phys29/blob/main/Phys29/homework/HW4/data/munich_temperatures.txt This is a text file containing the temperature (averaged over a 24 hour day) in Munich, Germany for each day between 1995-2013. The dates are represented by a floating point number where the integer part is the year and the decimal part is the fraction of the year. In other words, 1995.75 would be three quarters into the year 1995, or 10/01/1995. The average temperatures are in degrees Celsius.

a) Write a Python program to read this file using the `pandas` package. Assign the date and temperature columns of the `pandas DataFrame` to two arrays of shape `(ndays,)` where `ndays` is the number of days in the dataset.

b) The temperature data contains bad values. Analyze the temperatures and figure out how to identify which values are the bad ones (remember these are temperatures in degrees Celsius not Fahrenheit!). Use `numpy` boolean indexing to create two new date and temperature `numpy` arrays that contain only the good data.

c) Using the good data from part **b)**, compute the average (i.e. use `numpy.mean`) temperature over the entire time period covered by the dataset and print the value to the screen. **You are not allowed to perform any loops in this part of the problem.**

d) Compute the minimum, maximum and average temperature as a function of the year for each year 1995, 1996, ..., 2012 (ignore 2013 since it is incompletely covered). **You are allowed to loop over the years** (i.e. `range(1995,2013)`), **but no other loops are allowed**. Print the minimum, maximum, and average temperature for each year to the screen.

e) Convert the good temperatures in the Munich temperatures dataset onto the *absolute temperature* scale in degrees Kelvin (**no loops allowed!**).

f) Repeat Problem1(e) but for the good temperatures in the Munich temperature dataset **in degrees Kelvin** using the functions that you wrote for Problem 1. **Do not repeat code, i.e. use the general functions that you wrote in Problem 1 to solve this part and the rest of the parts of this Problem below.**

g) Repeat Problem1(f) but for the good temperatures in degrees Kelvin using the functions that you wrote for Problem 1.

h) Repeat Problem1(g) but for the good temperatures in degrees Kelvin using the functions that you wrote for Problem 1.

i) Do the daily temperatures in Munich in degrees Kelvin obey Benford's law? Explain your reasoning.

j) Make a histogram of good temperatures in degrees Kelvin as in Problem1(i). Think about the

best choices for the histogram bins (i.e. logarthmic or linear) and the scale of the x-axis (i.e. linear or logarithmic) given the dynamic range of this data.

Based on the *distribution* of temperatures, comment on why or why not you would expect the temperature data to obey Benford's law.

1.3 Problem 3: Array Slicing

Download the data slicing.txt at this link: https://github.com/enigma-igm/Phys29/blob/main/Phys29/homework/HW4/data/slicing.txt. This is a text file containing a 2D numpy array of `shape=(6,6,)`. Read the data file into a `numpy` array using `np.loadtxt`. See the Week4 lecture for an example of how to use `np.loadtxt`.

The colors in the image below indicate different sub-arrays that can be extracted from the main array. For each of the colors (sub-arrays), obtain the new numpy sub-array with a single `numpy` command using the slice/stride syntax. Print the shape of the sub-arrays and the contents of the sub-arrays to the screen to verify that your results are correct.

1.4 Problem 4: Coordinate Manipulation with Numpy

We can repesent the Cartesian coordinates $\mathbf{r_i} = (x_i, y_i, z_i)$ for four particles in 3D as a numpy arrays `positions`

[4]:
```
import numpy as np
positions = np.array([[0.0,0.0,0.0],[1.5,1.5,0.0],[10.5,0.0,-2.3],[0.0,1.3,1.
↪3]])
s = np.array([-3.6,-1.0,-1.3])
```

Above we also defined `s`, which is a translation vector, **s**, in 3D space. Both the coordinates and the translate vector are in units of meters. **Note: In what follows we use the zero-based indexing convention to refer to the particles, i.e. the zeroth particle is the one with coordinates** `[0.0, 0.0, 0.0]` and there is no fourth particle.

a) What is the shape of the array `positions` and what is its dimension. Print both to the screen.

b) What is the shape of the array `s` and what is its dimension. Print both to the screen.

c) How do you access the coordinates of the third particle in `positions`? Print its coordinates to the screen.

d) Assign the third particles z coordinate to a variable named `p3_z` and print it to the screen. What type of Python object is `p3_z`? What are the shape and dimension of `p3_z`? Print these to the screen as well.

e) How do you access the x coordinates of all the particles in `positions`? Assign them to a variable named `positions_x` and print values, shape, and dimensions of `positions_x` to the screen.

f) Write python function to perform a translation of the coordinates in `positions` by the translation vector `s` consistent with the docstring below:

```
def translate(pos, s):
    """
```

```
    Translate the coordinates in pos by the translation vector s.

    Parameters
    ----------
    pos : numpy.ndarray
        A numpy array of shape (n,3) where n is the number of particles and the 3 columns
        are the x,y,z coordinates of the particles in m.
    s : numpy.ndarray
        A numpy array of shape (3,) that represents the translation vector in m.

    Returns
    -------
    translated_pos: numpy.ndarray
        A numpy array of shape (n,3) where n is the number of particles and the 3 columns are
        the x,y,z coordinates of the particles in m after the translation.
    """
```

g) Test your function from part **f)** by translating the coordinates in `positions` by the translation vector `s` and print the translated coordinates to the screen. Also test your code on the new set of coordinates and translation vector:

```
position_2 = np.array([[1.5, -1.5, 3], [-1.5, 1.5, -3]])
s2 = np.array([-1.5, 1.5, 3])
```

and print the results to the screen.

1.5 Problem 5: The Riemann Zeta Function, Numpy Edition

In the Week3 lecture on loops, we showed how one can evaluate the Riemann Zeta function defined as

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

a) We stated that it can be proved mathematically that

$$\zeta(2) = \frac{\pi^2}{6}$$

or equivalently

$$\pi = \sqrt{6\zeta(2)} = \sqrt{6 \sum_{n=1}^{\infty} \frac{1}{n^2}}.$$

Use the `numpy` `arange`, `sum` and `sqrt()` functions to estimate π from the Riemann Zeta function and print your result to the screen. Use the first 1000 terms in the sum. Compute the relative error of your estimate of π with respect to the true value of π (which you can access via `np.pi`) and print to the screen.

b) Write a function to evaluate the Riemann Zeta function using `numpy` for any value of s consistent with the docstring below. **Note that your solution should only use numpy functions. No Python loops allowed anywhere!!**

```
def zeta(s, nterms=1000):
    """
    Evaluate the Riemann Zeta function for any value of s using the first nterms terms in the

    Parameters
    ----------
    s : np.ndarray
        The values of s for which to evaluate the Riemann Zeta function. shape = (ns,)
    nterms : int, optional
        The number of terms to use in the sum. Default is 1000.

    Returns
    -------
    zeta_s: np.ndarray
        The value of the Riemann Zeta function for the input values of s using the first nterm
        of zeta_s is the same as s, i.e. shape = (ns,).
    """
```

Test your function on the array `s=np.arange(1,11)`, i.e. `zofs = zeta(s)` and print the result (`print(zofs)`) and the shape of the result (`print(zofs.shape)`) to the screen.