Homework 1 Report

Ayhan Okuyan
ayhan.okuyan[at]ug.bilkent.edu.tr

March 1, 2020

Contents

Question 1

This question asks us to solve a series of questions about a linear system of equations $Ax = b$ given below in matrix form below that is assumed to be the computation of the weighted linear combination of a neural population.

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & 1 & -1 & 5 \\ 3 & 3 & 0 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix} \tag{1}$$

Question requires us to first analytically solve the questions by hand and the confirm the results using computational techniques. The choice of programming language for this question and assignment is Python.The library imports that are used throughout are given below.

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

Part A This part requires to solve the linear system of equations given as $Ax = 0$. Hence, we apply the row operations as given below to obtain the reduced row echelon form of the matrix. Below given the applied operations and their results.

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & 1 & -1 & 5 \\ 3 & 3 & 0 & 9 \end{bmatrix} \xrightarrow{R3 = R3 - 3R1} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & 1 & -1 & 5 \\ 0 & 3 & 3 & 3 \end{bmatrix} \xrightarrow{R2 = R2 - 2R1} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 3 & 3 \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 3 & 3 \end{bmatrix} \xrightarrow{R3 = R3 - 3R2} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

The result obtained in (3) is the row echelon form of the matrix. Here, we observe that while we have two fixed variables in x_1 and x_2, we also have two free variables as will be referred as α and β. Then we can rewrite the system of linear equations as follows.

$$x_1 - \alpha + 2\beta = 0 \tag{4}$$
$$x_2 + \alpha + \beta = 0 \tag{5}$$

Hence, we can find x_1 and x_2 in terms of α and β and write the solution of the system.

$$x_1 = \alpha - 2\beta \tag{6}$$
$$x_2 = -(\alpha + \beta) \tag{7}$$
$$x_3 = \alpha \tag{8}$$
$$x_4 = \beta \tag{9}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \alpha - 2\beta \\ -(\alpha+\beta) \\ \alpha \\ \beta \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix}, \alpha, \beta \in \mathbb{R} \tag{10}$$

Since, we have analytically found the solution, we can prove it is the correct solution by assigning random values for α and β and putting the vector through the equations and showing the results provide a zero vector.

```
alpha = np.random.rand()
beta = np.random.rand()
x = np.asarray([alpha-2*beta, -alpha-beta, alpha, beta])
result = np.matmul(A,x)
print('Result of Ax', result)
```

which gives the result,

```
Result of Ax [[0. 0. 0.]]
```

Part B This part requires us to find a particular solution to $Ax = b$. In order to achieve this, we have provided the general solution to the linear system of equations since finding all of the solution is asked in Part C, which we will show the derivations in that part. The solution of the system can be given as follows.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1+\alpha - 2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \alpha, \beta \in \mathbb{R} \tag{11}$$

For the particular solution, we have chosen the coefficients as $\alpha = 5, \beta = 10$. The solution vector then becomes as follows.

$$x_p = \begin{bmatrix} -14 \\ -13 \\ 5 \\ 10 \end{bmatrix} \tag{12}$$

To verify this is a valid solution, we repeat the previous step with the newly obtained solution and the defined parameter values and observe the output of Ax which should give the b vector as result.

```
alpha = 5
beta = 10
x = np.asarray([1+alpha-2*beta, 2-alpha-beta, alpha, beta])
result = np.matmul(A,x)
print('Result of Ax', result)
```

which gives the following result, letting us confirm that this is a valid particular solution for the system.

```
Result of Ax [[1 4 9]]
```

Part C This part, as mentioned in Part B, requires us to find the general solution to the linear system of equations $Ax = b$. In order to do that, we reduce the augmented matrix $[A|b]$ to its row echelon form, whose steps are given below.

$$\begin{bmatrix} 1 & 0 & -1 & 2 & 1 \\ 2 & 1 & -1 & 5 & 4 \\ 3 & 3 & 0 & 9 & 9 \end{bmatrix} \xrightarrow[\text{R3=R3-3R1}]{\text{R2=R2-2R1}} \begin{bmatrix} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 3 & 3 & 3 & 6 \end{bmatrix} \tag{13}$$

$$\begin{bmatrix} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 3 & 3 & 3 & 6 \end{bmatrix} \xrightarrow{R3 = R3 - 3R2} \begin{bmatrix} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{14}$$

Here, using this augmented matrix, we can rewrite this system of equations by defining two free variables as we have in Part A.

$$x_1 - \alpha + 2\beta = 1 \tag{15}$$
$$x_2 + \alpha + \beta = 2 \tag{16}$$
$$\tag{17}$$

Hence, we can write the each component of x as follows.

$$x_1 = 1 + \alpha - 2\beta \tag{18}$$
$$x_2 = 2 - \alpha - \beta \tag{19}$$
$$x_3 = \alpha \tag{20}$$
$$x_4 = \beta \tag{21}$$

Which can be written in parametric form as follows, equal to Equation (11).

$$x = \begin{bmatrix} 1 + \alpha - 2\beta \\ 2 - \alpha - \beta \\ \alpha \\ \beta \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \alpha, \beta \in \mathbb{R} \tag{22}$$

In order to control if this solution is the true solution, we assign random numbers and observe the outputs of Ax if it is equal to b, which is done by the following Python code segment.

```python
alpha = np.random.rand()
beta = np.random.rand()
x = np.asarray([1+alpha-2*beta, 2-alpha-beta, alpha, beta])
result = np.matmul(A,x)
print('Result of Ax', result)
```

The output of the code is given below, which is exactly same with the given b vector.

```
Result of Ax [[1. 4. 9.]]
```

