Caitlyn Jones

Prof. Thomas Fai

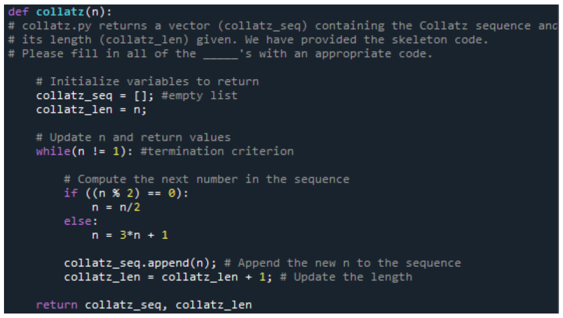
Math 122A: Numerical Methods and Big Data

10 September 2024

Homework 1

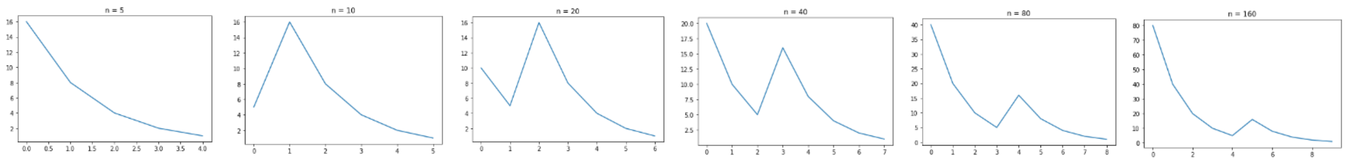
<u>Problem 1:</u>

A) In order to successfully complete this problem, we need to be able to understand the written instructions and convert them to a running python function. In this function we will need the following: a parameter that takes the value of n, a while loop that only terminates when n = 1, an if statement that runs when the n is even, and an else statement that runs when n is odd. Given the skeleton code for this problem, the following is the completed code with the requirements from the instructions translated into Python.



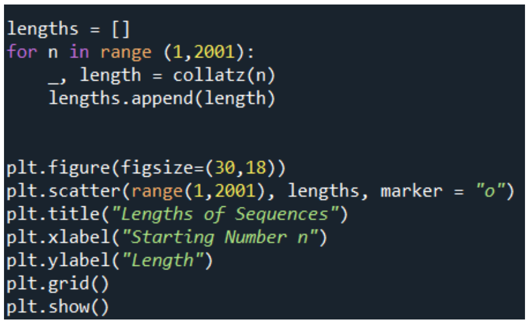
```python
def collatz(n):
# collatz.py returns a vector (collatz_seq) containing the Collatz sequence an
# its length (collatz_len) given. We have provided the skeleton code.
# Please fill in all of the _____'s with an appropriate code.

    # Initialize variables to return
    collatz_seq = []; #empty list
    collatz_len = n;

    # Update n and return values
    while(n != 1): #termination criterion

        # Compute the next number in the sequence
        if ((n % 2) == 0):
            n = n/2
        else:
            n = 3*n + 1

        collatz_seq.append(n); # Append the new n to the sequence
        collatz_len = collatz_len + 1; # Update the length

    return collatz_seq, collatz_len
```

B) After plotting the graphs for n = 5, 10, 20, 40, 80, 160, I noticed a pattern amongst them.

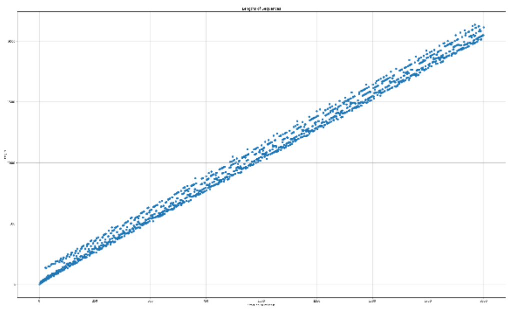


Above are the graphs for the various values of n. Putting them in sequential order, you can clearly see that each graph is an extension of the one before. Also, we can see that the peak of the graph is at 16, regardless of the n value. We can use this knowledge and the fact that each graph is an extension to predict that the final graph will be an exponential function as the n value gets progressively larger.

C) Yes there is pattern. The graph is a positively increasing linear function. In order to plot this, I needed to add lines of code that would scatter plot the n values from 1 to 2000.



