CS108 - Software Systems Lab

Lab 9 - Advanced Python Programming

Student: Aditya Sanapala, 23b0912@iitb.ac.in
Lecturer: Kameswari Chebrolu, chebrolu@cse.iitb.ac.in

Problem 1: The Matrix

A young programmer named Python yearned for freedom. He dreamed of breaking through the simulated reality that bound humanity, but the path was fraught with challenges.

Python knew that to unravel the Matrix, he needed speed and precision in his computations. Struggling with the limitations of his native tools, he turned to NumPy, a powerful ally renowned for its lightning-fast array operations. With the help of NumPy, Python successfully escaped "The Matrix". Let's see if you can do it too!

You are given a numpy array. You need to perform various operations on it, as detailed below.
The matrix given is

```
[[5, 5, 84, 3, 9],
[6, 11, 1, 55, 58],
[1, 20, 48, 12, 36],
[8, 4, 41, 93, 98],
[6, 17, 64, 0, 13]]
```

All expected outputs are given for this input matrix. Do not hardcode the outputs, otherwise they will fail for almost all other matrices!

Task 1: Return the transpose of the upper triangular matrix including the diagonal of the input matrix.
Testing: Uncomment the code in the main function corresponding to testing of Task 1.

```
>> print(task1(matrix))
```

Expected Output:

```
[[5, 0, 0, 0, 0],
[5, 11, 0, 0, 0],
[84, 1, 48, 0, 0],
[3, 55, 12, 93, 0],
[9, 58, 36, 98, 13]]
```

Task 2: Print the mean, median and standard deviation (all along x-axis), determinant and inverse of the matrix.
You need to print the inverse if the determinant is non-zero.
If the determinant is zero, there is a pseudo-inverse in NumPy (Moore Penrose Pseudo Inverse) which needs to be printed instead.
For the standard deviation, determinant and inverse, keep the precision as 2 decimals (check

the `around` function in NumPy).
Testing: Uncomment the code in the main function corresponding to testing of Task 2.

```
>> mean, median, std, det, inv, pseudoinv = task2(matrix)
>> print("Mean:  ", mean)
>> print("Median:  ", median)
>> print("Standard Deviation:  ", std)
>> print("Determinant:  ", det)
>> print("Inverse:  ", inv)
>> print("Pseudo-Inverse:  ", pseudoinv)
```

Expected Output: We will check precision of digits only upto 3-4 decimal places.
Mean:

```
[ 5.2, 11.4, 47.6, 32.6, 42.8]
```

Median:

```
[ 6., 11., 48., 12., 36.]
```

Standard Deviation:

```
[ 2.31516738, 6.34350061, 27.60144924, 36.0921044, 32.72552521]
```

Determinant:

```
1821201.0000000014
```

Inverse:

```
[[ -0.58373129, -0.52849521, -0.26407794, 0.36545554, 0.73834354],
[ 0.22828782, 0.28887256, 0.07659781, -0.18808632, -0.24109695],
[ 0.05239894, 0.034866, 0.01394959, -0.02410991, -0.04871016],
[ 0.31980819, 0.33957372, 0.06275584, -0.2084844, -0.33856065],
[ -0.28707979, -0.30548358, -0.04695912, 0.19598221, 0.29123364]]
```

Pseudo-Inverse:

```
[[ -0.58373129, -0.52849521, -0.26407794, 0.36545554, 0.73834354],
[ 0.22828782, 0.28887256, 0.07659781, -0.18808632, -0.24109695],
[ 0.05239894, 0.034866, 0.01394959, -0.02410991, -0.04871016],
[ 0.31980819, 0.33957372, 0.06275584, -0.2084844, -0.33856065],
[ -0.28707979, -0.30548358, -0.04695912, 0.19598221, 0.29123364]]
```

Task 3: Padding is a common operation in image processing.
You will be given an integer `n` and you need to pad (with value 0), in top, bottom, right and left, which will lead to an increase in the row and column size by $2 \times n$.
Essentially, this means that if you have a matrix of size $N \times M$, and you want to pad the matrix with dimension n and value 0, the matrix would be changed to size $(n+N+n) \times (n+M+n)$, where the n top and n bottom rows, as well as n left and n right columns will be 0.
The matrix would be at the center, and will still be of size $N \times M$.
Testing: Uncomment the code in the main function corresponding to testing of Task 3.

```
>> print(task3(matrix)) # default padding with number value 0 and dimension
3
```