Part D Here, apart from solving the system, the question asks us to find the pseudo-inverse of the matrix A or the Moore-Penrose inverse. This inverse concept generalizes the inverse concept which is applicable for square matrices with non-zero determinants. The main reasoning behind the inverse operation can be presented to solve a system of linear equations in the form $Ax = b$. We can see that if inverse exists, we can multiply both sides of the equation with A^{-1} from the left and reach $A^{-1}Ax = A^{-1}b$. Since from definition of inverse we know that $A^{-1}A = I$, we see the result $x = A^{-1}b$. Hence, if inverse exists, we can simply multiply it with b in order to find the solution. Also, the regular approach for the non-invertible matrices is to multiply it with its transpose as given in the following sequence.

$$A^TAx = A^Tb \tag{23}$$

$$(A^TA)^{-1}A^TAx = (A^TA)^{-1}A^Tb \tag{24}$$

$$x = (A^TA)^{-1}A^Tb \tag{25}$$

This situation holds true only if $(A^TA)^{-1}$ exists. Here, we can now also convey that $(A^TA)^{-1}A^T$ as the definition of pseudo-inverse for full-column rank matrices. To check is existence, we need to ensure that the matrix A^TA has non-zero determinant, which we will investigate.

$$A^TA = \begin{bmatrix} 14 & 11 & -3 & 39 \\ 11 & 10 & -1 & 32 \\ -3 & -1 & 2 & -7 \\ 39 & 32 & -7 & 110 \end{bmatrix} \tag{26}$$

Here, we observe that the there is a linear dependency between the rows 1, 2 and 3 since row 3 is the difference of row 1 and row 2. We can immediately conclude that the determinant is zero. Hence, we move to another method where we find the pseudo-inverse via using Singular Value Decomposition (SVD).

SVD is a widely used technique, which enables any matrix to be decomposed into a diagonal (Σ), and two square matrices (U, V) with orthonormal column vectors.The decomposition can be observed below.

$$A_{mxn} = U_{mxm}\Sigma_{mxn}V^T_{nxn} \tag{27}$$

In order to find U and V, we solve the following equation set.

$$Av = \delta u \tag{28}$$

$$A^Tu = \delta v \tag{29}$$

In order to find the solution, we can multiply Equation (28) with A^T and Equation (29) with A from the left, and apply some algebra as given below to obtain two distinct eigenvalue problems.

$$Av = \delta u \tag{30}$$
$$A^T Av = \delta A^T u \tag{31}$$
$$(A^T A)v = \delta^2 v \tag{32}$$
$$A^T u = \delta v \tag{33}$$
$$AA^T u = \delta Av \tag{34}$$
$$(AA^T)u = \delta^2 u \tag{35}$$

Hence, we need to find the eigenvalues and the corresponding eigenvectors of AA^T and $A^T A$. Here, Λ is used for δ^2 for simplicity.

$$det(A^T A - \lambda I) = 0 \tag{36}$$

$$\begin{vmatrix} 14-\lambda & 11 & -3 & 39 \\ 11 & 10-\lambda & -1 & 32 \\ -3 & -1 & 2-\lambda & -7 \\ 39 & 32 & -7 & 110-\lambda \end{vmatrix} = 0 \tag{37}$$

$$(14-\lambda)\big(-266\lambda + 122\lambda^2 - \lambda^3\big) - 11\big(19\lambda + 11\lambda^2\big) - 3\big(-76\lambda + 3\lambda^2\big) - 39\big(-95\lambda + 39\lambda^2\big) = 0 \tag{38}$$

$$\big(\lambda^4 - 136\lambda^3 + 323\lambda^2\big) = 0 \tag{39}$$

The roots of the characteristic polynomial gives the following eigenvalues.

$$\lambda_1 = 0 \tag{40}$$
$$\lambda_2 = 2.418 \tag{41}$$
$$\lambda_3 = 133.582 \tag{42}$$

From there, we can continue and find the orthonormal eigenvectors and repeat the same procedure for AA^T, which will provide the same eigenvalues. Hence, the SVD components are found as follows.

$$U = \begin{bmatrix} 0.1898 & 0.7003 & 0.6882 \\ 0.4761 & 0.5476 & -0.6882 \\ 0.8586 & -0.458 & 0.2294 \end{bmatrix} \tag{43}$$

$$\Sigma = \begin{bmatrix} 11.5578 & 0 & 0 & 0 \\ 0 & 1.555 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{44}$$

$$V = \begin{bmatrix} 0.3217 & 0.2702 & -0.8165 & 0.3961 \\ 0.2641 & -0.5322 & -0.4082 & -0.6931 \\ -0.0576 & -0.8023 & 0 & 0.5941 \\ 0.9074 & 0.0082 & 0.4082 & 0.099 \end{bmatrix} \tag{45}$$

After finding these matrices, we can find the pseudo-inverse as below.

$$A^+_{nxm} = V_{nxn} \Sigma^+_{nxm} U^T_{mxm} \tag{46}$$

where, Σ^+ is defined as the transpose of the matrix in which we take the reciprocals of the non-zero elements in Σ. Hence, we find the pseudo-inverse with Equation (46) and the result is given.

$$A^+ = \begin{bmatrix} 0.126 & 0.108 & -0.056 \\ -0.235 & -0.176 & 0.176 \\ -0.362 & -0.285 & 0.232 \\ 0.019 & 0.040 & 0.065 \end{bmatrix} \tag{47}$$

