EEE482 - Computational Neuroscience

Homework Assignment - 1

Ahmet Ali Nuhoğlu
21602149

Ahmet Ali NUHOĞLU
21602149

Homework 1

EEE482

Question 1

Given system of equations

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

a) In this part we are expected to find a general solution x_n to the system $Ax = 0$. To do that my initial step was to find the reduced row echelon form of the given matrix A. Corresponding row operations are shown below.

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

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & 1 & -1 & 5 \\ 3 & 3 & 0 & 9 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 - 3R_1]{R_2 \leftarrow R_2 - 2R_1} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 3 & 3 \end{bmatrix} \xrightarrow{R_3 \leftarrow R_3 - 3R_2} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

Therefore our equation (1) becomes

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = 0 \tag{2}$$

From the system of equations obtained above we can see that x_1 and x_2 are our pivot variables. x_3 and x_4 are free variables. We can write our pivot variables in terms of free variables, using the equations obtained from the equation (4) as follows:

$$\begin{matrix} x_1 - x_3 + 2x_4 = 0 \\ x_2 + x_3 + x_4 = 0 \end{matrix} \iff \begin{matrix} x_1 = x_3 - 2x_4 \\ x_2 = -x_3 - x_4 \end{matrix} \tag{3}$$

Finally we can interpret the general solution as follows:

$$x_n = \begin{bmatrix} x_3 - 2x_4 \\ -x_3 - x_4 \\ x_3 \\ x_4 \end{bmatrix} = x_3 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} + x_4 \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix} \qquad x_3,\ x_4 \in \mathbb{R} \tag{4}$$

Code given below shows the imports of the used libraries. Numpy library is used for easing the computations. I have used the sys library to show all the outputs of the matrices for better debugging.

```
import numpy as np # for easier computations
import matplotlib.pyplot as plt # for plotting
import math # for easier computations
import sys

# for displaying all of the outputs in case of a need for debugging
np.set_printoptions(threshold=sys.maxsize)
```

In the following code, I have validated what I have found by hand for the part b.

```
# defining the array A provided in the homework assignment
A = np.array ([ [1, 0, -1, 2], [2, 1, -1, 5], [3, 3, 0, 9] ])

# generating two random variables for the free variable values
x3 = np.random.rand()
x4 = np.random.rand()

# interpretation of the solution derived by hand
x_general = np.array([[x3-2*x4], [-x3-x4], [x3], [x4]])

print(np.around(A.dot(x_general).T, 4))
```

Output of the given code is shown below.

```
[[0. 0. 0.]]
```

b) In this part we are expected to find a particular solution to system of equations given in (1). To achieve a particular solution to the given system, initially I have reduced the matrix to its reduced row echelon form.

$$\left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 2 & 1 & -1 & 5 & 4 \\ 3 & 3 & 0 & 9 & 9 \end{array}\right] \xrightarrow[R_3 \leftarrow R_3 - 3R_1]{R_2 \leftarrow R_2 - 2R_1} \left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 3 & 3 & 3 & 6 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 3 & 3 & 3 & 6 \end{array}\right] \xrightarrow{R_3 \leftarrow R_3 - 3R_2} \left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{5}$$

therefore, from the reduced row echelon form of the matrix, we can write our free variables in terms of pivot variables as follows:

$$\begin{aligned} x_1 - x_3 + 2x_4 &= 1 \\ x_2 + x_3 + x_4 &= 2 \end{aligned} \iff \begin{aligned} x_1 &= x_3 - 2x_4 + 1 \\ x_2 &= -x_3 - x_4 + 2 \end{aligned} \tag{6}$$

from the derived equations (8), we can assume $x_3 = 0$ and $x_4 = 0$. Thus,

$$x_p = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

is a particular solution to the given system of equations. Validation of the result by code is as follows.

```
    # defining the given matrix A
    A = np.array ([ [1, 0, -1, 2], [2, 1, -1, 5], [3, 3, 0, 9] ])

    # defining a matrix for a particular solution which is derived␣
↪by hand
    x_particular = np.array([[1], [2], [0], [0]])

    # printing the output on the console. The output has to be␣
↪equal to the given b matrix
    print(A.dot(x_particular).T)
```

