Homework 1: Mathematical Preliminaries
CPE 490/590 ST

Instructor: Rahul Bhadani

Due: January 22, 2025, 11:59 PM
100 points

You are allowed to use a generative model-based AI tool for your assignment. However, you must submit an accompanying reflection report detailing how you used the AI tool, the specific query you made, and how it improved your understanding of the subject. You are also required to submit screenshots of your conversation with any large language model (LLM) or equivalent conversational AI, clearly showing the prompts and your login avatar. Some conversational AIs provide a way to share a conversation link, and such a link is desirable for authenticity. Failure to do so may result in actions taken in compliance with the plagiarism policy.

Additionally, you must include your thoughts on how you would approach the assignment if such a tool were not available. Failure to provide a reflection report for every assignment where an AI tool is used may result in a penalty, and subsequent actions will be taken in line with the plagiarism policy.

Submission instruction:

Submission instruction for this homework supersedes one mentioned in the Syllabus.

This homework requires all answers recorded in a single `.ipynb` Python notebook. You can use a combination text cell (i.e. markdown formatted cell) and code cell to provide your answer. To add equations you should be able to use Latex syntaxs in the text cells of your Python notebook. As a part of your submission, you must provide executed notebook with code, text, and outputs. Alternatively, you can also provide a url (whose permission you must change to 'anyone with link can view') of your Python notebook from Google Colab. The naming convention for your notebook should follow the format {firstname.lastname}_CPE 490/590 ST_hw01.ipynb. For example, if your name is Sam Wells, your file name should be sam.wells_CPE 490/590 ST_hw01.ipynb.

Python Practice

Note: Python community has adopted following import convention that we will follow for the rest of the course:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels as sm
```

1 The Longest Substring (10 Points)

Write a function `LongestSubstring` in Python that takes a Python string as an argument. Your function should perform the following task:

1. If the string is empty print 'empty string' and return.
2. If the string length exceeds 20 characters, print 'maximum length exceeded' and return.
3. Otherwise, find the long substring in the given string that contains same characters. For example, if the input string is '2242777232888823', then the output should be '8888'.
4. Write code to use the function `LongestSubstring` as well.

The implementation doesn't need to be the most efficient implementation, and you may choose to use nested `for` loops as well.

Answer

```python
    def LongestSubstring(s):
    # Check if the string is empty
    if not s:
        print("empty string")
        return

    # Check if the string length exceeds 20 characters
    if len(s) > 20:
        print("maximum length exceeded")
        return

    max_substring = ""
    current_substring = ""

    # Iterate through the string to find
    # the longest substring with the same characters
    for i in range(len(s)):
        if i == 0 or s[i] == s[i - 1]:
            current_substring += s[i]
        else:
            if len(current_substring) > len(max_substring):
                max_substring = current_substring
            current_substring = s[i]

    # Final check in case the longest
    # substring is at the end of the string
    if len(current_substring) > len(max_substring):
        max_substring = current_substring

    # Print the result
    print(max_substring)
# Example usage
LongestSubstring("2242777232888823")  # Output: '8888'
LongestSubstring("")              # Output: 'empty string'
LongestSubstring("1" * 24)         # 'maximum length exceeded'
LongestSubstring("aaabbbbaacccdd")   # Output: 'bbbb'
```

2 Data and Time in Python (5 Points)

The built-in Python `datetime` module provides `datetime`, `date`, and `time` types. The `datetime` type combines the information stored in date and time.

Write a Python program that creates an empty file with the file name in the format ML-HW01_%Y-%m-%d-%H-%M.txt, where %Y is substitute for current year, %m is the substitute for current month, %d is substitute for current day, %H is the substitute for current hour, and %M is the substitute for current minute. One possibility can be a file name **ML-HW01-2011-10-29-20-30.txt**.

Answer

```
import datetime

# Get the current date and time
time_now = datetime.datetime.now()



# Formatting to required filename
required_filename = time_now.strftime("ML-HW01-%Y-%m-%d-%H-%M.txt")

# This creates an empty file with the required filename
with open(required_filename, "w") as file:
   pass

print(f"{required_filename} is the empty file created.")
```

3 Working with Data Types in Python (5 Points)

In Python, there are some special data types such as table, and lists. A tuple is a fixed-length, immutable sequence of Python objects which, once assigned, cannot be changed. An example is `(2, 4, 5)`. On the other hand, lists are variable length and their contents can be modified in place. An example is `[1, 4, 5]`. Write a Python function that takes an argument as a list or a tuple, and returns a tuple containing the number of element the list or a tuple has, the maximum value, and a minimum value.

Answer

```python
# x should be a list or tuple input
def lenmaxmin(x):

    # Calculating the length, maximum, and minimum value of the list/tuple
    num_elements = len(x)
    maxi = max(x)
    mini = min(x)