In order to confirm this result, we will be running two separate codes, first we will be confirming the method through using SVD function of numpy to calculate the pseudo-inverse as we have analytically done and then we will be using numpy's pinv (pseudo-inverse) function and then we will be comparing similarity between our calculations and the acquired results.

```
u, sigma, v_t = np.linalg.svd(A)
sigma_ = np.zeros(A.shape)
fill_sig = 1/sigma
#to remove the computational errors (100 is arbitrary)
fill_sig[fill_sig >= 100] = 0
np.fill_diagonal(sigma_, fill_sig)
sigma_ = sigma_.T

psdA = np.matmul(np.matmul(v_t.T,sigma_), u.T)
print('Pseudo-inverse with SVD\n', psdA)

psdA_ = np.linalg.pinv(A)
print('Pseudo-inverse with pinv\n', psdA_)
```

and the result below shows that the method we used and the calculations we have done were correct.

```
Pseudo-inverse with SVD
 [[ 0.12693498  0.10835913 -0.05572755]
 [-0.23529412 -0.17647059  0.17647059]
 [-0.3622291  -0.28482972  0.23219814]
 [ 0.01857585  0.04024768  0.06501548]]
Pseudo-inverse with pinv
 [[ 0.12693498  0.10835913 -0.05572755]
 [-0.23529412 -0.17647059  0.17647059]
 [-0.3622291  -0.28482972  0.23219814]
 [ 0.01857585  0.04024768  0.06501548]]
```

Part E This part asks us to find the sparsest solution to the system $Ax = b$, meaning containing the most zeros in the solution vector. Finding sparsest solution to a system is normally defined as an NP-Hard problem, meaning there don't exist a problem that solves this in polynomial time. However, in our case, we can manually find the solutions, since we already know the solution to the system which depends on two free variables. In order to visualize the sparsest solution, we can use the general solution that is found through Part C and equate all of them to zero and plot their graphs on a alpha-beta named two dimensional coordinate axis.

$$1 + \alpha - 2\beta = 0 \tag{48}$$
$$2 - \alpha - \beta = 0 \tag{49}$$
$$\alpha = 0 \tag{50}$$
$$\beta = 0 \tag{51}$$

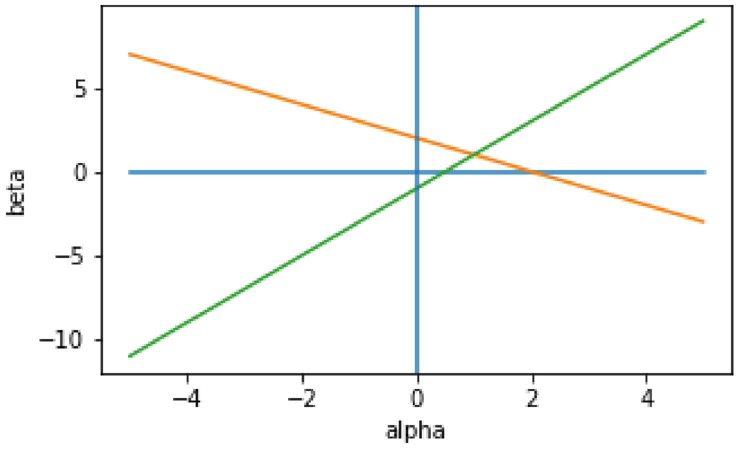


Figure 1: The linear parameter plots with respect to α and β.

Here, we observe that there are at most two lines intersecting each other in a single point. Here, we can conclude that there are six elements in the sparsest solution set, each solution containing two zero elements. Hence, now we can analytically find each one by setting only two from the elements to zero at a given instance.

(a) $(\alpha = 0,\ \beta = 0)$

$$x_1 = \begin{bmatrix} 1+\alpha-2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} \tag{52}$$

(b) $(\alpha = 0,\ \beta = 1/2)$

$$x_2 = \begin{bmatrix} 1+\alpha-2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ 3/2 \\ 0 \\ 1/2 \end{bmatrix} \tag{53}$$

(c) $(\alpha = 0,\ \beta = 2)$

$$x_3 = \begin{bmatrix} 1+\alpha-2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ 0 \\ 2 \end{bmatrix} \tag{54}$$

(d) $(\alpha = -1,\ \beta = 0)$

$$x_4 = \begin{bmatrix} 1+\alpha-2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ -1 \\ 0 \end{bmatrix} \tag{55}$$

(e) $(\alpha = 2,\ \beta = 0)$

$$x_5 = \begin{bmatrix} 1+\alpha-2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 2 \\ 0 \end{bmatrix} \tag{56}$$

(f) $(\alpha = 1,\ \beta = 1)$

$$x_6 = \begin{bmatrix} 1+\alpha-2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{57}$$

After finding six solutions, with the help of Figure 1, we can conclude that these solutions construct the set for the sparsest solutions for the linear system of equations $Ax = b$. In order to computationally conclude the validity of our solutions, we created a list of the solutions that we have obtained and found the result of Ax, in which all should be equal to the b vector.

```
sparse_soln_set = [(1,1), (0,0), (0,0.5), (0,2), (-1,0), (2,0)]
for alp,bet in sparse_soln_set:
    print('Alpha:', alp, 'Beta:', bet)
    x = np.asarray([1+alp-2*bet, 2-alp-bet, alp, bet])
    print('x:', x)
    result = np.matmul(A,x)
    print('b:', result[0])
```

The result is as follows.

```
Alpha: 1 Beta: 1
x: [0 0 1 1]
b: [[1 4 9]]
Alpha: 0 Beta: 0
x: [1 2 0 0]
b: [[1 4 9]]
Alpha: 0 Beta: 0.5
x: [0.  1.5 0.  0.5]
b: [[1. 4. 9.]]
Alpha: 0 Beta: 2
x: [-3  0  0  2]
b: [[1 4 9]]
Alpha: -1 Beta: 0
x: [ 0  3 -1  0]
b: [[1 4 9]]
```

```
Alpha: 2 Beta: 0
x: [3 0 2 0]
b: [[1 4 9]]
```

Part F This part of the question requires us to find the least-norm solution for $Ax = b$, which is the solution what would minimize the L2 Norm given as follows.