Output of the given code is:

```
[[1 4 9]]
```

c) From the system of equations (8),which are derived in the previous part, we can form a general solution to the system $Ax = b$.

$$x_n = \begin{bmatrix} x_3 - 2x_4 + 1 \\ -x_3 - x_4 + 2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} + x_3 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} + x_4 \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix} \qquad x_3,\ x_4 \in \mathbb{R} \tag{8}$$

General solution that is found is validated by the code below.

```
# defining the given matrix A
A = np.array ([ [1, 0, -1, 2], [2, 1, -1, 5], [3, 3, 0, 9] ])

# assigning random values to free variables
x3 = np.random.rand()
x4 = np.random.rand()

# assigning numerical values to the general solution, using the␣
 ↪random numbers generated above
x_general = np.array([[1+x3-2*x4], [2-x3-x4], [x3], [x4]])
```

```
# printing the output on the console. The output has to be equal␣
 ↪to the given b matrix
print(A.dot(x_general).T)
```

Output of the code is:

```
[[1. 4. 9.]]
```

d) Another way to solve this system of equations is to find the inverse of matrix A. Therefore, by multiplying both sides of the equation $Ax = B$ by A inverse, will give us the least norm solution to the system. But, it is not possible to find the exact inverse of matrix A, since it is not a square matrix. We will try to find a pseudo inverse for that matrix.

Pseudo inverse of matrix is denoted by A^+ and to calculate pseudo inverse we will get help from Singular Value Decomposition (SVD). Singular value decomposition of A can be shown as follows:

$$A_{mxn} = U_{mxm}\Sigma_{mxn}V^T_{nxn}$$
$$A^+ = V\Sigma^+U^T$$

In these representations of singular value decomposition, U and V denote the left-singular and right-singular vectors respectively. Σ is the matrix which consists of singular values on its diagonal.

We can find $v's$ for the right-singular vectors as follows:

$$A^TAv = \sigma^2 v, \text{ for } v \neq 0 \tag{9}$$

We can find $u's$ for the left-singular vectors as follows:

$$AA^Tu = \sigma^2 u, \text{ for } u \neq 0 \tag{10}$$

Dimensions of AA^T and A^TA will be $3x3$ and $4x4$ respectively. Since they will have the same eigenvalues and $3x3$ will be easier to compute, I tried to compute left-singular vectors first.

$$AA^T = \begin{bmatrix} 6 & 13 & 21 \\ 13 & 31 & 54 \\ 21 & 54 & 99 \end{bmatrix} \tag{11}$$

we can rewrite equation (11) as follows:

$$(AA^T - I\sigma^2)u = 0 \qquad \Longrightarrow \qquad det(AA^T - I\sigma^2) = 0 \tag{12}$$

Afterwards we can assume $\lambda = \sigma^2$ and we can find the singular values.

$$det(AA^T - I\lambda) = \begin{vmatrix} 6-\lambda & 13 & 21 \\ 13 & 31-\lambda & 54 \\ 21 & 54 & 99-\lambda \end{vmatrix} \tag{13}$$

After some basic algebraic computations, we can find its characteristic polynomial as

$$-\lambda^3 + 136\lambda^2 - 323\lambda = 0 \tag{14}$$

Eventually, we can find the eigenvalues as:

$$\lambda_1 = 68 + \sqrt{4301} \qquad \lambda_2 = 68 - \sqrt{4301} \qquad \lambda_3 = 0$$

After finding those eigenvalues, since it was computationally hard to keep going by hand, I tried to calculate the rest in python but without using the built in pseudo inverse function directly. Corresponding code is given below.

```
# using the svd function defined in the numpy library to find
# the elements of the singular value decomposition
u, s, vt = np.linalg.svd(A)
print("Left Singular Vector matrix U:\n", u, "\n")
print("Right Singular Vector matrix V:\n", vt)
print('\nSigma:')

# for diagonalizing the sigma values we create a matrix with the
# same shape with the provided matrix A. Therefore, using a for
# loop we diagonalize the singular values.
sigma = np.zeros(A.shape)
rows, cols = sigma.shape
for i in range(rows):
    sigma[i][i] = np.around(s[i], 7)
print(sigma, '\n')


# finding pseudo inverse of the sigma from the sigma
sigma_pseudo = np.zeros(sigma.shape)
for i in range(rows):
    if sigma[i][i] != 0:
        sigma_pseudo[i][i] = 1/sigma[i][i]
    else:
        sigma_pseudo[i][i] = 0
```

```
sigma_pseudo = sigma_pseudo.T

A_pseudo = vt.T.dot(sigma_pseudo).dot(u.T)
print("A = U.Sigma.V_t\n", np.around(u.dot(sigma).dot(vt)))
print('\n A pseudo inverse:\n', A_pseudo)
print('\n A(A^T)A: \n', np.around(A.dot(A_pseudo).dot(A)))
```