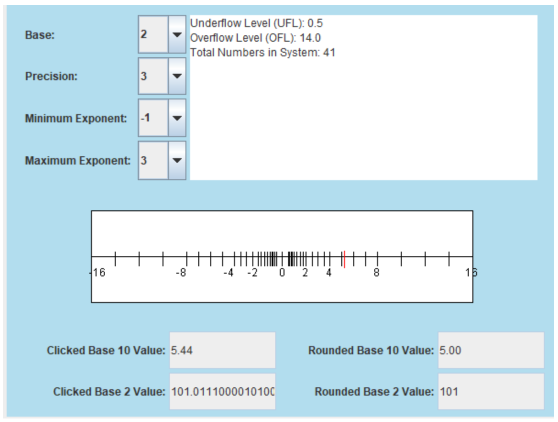
```
lengths = []
for n in range (1,2001):
    _, length = collatz(n)
    lengths.append(length)


plt.figure(figsize=(30,18))
plt.scatter(range(1,2001), lengths, marker = "o")
plt.title("Lengths of Sequences")
plt.xlabel("Starting Number n")
plt.ylabel("Length")
plt.grid()
plt.show()
```



Problem 2:

UFL = 0.5; OFL = 14.0; $\epsilon_{\text{mach}} = .125$

$$\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p} = \frac{1}{2}(2)^{1-3} = .125$$



Base: 2

Precision: 3

Minimum Exponent: -1

Maximum Exponent: 3

Underflow Level (UFL): 0.5
Overflow Level (OFL): 14.0
Total Numbers in System: 41

Clicked Base 10 Value: 5.44

Rounded Base 10 Value: 5.00

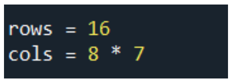
Clicked Base 2 Value: 101.0111000010100

Rounded Base 2 Value: 101

There are 41 total numbers in this system.

Problem 3:

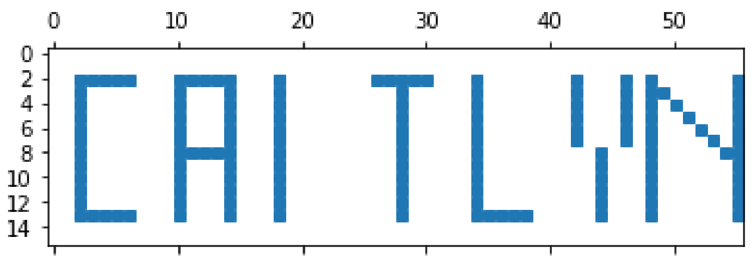
I decided to use my name as the design for the binary matrix. To start, I needed to determine the size of my matrix. Since my name has 7 letters, n = 7, making my matrix 16 X 8*7 or 16 X 56.



```
rows = 16
cols = 8 * 7
```

Next, I needed to actually create my matrix by drawing the lines of my name on this grid. My input and output are as follows:

```
matrix = np.zeros((rows, cols), dtype=int)
matrix[2:14, 2] = 1
matrix[2, 2:7] = 1
matrix[13, 2:7] = 1
matrix[2:14, 10] = 1
matrix[2:14, 14] = 1
matrix[2, 10:15] = 1
matrix[8, 10:15] = 1
matrix[2:14, 18] = 1
matrix[2, 26:31] = 1
matrix[2:14, 28] = 1
matrix[2:14, 34] = 1
matrix[13, 34:39] = 1
matrix[2:8, 42] = 1
matrix[2:8, 46] = 1
matrix[8:14, 44] = 1
matrix[2:14, 48] = 1
matrix[2:14, 55] = 1
for i in range(7):
    matrix[3 + i, 49 + i] = 1
```



Finally, I needed to save my name created in matrix form in binary number form.

```
plt.spy(matrix, markersize=5)
plt.show()
matrix = np.savetxt('cjones_name.txt', matrix, fmt = "%d")
```

Problem 4:

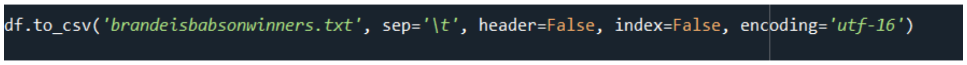
A) For this problem, I used the package panda in order to complete this problem without using any loops. To start, I had to read the brandeisbabson.txt file into the terminal and split up each row into its respective columns.

```
import pandas as pd

df  = pd.read_csv("brandeisbabson (1).txt", sep = "\t", header = None, encoding = "utf-16")
```

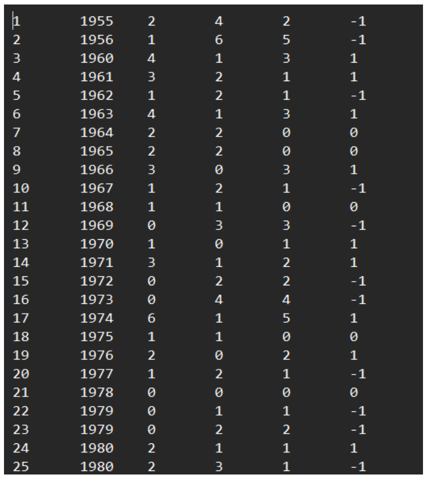
Next, I needed a way of checking the values in the third and fourth columns. The absolute value of the difference of these values will be used to create the fifth column. Also, the values in the third and fourth columns will also help determine the values in the sixth column. If the value in column three is larger than the value in column four, the value in column six will be 1. If the value in column three is smaller than the value in column four, the value in column six will be -1. If they are the same, the value in column six will be 0.

```
df[4] = (df[2] - df[3]).abs()
df[5] = (df[2] > df[3]).astype(int) - (df[2] < df[3]).astype(int)
```

Lastly, I needed to save the new columns along with the original text file into a new text file, brandeisbabsonwinners.txt.