$$L_2(x) = ||x||_2 = \sqrt{\sum_i x_i^2} \tag{58}$$

In order to find such a solution, one can use the least-squares solution by simply finding the values for α and β that minimizes $||x||_2$. Hence, we find the algebraic expansion of $||x||_2$ and equate the partial derivatives to zero to find the optimal values.

$$||x|| = \sqrt{(1+\alpha-2\beta)^2 + (2-\alpha-\beta)^2 + \alpha^2 + \beta^2} \tag{59}$$

$$= \sqrt{3\alpha^2 + 6\beta^2 - 2\alpha\beta - 2\alpha - 8\beta + 5} \tag{60}$$

Since square-root is a monotonic operation, we can remove the square root and minimize $||x||_2^2$ instead.

$$||x||_2^2 = 3\alpha^2 + 6\beta^2 - 2\alpha\beta - 2\alpha - 8\beta + 5 = f(\alpha, \beta) \tag{61}$$

$$\frac{\partial f}{\partial \alpha} = 6\alpha - 2\beta - 2 = 0 \tag{62}$$

$$\alpha = \frac{\beta + 1}{3} \tag{63}$$

$$\frac{\partial f}{\partial \beta} = 12\beta - 2\alpha - 8 = 0 \tag{64}$$

$$\alpha = 6\beta - 4 \tag{65}$$

We can then use (63) and (65) to solve the system for α and β.

$$\beta + 1 = 18\beta - 12 \tag{66}$$

$$\beta^* = 13/17 \tag{67}$$

$$\alpha^* = 10/17 \tag{68}$$

We were able to find an extremum, however to check if this is a minimum, we should check the sign of the determinant of the Hessian of f, which is equivalent to the second derivative test.

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial \alpha^2} & \frac{\partial^2 f}{\partial \alpha \beta} \\ \frac{\partial^2 f}{\partial \beta \alpha} & \frac{\partial^2 f}{\partial \beta^2} \end{bmatrix} = \begin{bmatrix} 6 & -2 \\ -2 & 12 \end{bmatrix} \tag{69}$$

$$det(H) = 6 * 12 - (-2)^2 = 68 > 0 \tag{70}$$

Hence, we can conclude that this tuple of (α, β) values define a local minimum. Thus, we can write the least norm solution as follows.

$$x_{L_2} = \begin{bmatrix} 1+\alpha-2\beta \\ 2-\alpha-\beta \\ \alpha \\ \beta \end{bmatrix} \xrightarrow[\alpha=10/17]{\beta=11/17} \begin{bmatrix} 1/17 \\ 11/17 \\ 10/17 \\ 13/17 \end{bmatrix} = \begin{bmatrix} 0.0588 \\ 0.6475 \\ 0.5882 \\ 0.7647 \end{bmatrix} \tag{71}$$

However we can also use the pseudo-inverse of matrix A, since A^+b also gives the unique least-norm solution. Hence, we can simply multiply A^+ with b to find the least norm solution and we also use this procedure to observe if the least norm solution is correct.

$$A^+b = \begin{bmatrix} 0.126 & 0.108 & -0.056 \\ -0.235 & -0.176 & 0.176 \\ -0.362 & -0.285 & 0.232 \\ 0.019 & 0.040 & 0.065 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix} = \begin{bmatrix} 0.05882353 \\ 0.64705882 \\ 0.58823529 \\ 0.76470588 \end{bmatrix} \tag{72}$$

Here, we see that both of our solutions are correct and working. In order to computationally verify this, we use the following code.

```
least_norm_calc = [1/17, 11/17, 10/17, 13/17]
least_norm_psdinv = np.matmul(psdA, [1,4,9])
print('Least Squares Solution,\n', np.asarray(least_norm_calc))
print('Least norm by pseudo-inverse,\n', least_norm_psdinv)
```

```
Least Squares Solution,
 [0.05882353 0.64705882 0.58823529 0.76470588]
Least norm by pseudo-inverse,
 [[0.05882353 0.64705882 0.58823529 0.76470588]]
```

This results concludes this part and this question since they are all simply equal.

Question 2

This question is interested in the concept of 'Reverse Inference'. Although Broca's Area is generally associated with the concept of language, meaning that if Broca's Area is activated during a task, the researchers generally infer that the task is associate with language. However, after some research, it is found that,

- Broca's area was reported to be activated in 103 out of 869 fMRI tasks involving language but,
- this area was also active in 199 out of 2353 tasks not involving language.

Here, we are simply asked to solve some statistics related problems regarding these priors.The question is solved through Python 3 scripting and the used packages are provided below.

```
import numpy as np
import scipy.special as spc
import matplotlib.pyplot as plt
```

Part A This part requires us to compute the Maximum Likelihood Estimates (MLE) of activation frequencies conditioned on the language and not language which is denoted by x_l and x_{nl}, assuming that each trial is an independent Bernoulli random variable with probability $p = x_l$ and $p = x_{nl}$. Then, we are asked to compute these functions at $x = [0 : 0.001 : 1]$ and plot their respective bar charts. We know that multiple Bernoulli trials converge into a binomial distribution, hence, we can declare the Binomial distributions with estimated parameter as the probability of success.Here, *act* is used to denote the state *activation* = 1