Output of the given code is:

```
Left Singular Vector matrix U:
 [[-0.1898465  -0.70019575 -0.6882472 ]
 [-0.47607011 -0.54742401  0.6882472 ]
 [-0.85867081  0.45831524 -0.22941573]]

Right Singular Vector matrix V:
 [[-0.32168832 -0.26407196  0.05761637 -0.90744861]
 [-0.27016145  0.53217213  0.80233358 -0.00815077]
 [ 0.87970359 -0.13403501  0.3825912  -0.2485562 ]
 [-0.22282505 -0.79315411  0.45449439  0.33865972]]

Sigma:
[[11.5577684  0.         0.         0.        ]
 [ 0.         1.5549888  0.         0.        ]
 [ 0.         0.         0.         0.        ]]

A = U.Sigma.V_t
 [[ 1.  0. -1.  2.]
 [ 2.  1. -1.  5.]
 [ 3.  3. -0.  9.]]

 A pseudo inverse:
 [[ 0.12693499  0.10835914 -0.05572756]
 [-0.23529412 -0.17647059  0.17647059]
 [-0.36222911 -0.28482973  0.23219815]
 [ 0.01857585  0.04024768  0.06501548]]

 A(A^T)A:
 [[ 1. -0. -1.  2.]
 [ 2.  1. -1.  5.]
 [ 3.  3.  0.  9.]]
```

Also, I have used the pinv function of the numpy library, to validate the pseudo inverse of A that I have found using SVD. Code and its output to validate my results:

```python
# validating the previous result from the built in
# function to find pseudo inverse of A
A_pinv = np.linalg.pinv(A)
print("A pseudo inverse: \n", A_pinv, "\n")
print("A*A_pinv*A: \n", np.around(A.dot(A_pinv).dot(A)))
```

```
A pseudo inverse:
 [[ 0.12693498  0.10835913 -0.05572755]
 [-0.23529412 -0.17647059  0.17647059]
 [-0.3622291  -0.28482972  0.23219814]
 [ 0.01857585  0.04024768  0.06501548]]

A*A_pinv*A:
 [[ 1. -0. -1.  2.]
 [ 2.  1. -1.  5.]
 [ 3.  3.  0.  9.]]
```

e) To find the sparsest solution we have to find the vectors in the solution set which has the most number of zeros in it. To do that, I have tried several ways, one of them was to assign zeros to the free variables x_3 and x_4 in the general solution, which is given in the equation (4) at **part a**. Therefore, I obtained the following result.

$$x_3 = 0 \text{ and } x_4 = 0 \qquad \Longrightarrow \qquad \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} \tag{15}$$

Secondly, I have tried to solve $x_1 = 0$ and $x_2 = 0$ with respect to free variables.

$$\begin{aligned} x_3 - 2x_4 + 1 &= 0 \\ -x_3 - x_4 + 2 &= 0 \end{aligned} \qquad x_3, x_4 \neq 0 \tag{16}$$

This resulted in the following vector.

$$x_3 = 1 \text{ and } x_4 = 1 \qquad \Longrightarrow \qquad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{17}$$

From the equations given in (16), I thought that we can also obtain vectors include 2 zeros in it, by assigning 0 to one of the free variables an then solving one of the equations of pivot variables with respect to it. resulting vectors are given below.