    # This returns the value as tuple
    return (num_elements, maxi, mini)


print(lenmaxmin((12,6,4,65)))
print(lenmaxmin([311,45,9]))
print(lenmaxmin([311,45,9, 74, 10222, 89, -93]))
```

4 Slicing (5 Points)

You can select sections of most sequence types by using slice notation, which in its basic form consists of `start:stop` passed to the indexing operator `[]`:

In this part of the homework, you will create a 4×4 tensor using PyTorch package. Initialize your tensor randomly with only 0s and 1s. Tensor is a generalized name of a multi-dimensional array or matrix in machine learning domain. Note that this concept of tensor differs from one seen in Mathematics textbooks. Write a Python program to slice the last column of a 4×4 tensor.

Answer

```python
import torch

# This creates a random tensor of size 4 x 4 consisting of 1s and 0s
rand_tensor = torch.randint(0, 2, (4, 4))
   # or tensor[:,3]. This slices the last column of the 4 x 4 tensor
last_column = rand_tensor[:, -1]
# convert to a column vector
last_column = last_column.reshape(-1, 1)
print(f"The tensor is:\n{rand_tensor}\n")
print(f"The last column is:\n{last_column}")
```

5 Dictionary (5 Points)

Dictionary is a key-value pair data-types in Python. Create a dictionary in Python consisting of two letter state code of the neighboring states of Alabama as the key and their capital as the value.

Answer

```python
#Dictionary of Alabama's neighbouring states'
# two letter code as keys and the states' capital as value
AL_neighbouring_states = {"MS": "Jackson", "GA":"Atlanta",
   "FL":"Tallahassee", "TN":"Nashville"}
print(AL_neighbouring_states)
print(AL_neighbouring_states["FL"])
print(AL_neighbouring_states["MS"])
print(AL_neighbouring_states["GA"])
print(AL_neighbouring_states["TN"])
```

Linear Algebra Practice

6 Linear Dependence (5 Points)

Let

$$A = \{(1,2,3),(2,3,4),(3,4,5)\} \tag{1}$$

$$B = \{(1,2,3),(2,3,1),(3,1,2)\} \tag{2}$$

Write the system of equations to test the linear independence of A and B and solve them using `sympy` package from Python.

Answer

Linear dependence is reduced to

$$\begin{aligned} x + 2y + 3z &= 0 \\ 2x + 3y + 4z &= 0 \\ 3x + 4y + 5z &= 0 \end{aligned} \tag{3}$$

for A,

$$\begin{aligned} x + 2y + 3z &= 0 \\ 2x + 3y + z &= 0 \\ 3x + y + 2z &= 0 \end{aligned} \tag{4}$$

for B.

```python
from sympy import solve
from sympy.abc import x, y, z
A = solve([x + 2*y + 3*z, 2*x + 3*y + 4*z, 3*x + 4*y + 5*z], [x, y, z])
print (A)

from sympy import solve
from sympy.abc import x, y, z
B = solve([x + 2*y + 3*z, 2*x + 3*y + z, 3*x + y + 2*z], [x, y, z])
print (B)
```

Output:

```
{x: z, y: -2*z}
{x: 0, y: 0, z: 0}
```

7 Diagonal Matrix (5 points)

Write a function `ToDiagonal` in Python that accepts a square matrix (declared as numpy array of size $m \times m$) and returns an equivalent diagonal matrix. You also need to write code-snippet that uses the function `ToDiagonal`.

Answer

```python
import numpy as np

# The input for the function should be a square matrix
def ToDiagonal(m):

   # This will extract the diagonal elements of input matrix
   diag_ele = np.diag(m)

   # A diagonal matrix will be created
   # using the extracted diagonal elements
   diag_mat = np.diag(diag_ele)

   return diag_mat

example_mat = np.array([[1,2,3],[10,121,12],[0.5,0.6,0.3]])

print("Original Matrix:")
print(example_mat)

diagonal_matrix = ToDiagonal(example_mat)

print("\n The Diagonal Matrix:")
print(diagonal_matrix)
```

8 Solving a Matrix Equation (5 Points)

We have

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix} \tag{5}$$

and

$$B = \begin{bmatrix} a & b \\ c & d \\ 2 & 1 \end{bmatrix} \tag{6}$$

such that

$$AB = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7}$$

Use `sympy` Python package to write python code to compute the value of a, b, c and d.

Answer