$$P(act|p = x_l) = \binom{869}{103} x_l^{103}(1 - x_l)^{869-103} \tag{1}$$

$$P(act|p = x_{nl}) = \binom{2353}{199} x_l^{199}(1 - x_l)^{2353-199} \tag{2}$$

Hence, the MLE estimates are defined as,

$$\arg\max_{x_l} P(act|p = x_l) \tag{3}$$

$$\arg\max_{x_{nl}} P(act|p = x_{nl}) \tag{4}$$

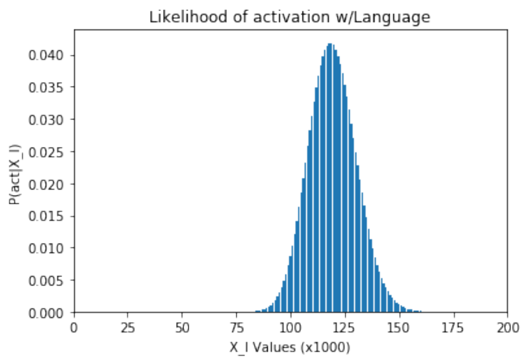
The probabilities are found through the helper function written which computes the probability when the success probability, the number of all trials and the number of successes are provided.

```
def binomialLikelihood(prob, all_ins, pos_ins):
    return spc.comb(all_ins, pos_ins) * prob**(pos_ins) * (1.0-
                                        prob)**(all_ins - pos_ins)
```

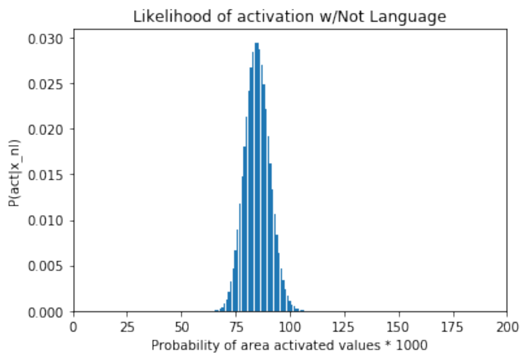
Hence, we use the following code snippets to find and plot the charts for the language and not language case.

```
prob_list = np.arange(0.0,1.0,0.001)
xl_list = []
for pr in prob_list:
    xl_list.append(binomialLikelihood(pr,869,103))
index = np.arange(0,1000)
plt.xlim(0,200)
plt.bar(index, xl_list)
plt.xlabel('Probability of area activated values * 1000')
plt.ylabel('P(act|x_l)')
plt.title('Likelihood of activation w/Language')
plt.show()

xnl_list = []
for pr in prob_list:
    xnl_list.append(binomialLikelihood(pr,2353,199))
plt.xlim(0,200)
plt.bar(index, xnl_list)
plt.xlabel('Probability of area activated values * 1000')
plt.ylabel('P(act|x_nl)')
plt.title('Likelihood of activation w/Not Language')
plt.show()
```

The chart is provided below.



(a) Probability of Activation with Language



(b) Probability of Activation with not Language

Figure 2: Activation Probabilities assuming Bernoulli Priors

Part B This part requires us to find the x_l and x_{nl} values that maximize their respective discretized likelihood functions. We simply find the value that gives the maximum activation.

```
print('Max Likelihood Estimate of P(act|x_l):', prob_list[np.
                                    argmax(xl_list)])
print('Max Likelihood Estimate of P(act|x_nl):', prob_list[np.
                                    argmax(xnl_list)])
```

Hence, the result is as follows.

```
Max Likelihood Estimate of P(act|x_l): 0.11900000000000001
Max Likelihood Estimate of P(act|x_nl): 0.085
```

Part C This part requires us to compute the posterior distributions of $P(X = x|Data)$ using the likelihood functions that we have computed in the previous parts. Here, the label *Data* is used for what is addressed as *act* in the previous parts. Furthermore, we are informed about using a prior probability distribution of $P(x) \propto 1$. Here, we use the Bayes rule to find these two distributions, which is as follows.

$$P(X = x|Data) = \frac{P(Data|X = x)P(X = x)}{P(Data)} \tag{5}$$

$$= \frac{P(Data|X = x)P(X = x)}{\sum_i P(Data|X = x_i)P(x = x_i)} \tag{6}$$

Since we know that $P(X = x) = 1$, we should calculate the separate normalizing constants, which for a discrete case corresponds to the sum of all the values in the given distribution. Furthermore, the question asks us to compute the discretized CDFs (Cumulative Distribution Functions) of the posterior distributions. For that, we simply add up the posterior probabilities in each calculation step since the definition of CDF is given as follows.

$$F_x(x|Data) = P(X \leq x|Data) = \sum_{x_i \leq x} P(X = x_i|Data) = \sum_{x_i \leq x} p(x_i|Data) \tag{7}$$

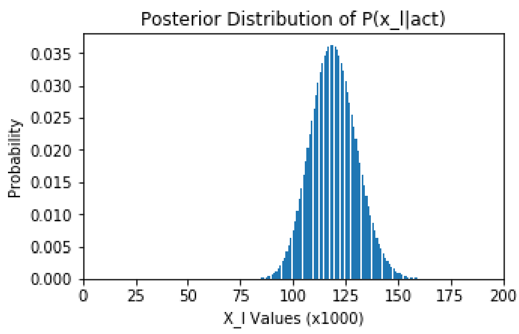
Hence, we have constructed a helper function for posterior probabilities which takes an additional input in the form of normalizing constant and then normalizes the prior.

```
def posterior_distribution(prob, normalization_constant, all_ins,
                                pos_ins):
    #assuming the prior distribution of a uniform distribution of
                                    1.
    return (spc.comb(all_ins, pos_ins) * prob**(pos_ins) * (1.0-
                                    prob)**(all_ins - pos_ins)
                                    ) / normalization_constant
```