$$x_3 = 0 \text{ and } x_4 = 0.5 \qquad \Longrightarrow \qquad \begin{bmatrix} 0 \\ 1.5 \\ 0 \\ 0.5 \end{bmatrix} \tag{18}$$

$$x_3 = 0 \text{ and } x_4 = 2 \quad \Longrightarrow \quad \begin{bmatrix} -3 \\ 0 \\ 0 \\ 2 \end{bmatrix} \tag{19}$$

$$x_3 = -1 \text{ and } x_4 = 0 \quad \Longrightarrow \quad \begin{bmatrix} 0 \\ 3 \\ -1 \\ 0 \end{bmatrix} \tag{20}$$

$$x_3 = 2 \text{ and } x_4 = 0 \quad \Longrightarrow \quad \begin{bmatrix} 3 \\ 0 \\ 2 \\ 0 \end{bmatrix} \tag{21}$$

There might be other sparsest solutions to the system, those are the ones that I could find. Validation of the solutions are given below.

[15]:
```
# found sparsest solutions
xs_1 = [[1],[2],[0],[0]]
xs_2 = [[0],[0],[1],[1]]
xs_3 = [[0],[1.5],[0],[0.5]]
xs_4 = [[-3],[0],[0],[2]]
xs_5 = [[0],[3],[-1],[0]]
xs_6 = [[3],[0],[2],[0]]

print("A.xs_1 = ", A.dot(xs_1).T)
print("A.xs_2 = ", A.dot(xs_2).T)
print("A.xs_3 = ", A.dot(xs_3).T)
print("A.xs_4 = ", A.dot(xs_4).T)
print("A.xs_5 = ", A.dot(xs_5).T)
print("A.xs_6 = ", A.dot(xs_6).T)
```

```
A.xs_1 =  [[1 4 9]]
A.xs_2 =  [[1 4 9]]
A.xs_3 =  [[1. 4. 9.]]
A.xs_4 =  [[1 4 9]]
A.xs_5 =  [[1 4 9]]
A.xs_6 =  [[1 4 9]]
```

f) Definition of a least norm solution is

$$\|x\| = \sqrt{\sum_i x_i^2} \tag{22}$$

Since we have found the pseudo inverse A^+ in the **part d**, we can multiply both sides of the equation $Ax = b$ with A^+ from the left. Therefore, we will end up with the least norm solution.

$$x^* = A^+ b \qquad \Longrightarrow \qquad x^* = \begin{bmatrix} 0.127 & 0.108 & -0.056 \\ -0.235 & -0.176 & 0.176 \\ -0.362 & -0.284 & 0.232 \\ 0.0186 & 0.040 & 0.065 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix} \tag{23}$$

Thus, we find x^* as

$$x^* = \begin{bmatrix} 0.059 \\ 0.647 \\ 0.588 \\ 0.764 \end{bmatrix} \tag{24}$$

Code for the validation of the found least norm solution is given below.

```
# Here we find the least norm solution using
# pseudo inverse of matrix A
B = np.array ([[1], [4], [9]])
x_least_norm = A_pinv.dot(B)
print("Least norm solution:", x_least_norm.T)
```

```
Least norm solution: [[0.05882353 0.64705882 0.58823529 0.76470588]]
```

Question 2

In this question we are told that, after a wide search, it has been found that Broca's area was active in 103 out of 869 tasks that involve language, but it is also told that in 199 out of 2353 tasks that do not involve language, Broca's area was active.

a) In this part we are told that both activation given language or activation not given no language are distributed with Bernoulli distribution.

$$P(data|x) = \begin{cases} x & \text{if } data = 1 \\ 1-x & \text{if } data = 0 \end{cases} \tag{1}$$

We are expected to compute these values for data points from 0 to 1 with step size 0.001. Eventually, we are expected to plot them as bar charts. We can find their likelihood functions by doing the following computations for all data points as probabilities x_l and x_{nl}.