```
from sympy import Matrix, solve, eye
from sympy.abc import a, b, c, d
A = Matrix([[1, 2, 3], [2, 3, 4]])
B = Matrix([[a, b], [c, d], [2, 1]])
ans = solve(A*B - eye(2), [a, b, c, d])
print(ans)
```

9 Adjoint and Transpose (5 Points)

Using `numpy` Compute the transpose and adjoint of A where

$$A = \begin{bmatrix} 1+2j & 2+3j & 3+4j \\ 2+3j & 3+4j & 4+5j \end{bmatrix}. \tag{8}$$

Probability & Statistics Practice

10 Disjoint Sets (10 points)

If $P(A) = \frac{1}{3}$ and $P(B^c) = \frac{1}{4}$, can A and B be disjoint? Explain.

Answer

If A and B were disjoint, $P(A \cup B) = P(A) + P(B) = 1/3 + 3/4 = 13/12$, which is impossible. More generally, if A and B were disjoint, then $A \subseteq B^c$ and $P(A) \leq P(B^c)$. But here $P(A) > P(B^c)$, so A and B cannot be disjoint.

11 Coin Toss (10 points)

Write a Python program to simulate the tossing of a coin. After 100 tosses what is the probability of getting a head? What about after 1000 tosses? What about after 10000 tosses?

Answer

```python
import random

def simulate_tosses(n):
    # Simulate n tosses
    tosses = [random.choice(['Head', 'Tail']) for _ in range(n)]

    # Count the number of heads
    num_heads = tosses.count('Head')

    # Calculate the probability
    probability = num_heads / n

    return probability

# Simulate 100, 1000, and 10000 tosses
for n in [100, 1000, 10000, 100000]:
    probability = simulate_tosses(n)
    print(f'After {n} tosses, the probability of getting a head is {probability:.4f}')
```

Output:

```
After 100 tosses, the probability of getting a head is 0.4500
After 1000 tosses, the probability of getting a head is 0.4970
After 10000 tosses, the probability of getting a head is 0.5019
After 100000 tosses, the probability of getting a head is 0.5001
```

Note:
1. Your answer may vary.
2. I added 100000 tosses as well to demonstrate the convergence to 0.5.

12 Three coins toss and CDF (10 points)

Consider the experiment of tossing three fair coins, and let X = number of heads observed. The CDF of X is

$$F_X(x) = \begin{cases} 0, & -\infty < x < 0 \\ \frac{1}{8}, & 0 \leq x < 1 \\ \frac{1}{2}, & 1 \leq x < 2 \\ \frac{7}{8}, & 2 \leq x < 3 \\ 1, & 3 \leq x < \infty \end{cases} \tag{9}$$

Write a Python program to draw a graph $F_X(x)$ using step functions. Determine $F_X(2.5)$ using Python.

Answer

```
import numpy as np
import matplotlib.pyplot as plt

# The CDF of X
def F_X(x):
    if x < 0:
        return 0
    elif 0 <= x < 1:
        return 1 / 8
    elif 1 <= x < 2:
        return 1 / 2
    elif 2 <= x < 3:
        return 7 / 8
    else:
        return 1

# x and y values for plotting
x_vals = np.linspace(-1, 4, 1000)
y_vals = [F_X(x) for x in x_vals]

# plotting the graph
plt.step(x_vals, y_vals, label=r'$F_X(x)$')
plt.title('CDF of X - Three Fair Coin Tosses')
plt.xlabel('x')
plt.ylabel(r'$F_X(x)$')
plt.grid()
plt.legend()
plt.show()



result = F_X(2.5)
print(f"Fx(2.5): {result}")
```

13 Probability Mass Functions: Poisson Distribution (10 points)

Refer to the Python Notebook provided at the end of Lecture 03.

Write a Python program to generate 10000 random samples that follow a Poisson distribution. You should choose values of λ as 0.25, 1.0, and 3.0. Plot the probability mass function for each the case overlaying on the same graph. Choose colors in the graph wisely so as to make the visualization understandable. Comment on how changing λ changes the shape of the probability mass function.

Answer

1. Note that calculating factorial directly for a large number is not possible, hence overall evaluation of the function is done by taking log and Gamma function.
2. From the drawn samples you will notice that as x increases, y approaches zero. Hence, we will only plot points up to $x = 20$ to visualize.
3. As we increase the value of λ, the peak of the probability density function spreads out. For the smaller value of λ, the distribution is skewed to the right.
4. If λ is small, then rare events (those with a low probability of occurrence) have a relatively high probability under the Poisson distribution. As λ increases, the probability of these rare events decreases.