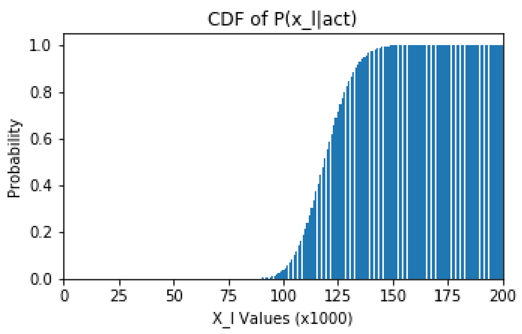
The code used during the procedure for language and not language case and the respective plots are provided below.

```
prob_list = np.arange(0.0,1.0,0.001)

sum_prob_xl = np.sum(xl_list)
sum_prob_xnl = np.sum(xnl_list)

sum = 0
cdf_xl = []
posterior_xl = []
for pr in prob_list:
    temp = posterior_distribution(pr,sum_prob_xl,869,103)
```

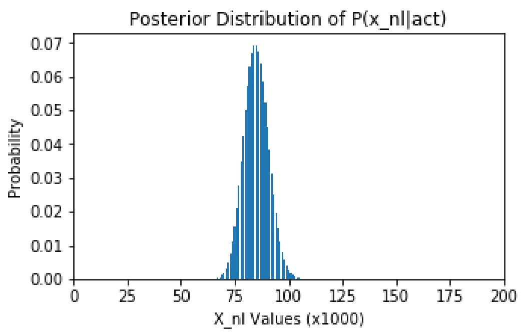
```
        sum += temp
        cdf_xl.append(sum)
        posterior_xl.append(posterior_distribution(pr,sum_prob_xl,869,
                                                    103))

index = np.arange(0,1000)
plt.xlim(0,200)
plt.bar(index, posterior_xl)
plt.xlabel('Probability of area activated values * 1000')
plt.ylabel('P(act|x_nl)')
plt.title('Posterior Distribution P(x_nl|act)')
plt.show()

index = np.arange(0,1000)
plt.xlim(0,200)
plt.bar(index, cdf_xl)
plt.xlabel('Probability of area activated values * 1000')
plt.ylabel('P(act|x_nl)')
plt.title('CDF of P(x_l|act)')
plt.show()
```



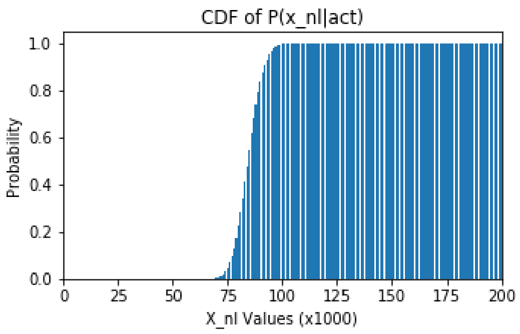
(a) Discretized Probability Distribution of $P(X_l|act)$



(b) CDF of $P(X_l|act)$



(c) Discretized Probability Distribution of $P(X_{nl}|act)$



(d) CDF of $P(X_{nl}|act)$

Figure 3: Activation Probabilities assuming Bernoulli Priors

After these plots, question requires us to find the appropriate upper and lower 95% confidence intervals using the cumulative distributions.Hence, we look for the argmin of the discretized CDF functions when 0.025 is subtracted which corresponds to the lower 5% and 0.975 which corresponds to the upper 5%. The code snippet with the results are provided.

```
index_lower = np.argmin(np.abs(np.asarray(cdf_xl)-0.025))
index_upper = np.argmin(np.abs(np.asarray(cdf_xl)-0.975))
print('Lower Bound for X_l:',prob_list[index_lower])
print('Upper Bound for X_l:',prob_list[index_upper])
```

```
Lower Bound for X_l: 0.098
Upper Bound for X_l: 0.14100000000000001
```

```
index_lower = np.argmin(np.abs(np.asarray(cdf_xnl)-0.025))
index_upper = np.argmin(np.abs(np.asarray(cdf_xnl)-0.975))
print('Lower Bound for X_nl:',prob_list[index_lower])
print('Upper Bound for X_nl:',prob_list[index_upper])
```

```
Lower Bound for X_nl: 0.073
Upper Bound for X_nl: 0.096
```

Part D Here, we are asked to plot the joint distribution of $P(X_l, X_{nl}|data)$, assuming the two variables are independent from each other. THis corresponds to the outer product of the two discretized distributions. The question requires us to specifically use the imagesc function of MATLAB, however, since we are using Python, we have used the contourf to plot the distribution. The code is provided below.

```
post_xl = np.asarray(posterior_xl).reshape((len(posterior_xl),1))
post_xnl = np.asarray(posterior_xnl).reshape((len(posterior_xnl),1
                                    )).T
outer_product = np.matmul(post_xl,post_xnl)

plt.figure(figsize=(7,5))
plt.contourf(prob_list, prob_list, outer_product)
plt.colorbar()
plt.title('Joint Probability Contour of P(X_l,X_nl|Data)')
plt.show()
```

The result is as follows. The lighter dot on the lower left corner represents the peak of the distribution with a probability value of approximately 0.0028 and nearly all other values are zero. This point also corresponds to the ML estimates of the posterior distributions when marginal probabilities are considered. The other part of the question requires us to calculate two probabilities given as $P(X_l > X_{nl}|Data)$ and $P(X_l X_{nl}|Data)$ from the discretized joint distribution. If we consider this distribution as a matrix, the first probability would correspond to the sum of elements in the lower-half triangle of the matrix and the second would correspond to the sum of upper-half triangle combined with the diagonal. Hence, we used the following code snippet to find and the result is as provided.