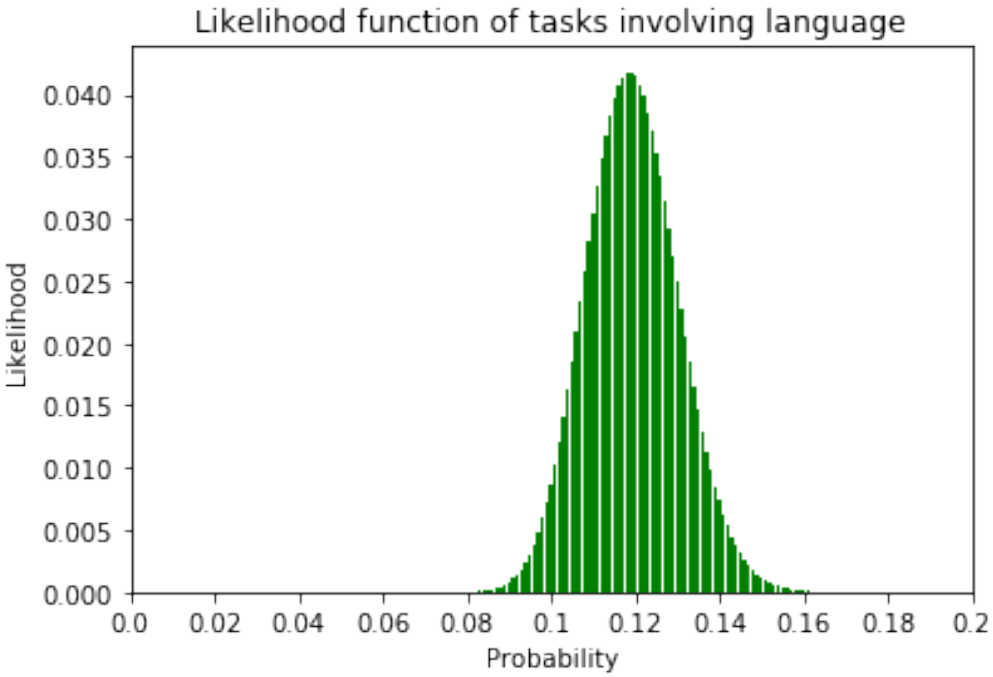
$$\begin{aligned} P(data|X_l = x_l) &= \binom{869}{103} x_l^{103}(1-x_l)^{766} \\ P(data|X_{nl} = x_{nl}) &= \binom{2353}{199} x_l^{199}(1-x_{nl})^{2154} \end{aligned} \tag{2}$$

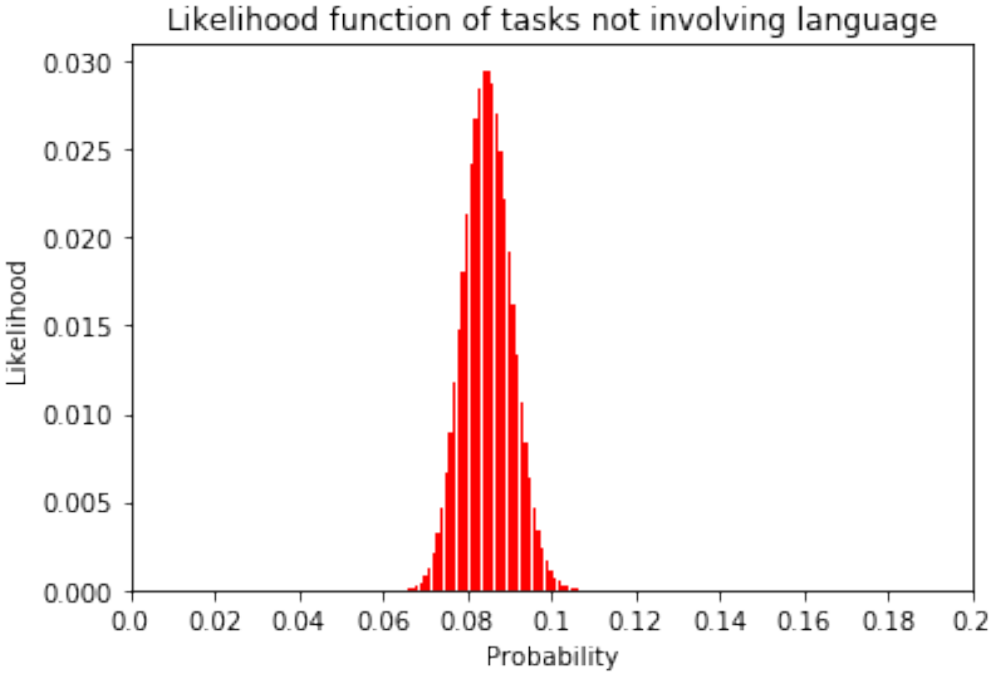
Bar charts of these likelihood functions and code to generate those charts is given below.