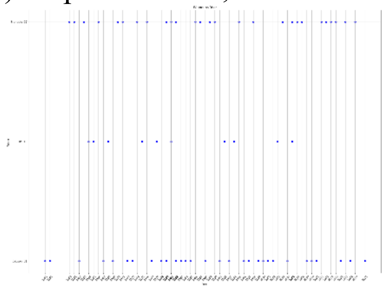


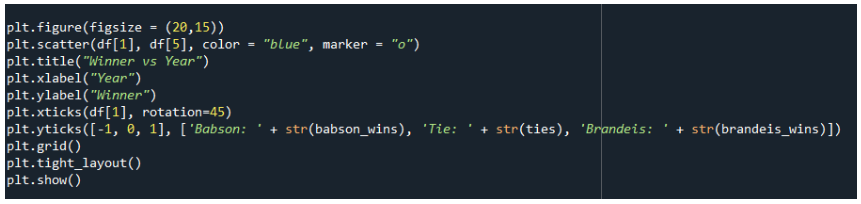
```
df.to_csv('brandeisbabsonwinners.txt', sep='\t', header=False, index=False, encoding='utf-16')
```



```
1       1955    2       4       2       -1
2       1956    1       6       5       -1
3       1960    4       1       3       1
4       1961    3       2       1       1
5       1962    1       2       1       -1
6       1963    4       1       3       1
7       1964    2       2       0       0
8       1965    2       2       0       0
9       1966    3       0       3       1
10      1967    1       2       1       -1
11      1968    1       1       0       0
12      1969    0       3       3       -1
13      1970    1       0       1       1
14      1971    3       1       2       1
15      1972    0       2       2       -1
16      1973    0       4       4       -1
17      1974    6       1       5       1
18      1975    1       1       0       0
19      1976    2       0       2       1
20      1977    1       2       1       -1
21      1978    0       0       0       0
22      1979    0       1       1       -1
23      1979    0       2       2       -1
24      1980    2       1       1       1
25      1980    2       3       1       -1
```

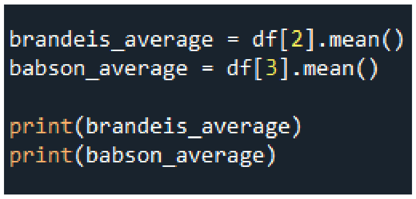
B) To plot the data, I had the x-axis be the year, and the y-axis be the team.



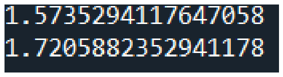


```
plt.figure(figsize = (20,15))
plt.scatter(df[1], df[5], color = "blue", marker = "o")
plt.title("Winner vs Year")
plt.xlabel("Year")
plt.ylabel("Winner")
plt.xticks(df[1], rotation=45)
plt.yticks([-1, 0, 1], ['Babson: ' + str(babson_wins), 'Tie: ' + str(ties), 'Brandeis: ' + str(brandeis_wins)])
plt.grid()
plt.tight_layout()
plt.show()
```

C) Brandeis won 27 games. Babson won 31 games. In order to find the average of how many points Brandeis and Babson scored, I needed to find a way to add up the values in columns 3 and 4 and find the average.

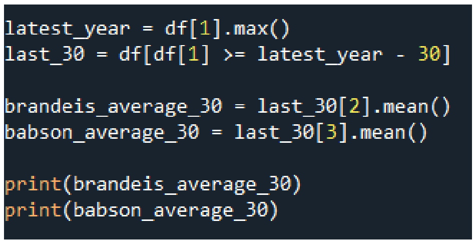


```
brandeis_average = df[2].mean()
babson_average = df[3].mean()

print(brandeis_average)
print(babson_average)
```

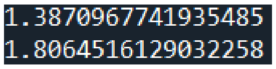


```
1.5735294117647058
1.7205882352941178
```

D) In order to find the average of Brandeis and Babson points only from 1992 to 2021, I can modify the above code to fit the requirements. I added a few lines that will get the data from the last 30 years and compute the average of points scored by Brandeis and Babson.



```
latest_year = df[1].max()
last_30 = df[df[1] >= latest_year - 30]

brandeis_average_30 = last_30[2].mean()
babson_average_30 = last_30[3].mean()

print(brandeis_average_30)
print(babson_average_30)
```



```
1.3870967741935485
1.8064516129032258
```