```
import math
from scipy.stats import rv_discrete
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.family'] = 'Serif'
import scipy.special

L = 0.25
n = 10000
x1k = np.arange(n)
```

```
y1k = np.zeros(n)
for i, x in enumerate(x1k):
    # Calculate the expression using
    # the logarithm of the factorial because for large x
    # it is not possible to directly calculate factorial(x)
    y1k[i] = np.exp(-L + x * np.log(L) - scipy.special.gammaln(x + 1))

y1k =y1k/sum(y1k)
pmf1 = rv_discrete(name='Poisson', values=(x1k, y1k))

L = 1.0
n = 10000
x2k = np.arange(n)
y2k = np.zeros(n)
for i, x in enumerate(x2k):
    # Calculate the expression using
    # the logarithm of the factorial because for large x
    # it is not possible to directly calculate factorial(x)
    y2k[i] = np.exp(-L + x * np.log(L) - scipy.special.gammaln(x + 1))

y2k =y2k/sum(y1k)
pmf2 = rv_discrete(name='Poisson', values=(x2k, y2k))

L = 3.0
n = 10000
x3k = np.arange(n)
y3k = np.zeros(n)
for i, x in enumerate(x3k):
    # Calculate the expression using
    # the logarithm of the factorial because for large x
    # it is not possible to directly calculate factorial(x)
    y3k[i] = np.exp(-L + x * np.log(L) - scipy.special.gammaln(x + 1))

y3k =y3k/sum(y3k)
pmf3 = rv_discrete(name='Poisson', values=(x3k, y3k))

plt.rcParams['font.family'] = 'Serif'
plt.rcParams['font.size'] = 15


fig, ax = plt.subplots(1, 1)
ax=np.ravel(ax)

ax[0].plot(x1k, pmf1.pmf(x1k))
```

```
ax[0].plot(x2k, pmf2.pmf(x2k))
ax[0].plot(x3k, pmf3.pmf(x3k))
ax[0].legend(['$\lambda=0.25$', '$\lambda=1.0$', '$\lambda=3.0$'])
ax[0].set_xlim([0, 20])
```

Alternative method

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

# The lambda values
lmdas = [0.25, 1.0, 3.0]


plt.figure(figsize=(10, 6))

for i in range(len(lmdas)):
   l = lmdas[i]

   # Generating 10000 samples from a Poisson distribution with a given lambda
   rand_sam = np.random.poisson(l, 10000)

   # This sets the range of K values or occurrences that will be used for plotting
   x = np.arange(0, max(rand_sam) + 1)

   # Generating the probability mass function
   prob_val = poisson.pmf(x, l)


   plt.plot(x, prob_val, marker='o', linestyle='-', label=f' = {l}')


plt.title('Poisson Distribution for Different  Values')
plt.xlabel('Number of Events (k)')
plt.ylabel('Probability')
plt.legend()
plt.grid()
plt.show()
```

14 Probability Density Functions: Normal Distribution (10 points)

Refer to the Python Notebook provided at the end of Lecture 03. Write a Python program to generate 10000 random samples following a Normal distribution. Keep $\mu = 0.5$ fixed and vary the standard deviation $\sigma = 0.3$, 5.0, and 10.0. Plot the probability density function using histogram and kernel density estimation (KDE) plot. Comment on your observation about changing the standard deviation and its effect on the shape of the probability density function.

Answer

We see that as the standard deviation σ is increased, the peak reduces and the probability density function spreads.

```
import math
from scipy.stats import rv_discrete
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.family'] = 'Serif'
import seaborn as s
from scipy.stats import norm

B1  = norm.rvs(loc=0.5, scale=0.3, size=10000)
B2  = norm.rvs(loc=0.5, scale=5.0, size=10000)
B3  = norm.rvs(loc=0.5, scale=10.0, size=10000)

fig, ax = plt.subplots(1, 1, figsize=(10,5))
ax=np.ravel(ax)



s.histplot(B1, ax = ax[0], stat='density', kde=True,
label='$\sigma = 0.3$, $\mu = 0.5$')
s.histplot(B2, ax = ax[0], stat='density', kde=True,
label='$\sigma = 5.0$, $\mu = 0.5$')
s.histplot(B3, ax = ax[0], stat='density', kde=True,
label='$\sigma = 10.0$, $\mu = 0.5$')

ax[0].set_xlim([-25, 25])
ax[0].set_xlabel('x',  fontsize = 20)
ax[0].set_ylabel('f(x)', fontsize =20)
ax[0].set_title('Gaussian, $\mu = 0.5$, $\sigma$ varies', fontsize =20)
```

```
ax[0].legend()
```

Alternative method

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# defining the mean and standard deviations
mean = 0.5
stan_devs = [0.3, 5.0, 10.0]


plt.figure(figsize=(10,5))


for s in stan_devs:
    # Generating the random samples from a normal
    # distribution of specified mean and normal distribution
    rand_sam = np.random.normal(loc=mean, scale=s, size=10000)

    # Plotting the probability density function
    # with histogram and kernel density estimation
    sns.histplot(rand_sam, kde=True, stat = 'density',
                bins=100, label=f"s = {s}")


plt.title("Probability Density Function for Varying Standard Deviations")
plt.xlabel("x")
plt.ylabel("Density")
plt.legend(title="Standard Deviation")
plt.grid()
plt.show()
```