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

# this function is used to generate a dataset with bernuolli␣
 ↪distribution, with given parameters
def bernuolli(x, total_trials, positive_results):
    comb = (math.factorial(total_trials)/(math.
 ↪factorial(positive_results)*math.
 ↪factorial(total_trials-positive_results)))
    return␣
 ↪comb*(x**positive_results)*((1-x)**(total_trials-positive_results))

data_points = np.arange(0, 1.001, 0.001)
x_l  = bernuolli(data_points, 869, 103)
x_nl = bernuolli(data_points, 2353, 199)

plt.figure()
plt.bar(np.arange(len(data_points)), x_l, color='g')
plt.xlim(0, 200)
```

```
plt.xticks(np.arange(0, 201, 20), np.around(np.arange(0, 0.201, 0.
↪02),2))
plt.title("Likelihood function of tasks involving language")
plt.xlabel("Probability")
plt.ylabel("Likelihood")

plt.figure()
plt.xlim(0, 200)
plt.xticks(np.arange(0, 201, 20), np.around(np.arange(0, 0.201, 0.
↪02),2))
plt.title("Likelihood function of tasks not involving language")
plt.xlabel("Probability")
plt.ylabel("Likelihood")
plt.bar(np.arange(len(data_points)), x_nl, color='r')
plt.show(block=False)
```

In these bar charts, since for probabilities higher than (0.2) likelihood is approximately zero, I have limited the x-axis only to show the probabilities in between 0 and (0.2). Also, we do not lose any information limiting the x-axis between 0 and (0.2) and gives us a more clear view of the likelihood function.





b) We are expected to find the values of x_l and x_{nl} which maximizes the likelihood of their functions. To obtain those data points my approach was to find the peak of the likelihood function. Then, finding its index where indexes denote the values of x_l and x_{nl}. While doing that, I have planned to use the **numpy.amax()** function but therefore, I found that **numpy.argmax()** function was actually what I need. That function directly returns the index of the peak value. By using those functions I have found and printed the values as follows:

```
print("Maximum likelihood of x_l and its probability respectively:␣
 ↪(", np.amax(x_l), ", ", np.argmax(x_l)/1000, ")")
print("Maximum likelihood of x_nl and its probability respectively:
 ↪ (", np.amax(x_nl), ", ", np.argmax(x_nl)/1000, ")")
```