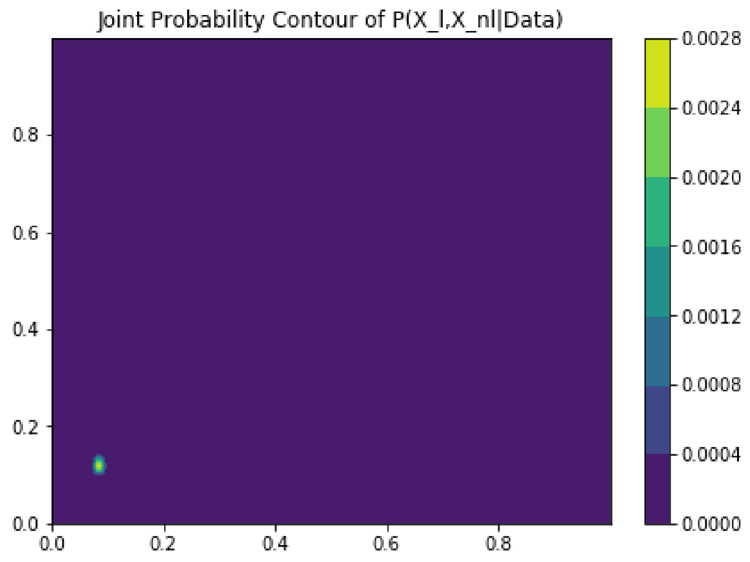


Figure 4: Joint Distribution Contour of $P(X_l, X_{nl}|Data)$

```
sumBigger=0
sumLower=0
numRows = outer_product.shape[0]
numCols = outer_product.shape[1]

for x in range(numRows):
    for y in range(numCols):
        if x>y:
            sumBigger += outer_product[x,y]
        else:
            sumLower += outer_product[x,y]
print('P(X_l>X_nl|Data) =',sumBigger)
print('P(X_l<=X_nl|Data) =',sumLower)
```

```
P(X_l>X_nl|Data) = 0.9978520275861242
P(X_l<=X_nl|Data) = 0.002147972413864103
```

We can comment that given data, the probability of that data being related with language is significantly higher than its inverse.

Part E Here, we are asked to calculate the probability $P(language|activation)$ with prior $P(activation) = 0.5$, and infer the activation with respect to the existence of a language event. In order to build a reverse inference type of relationship, we have used the Bayes rule given as follows with its extended form.

$$P(language|activation) = \frac{P(activation|language)P(language)}{P(activation)} \tag{8}$$

$$= \frac{P(activation|language)P(language)}{P(activation|language)P(language) + P(activation|no\,language)P(no\,language)} \tag{9}$$

Since $P(language) = 0.5$ is given, we can infer that $P(nolanguage) = 1-0.5 = 0.5$. Furthermore, $P(activation|language)$ and $P(activation|nolanguage)$ are the maximum likelihood estimates that we have found in Part B. Hence, we can calculate the results as,

$$P(language|activation) = \frac{0.119 * 0.5}{0.119 * 0.5 + 0.085 * 0.5} = 0.58\overline{3} \tag{10}$$

Here, we can say that reverse inference is a valid statement to make since statistically 58.3% of the stimulus that occurs through the Broca's Area is related with language, and we can conclude that it is more likely for the stimulus to be a language affiliated stimuli than a non-language one if Broca's Area is activated.