Expected Output:

```
[[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 5, 5, 84, 3, 9, 0, 0, 0],
[ 0, 0, 0, 6, 11, 1, 55, 58, 0, 0, 0],
[ 0, 0, 0, 1, 20, 48, 12, 36, 0, 0, 0],
[ 0, 0, 0, 8, 4, 41, 93, 98, 0, 0, 0],
[ 0, 0, 0, 6, 17, 64, 0, 13, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Some Helpful References:

1. Statistics
2. Matrix Operations
3. Padding

Problem 2: Distributions

NumPy is also used widely for the various and commonly occurring mathematical and probabilistic distributions and sampling. Combined with Pyplot from Matplotlib, they give a good API for data analysis and interpretation.

Your task would be to sample each of the below six distributions $S = 1000000$ times and then plot the frequency histograms for these samples. You need to create a single figure that has 6 subplots which should be arranged as 1, 2 in the first row, 3, 4 in second row and 5 and 6 in the third row (so 3×2 sub-plot).

For sampling each of them, the input size should be set to S. Larger the sample size, better the estimation of the distribution. You can go through the documentation reference (provided in the links) to find the appropriate parameter to set in the NumPy functions.

Some clarifications for the terminology used below:

1. The parameters (`a, b, scale, loc`, etc) all refer to the statistical variables that are used to adjust the shape of the distributions (check the documentation and the formulae for them for more details as all these distributions are used very commonly).
2. The `values` are the outputs obtained from the distributions, so there will be S values for each of the distributions, and you need to scale them as indicated(by scaling we mean, multiplying by the factor as mentioned).

3. The `range` and `step` are both for the x-axis of the plots, and indicate the range of input, so the histogram will essentially tell you how many of the samples fell in which range.

4. Apart from these, we also mention the visual appeal of each of the subplots. Read through the documentation to figure out how to achieve these.

Every subplot needs to have a title corresponding to the distribution being plotted as a histogram.

The Distributions:

1. Beta. $a = 4$, $b = 20$. Multiply the values by 100. Range -5 to 50. Step 1. Color of histogram is red. Title = "Beta"

2. Exponential. scale = 0.1. Multiply the values by 100. Range -1 to 50. Step 1. Color of histogram is green with 0.5 blending factor. Title = "Exponential"

3. Gamma. $scale = 0.1$, $shape = 2$. Multiply the values by 100. Range -1 to 50. Step 1. Color of histogram is black with a blending factor of 0.8 but orientation is horizontal. Title = "Gamma"

4. Laplace. scale = 0.5, loc = 0. Multiply the values by 100. Range -1 to 50. Step 1. Color is orange. Title = "Laplace"

5. Normal (Gaussian). $\mu = 0$, scale = 3. Range -10 to 11. Step 1. Default color. Title = "Normal"

6. Poisson. $\lambda = 3$. Range -1 to 11. Step 1. Default color. Title = "Poisson"

Expected output for this is shown in `expected_plot.png`.

Note:

1. You may notice that the autograder is stochastic and possibly may give different marks on different runs. To make this deterministic, set a seed before you do the sampling from distributions.

2. The autograder may not always be correct. It may give you full score even when partially correct. You need to verify your `generated_plot.png` by comparing it against `expected_plot.png`.

Some Helpful References:

1. Legacy Random Generation

2. `matplotlib.pyplot.subplot`

3. `matplotlib.pyplot.hist` (Check `**kwargs` for blending factor)

4. `numpy.random.seed`

Problem 3: Lisan al-Gaib (imagine Javier Bardem)

The House of Harkonnens, the great spice miners of Arrakis faced a grave dilemma. The extraction of the precious spice, Melange, was fraught with uncertainty. The spice blooms in remote regions, hidden beneath the endless dunes, and its distribution was irregular and unpredictable.

Legend has it that an old wise woman, inspired by the prophecy of the Lisan al-Gaib, devised a method to unveil the spice's secret locations. She believed that the spice's distribution held patterns, invisible to the naked eye but decipherable through the whispers of data.

Her method, known as the Lisan al-Gaib Algorithm, became a beacon of hope for the spice miners. This algorithm takes as input the available locations where traces of spice were found(called spice points), and finds out potential locations of spice nodes(or spice centers). Every spice center has a cluster of spice points. Or conversely, every spice point belongs to a cluster of a spice center. The algorithm is explained below.

Fill in the to-dos of the `spice.py` file.

When done with all to-dos, you need to visualize the final results of clustering of spice nodes. Run the `run.py` script. You can change the `data_path` and value of K (number of clusters) and observe the output clustering.

You are given with `kmeans_1.png` (`data_path = "spice_locations.txt'`, $K = 2$) and `kmeans_2.png` (`data_path = "spice_locations2.txt"`, $K = 4$)

Note:

1. Do not change anything other than where asked for filling the to-dos.
2. You need to achieve the required output without using loops. To check no usage of loops, we shall run a naive approach of parsing files to search. Each additional for/while will incurs a penalty of -2. So do not use them even within comments!

Lisan al-Gaib Algorithm:
Input:

1. Dataset containing 2D coordinates of locations where traces of spice were found
2. K (guess of number of spice nodes)

Output:
Potential locations of K spice centers/nodes

Algorithm:

1. For each of the spice point, maintain a label (0 to $K - 1$, corresponding to the cluster of the spice center it belongs to).
2. Initialize with a random guess for the coordinates of K spice centers, choose them to be randomly K of the spice locations themselves.

3. Loop:

   (a) E Step: (Keeping the coordinates of the K spice centers fixed, update the labels)
   - For each spice point, find the nearest spice center, and assign the spice point to the cluster of that spice center.
   - To do this, for each spice point, we first compute distance(here, euclidean) to all the spice centers and assign the label corresponding to that spice node which has the shortest of all distances.

   (b) M Step: (Keeping the labels of spice points fixed, update the coordinates of the spice centers)
   - The new coordinate of the ith spice center is the mean of coordinates of allspice points belonging to the cluster with label $= i$

   (c) Repeat Loop until the labels don't change.

Some Helpful References:

1. Array Creation Routines
2. Logic Functions
3. Broadcasting
4. Linear Algebra
5. Random Generation