```
Maximum likelihood of x_l and its probability respectively:
( 0.0417952478261316,  0.119 )
Maximum likelihood of x_nl and its probability respectively:
( 0.02946375315559796,  0.085 )
```

c) To obtain $P(X|data)$ from what we already have, we will use Bayes' Rule. **Bayes' Rule :**

$$P(X|data)P(data) = P(data|X)P(X)$$

$$P(X|data) = \frac{P(data|X)P(X)}{P(data)} \tag{3}$$

For using Bayes' Rule to obtain $P(X|data)$, $P(X)$ is given to us as a uniform distribution as follows:

$$P(X) = \frac{1}{\#\ of\ all\ data\ points} = \frac{1}{1001}, \quad \forall x \in [0, 1000] \tag{4}$$

There is only $P(data)$ left unknown for us to compute the $P(X|data)$. Even though we do not know $P(data)$ directly, we can compute it from,

$$P(data) = \sum_x P(data|x)P(x) \tag{5}$$

If we put our findings in the Bayes' Rule and as $P(X)$ is a constant, $P(X)$'s in the numerator and denominator will cancel out each other.Thus, we end up with the following equation to find the posterior distribution $P(X|data)$.

$$P(X|data) = \frac{P(data|X)}{\sum_x P(data|x)} \tag{6}$$

Also, it has been expected from us to calculate the bounds of the 95% confidence intervals using the cumulative distribution functions of the posterior distribution functions that we have computed. to find the bounds the 95% confidence intervals, we need to find the values where $x = 0.025$ and $x = 0.975$. Since we have the step size 0.001, $x = 0.025$ and $x = 0.975$ respectively corresponds to 2.5% and 97.5%.

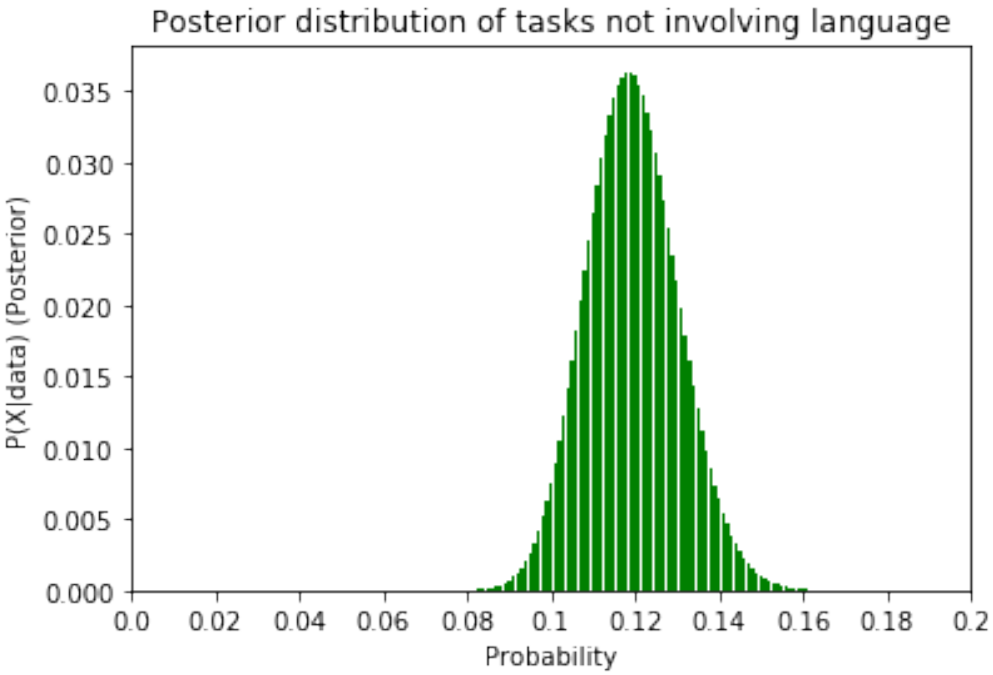
Part of the code which I made the corresponding calculations and obtained the bounds of the 95% confidence intervals and the expected plots of the posterior distributions are given below.

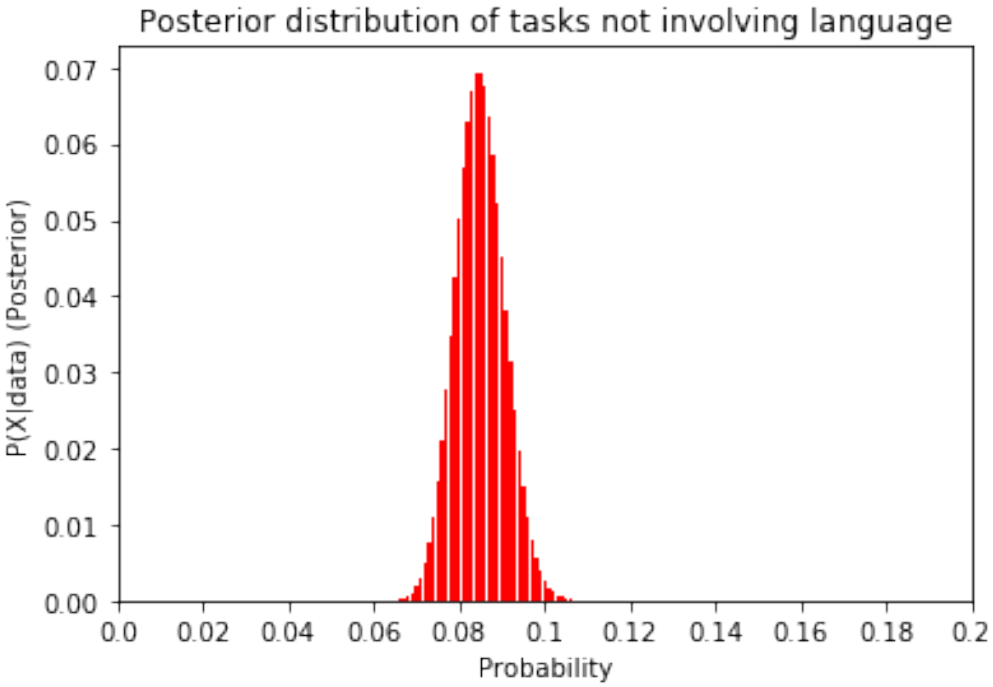
```
# (given in the assignment) uniform prior
uniform_pri = 1 / len(data_points)