Python Code

```
import sys
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as spc


question = sys.argv[1]

def ayhan_okuyan_21601531_hw1(question):
    if question == '1' :
        print('Part A')
        A = np.matrix('[1, 0, -1, 2; 2, 1, -1, 5; 3, 3, 0, 9]')
        alpha = np.random.rand()
        beta = np.random.rand()
        x = np.asarray([alpha - 2 * beta, -alpha - beta, alpha, beta])
        result = np.matmul(A, x)
        print('Result of Ax', result)

        print('Part B')
        alpha = 5
        beta = 10
        x = np.asarray([1 + alpha - 2 * beta, 2 - alpha - beta, alpha,
                                    beta])
        result = np.matmul(A, x)
        print('Result of Ax', result)

        print('Part C')
        alpha = np.random.rand()
        beta = np.random.rand()
        x = np.asarray([1 + alpha - 2 * beta, 2 - alpha - beta, alpha,
                                    beta])
        result = np.matmul(A, x)
        print('Result of Ax', result)

        print('Part D')
        u, sigma, v_t = np.linalg.svd(A)

        sigma_ = np.zeros(A.shape)

        fill_sig = 1 / sigma
        # to remove the computational errors (100 is arbitrary)
        fill_sig[fill_sig >= 100] = 0
        np.fill_diagonal(sigma_, fill_sig)
        sigma_ = sigma_.T

        psdA = np.matmul(np.matmul(v_t.T, sigma_), u.T)
        print('Pseudo-inverse with SVD\n', psdA)

        psdA_ = np.linalg.pinv(A)
        print('Pseudo-inverse with pinv\n', psdA_)

        print('Part E')
        rng = np.arange(-5, 5, 0.01)
        alpha = np.zeros(rng.shape)

        plt.figure(figsize=(5, 3))
        plt.plot(rng, alpha)
        plt.plot(rng, 2 - rng)
        plt.plot(rng, 2 * rng - 1)
        plt.axvline(x=0)
        plt.xlabel('alpha')
        plt.ylabel('beta')
        plt.show()

        sparse_soln_set = [(1, 1), (0, 0), (0, 0.5), (0, 2), (-1, 0), (2
                                                   , 0)]
        for alp, bet in sparse_soln_set:
            print('Alpha:', alp, 'Beta:', bet)
            x = np.asarray([1 + alp - 2 * bet, 2 - alp - bet, alp, bet])
            print('x:', x)
            result = np.matmul(A, x)
            print('b:', result[0])

        print('Part F')
        least_norm_calc = [1 / 17, 11 / 17, 10 / 17, 13 / 17]
        least_norm_psdinv = np.matmul(psdA, [1, 4, 9])
        print('Least Squares Solution,\n', np.asarray(least_norm_calc))
        print('Least norm by pseudo-inverse,\n', least_norm_psdinv)

    elif question == '2' :
        print('Part A')
        prob_list = np.arange(0.0, 1.0, 0.001)
        xl_list = []
        for pr in prob_list:
            xl_list.append(binomialLikelihood(pr, 869, 103))

        index = np.arange(0, 1000)
        plt.xlim(0, 200)
        plt.bar(index, xl_list)
        plt.xlabel('X_l Values (x1000)')
        plt.ylabel('Likelihood')
        plt.title('Likelihood of activation w/Language')
        plt.show()

        prob_list = np.arange(0.0, 1.0, 0.001)
        xnl_list = []
        for pr in prob_list:
            xnl_list.append(binomialLikelihood(pr, 2353, 199))

        index = np.arange(0, 1000)
        plt.xlim(0, 200)
        plt.bar(index, xnl_list)
        plt.xlabel('X_nl Values (x1000)')
        plt.ylabel('Likelihood')
        plt.title('Likelihood of activation w/o Language')
        plt.show()
```

```
print('Part B')
print('Max Likelihood Estimate of P(act|X_l):', prob_list[np.
                                            argmax(xl_list)])
print('Max Likelihood Estimate of P(act|X_nl):', prob_list[np.
                                            argmax(xnl_list)])

print('Part C')
prob_list = np.arange(0.0, 1.0, 0.001)

sum_prob_xl  = np.sum(xl_list)
sum_prob_xnl  = np.sum(xnl_list)

sum = 0
cdf_xl = []
posterior_xl = []
for pr in prob_list:
    temp = posterior_distribution(pr, sum_prob_xl, 869, 103)
    sum += temp
    cdf_xl.append(sum)
    posterior_xl.append(posterior_distribution(pr, sum_prob_xl,
                                            869, 103))

index = np.arange(0, 1000)
plt.figure(figsize=(5, 3))
plt.xlim(0, 200)
plt.bar(index, posterior_xl)
plt.xlabel('X_l Values (x1000)')
plt.ylabel('Probability')
plt.title('Posterior Distribution of P(x_l|act)')
plt.show()

index = np.arange(0, 1000)
plt.figure(figsize=(5.23, 3))
plt.xlim(0, 200)
plt.bar(index, cdf_xl)
plt.xlabel('X_l Values (x1000)')
plt.ylabel('Probability')
plt.title('CDF of P(x_l|act)')
plt.show()

prob_list = np.arange(0.0, 1.0, 0.001)

sum_prob_xl  = np.sum(xl_list)
sum_prob_xnl  = np.sum(xnl_list)

sum = 0
cdf_xnl = []
posterior_xnl = []
for pr in prob_list:
    temp = posterior_distribution(pr, sum_prob_xnl, 2353, 199)
    sum += temp
    cdf_xnl.append(sum)
    posterior_xnl.append(temp)

index = np.arange(0, 1000)
```

```
plt.figure(figsize=(5.1, 3))
plt.xlim(0, 200)
plt.bar(index, posterior_xnl)
plt.xlabel('X_nl Values (x1000)')
plt.ylabel('Probability')
plt.title('Posterior Distribution of P(x_nl|act)')
plt.show()

index = np.arange(0, 1000)
plt.figure(figsize=(5.17, 3))
plt.xlim(0, 200)
plt.bar(index, cdf_xnl)
plt.xlabel('X_nl Values (x1000)')
plt.ylabel('Probability')
plt.title('CDF of P(x_nl|act)')
plt.show()

index_lower = np.argmin(np.abs(np.asarray(cdf_xl) - 0.025))
index_upper = np.argmin(np.abs(np.asarray(cdf_xl) - 0.975))
print('Lower Bound for X_l:', prob_list[index_lower])
print('Upper Bound for X_l:', prob_list[index_upper])

index_lower = np.argmin(np.abs(np.asarray(cdf_xnl) - 0.025))
index_upper = np.argmin(np.abs(np.asarray(cdf_xnl) - 0.975))
print('Lower Bound for X_nl:', prob_list[index_lower])
print('Upper Bound for X_nl:', prob_list[index_upper])

print('Part D')
post_xl = np.asarray(posterior_xl).reshape((len(posterior_xl), 1
                                           ))
post_xnl = np.asarray(posterior_xnl).reshape((len(posterior_xnl)
                                           , 1)).T
outer_product = np.matmul(post_xl, post_xnl)

plt.figure(figsize=(7, 5))
plt.contourf(prob_list, prob_list, outer_product)
plt.colorbar()
plt.title('Joint Probability Contour of P(X_l,X_nl|Data)')
plt.show()

sumBigger = 0
sumLower = 0
numRows = outer_product.shape[0]
numCols = outer_product.shape[1]

for x in range(numRows):
    for y in range(numCols):
        if x > y:
            sumBigger += outer_product[x, y]
        else:
            sumLower += outer_product[x, y]
print('P(X_l>X_nl|Data) =', sumBigger)
print('P(X_l<=X_nl|Data) =', sumLower)
```

```
        print('Maximum Value of Marginal Distribution:', np.max(post_xl)
                                                   * np.max(post_xnl))

        print('Part E')
        result = (0.119 * 0.5) / (0.119 * 0.5 + 0.085 * 0.5)
        print(result)


def binomialLikelihood(prob, all_ins, pos_ins):
    return spc.comb(all_ins, pos_ins) * prob ** (pos_ins) * (1.0 - prob)
                                                 ** (all_ins - pos_ins)

def posterior_distribution(prob, normalization_constant, all_ins,
                                            pos_ins):
    #assuming the prior distribution of a uniform distribution of 1.
    return (spc.comb(all_ins, pos_ins) * prob**(pos_ins) * (1.0-prob)**(
                                                all_ins - pos_ins)) /
                                                normalization_constant


ayhan_okuyan_21601531_hw1(question)
```