# computing the normalizer with the previors values found
xl_normalizer = np.sum(x_l * uniform_pri)
xnl_normalizer = np.sum(x_nl * uniform_pri)

# computing the posterior distributions with the obtained
# values
xl_posterior = x_l * uniform_pri / xl_normalizer
xnl_posterior = x_nl * uniform_pri / xnl_normalizer

# creating numpy arrays to store the cdf's of the posteriors
xl_posterior_cdf = np.zeros(len(data_points))
xnl_posterior_cdf = np.zeros(len(data_points))
```

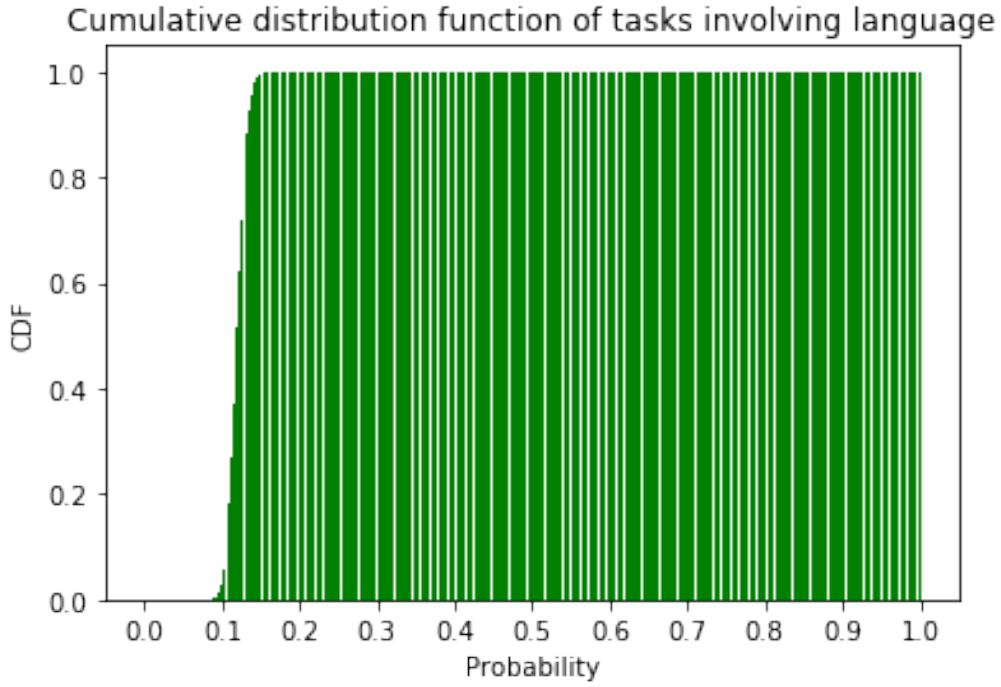
```
# booleans to find the bounds of the 95% CI of tasks
# involving language, using CDF
xl_lower = -1
found_xl_lower = False
xl_upper = -1
found_xl_upper = False

# computing the CDF of the posterior function of tasks
# involving language and holding the CI bounds
for i in range (0, len(data_points)):
    if i == 0:
        xl_posterior_cdf[i] = xl_posterior[i]
    else:
        xl_posterior_cdf[i] = xl_posterior_cdf[i-1] +␣
 ↪xl_posterior[i]

    if xl_posterior_cdf[i] > 0.025 and not found_xl_lower:
        xl_lower = np.round(data_points[i], 3)
        found_xl_lower = True
    elif xl_posterior_cdf[i] > 0.975 and not found_xl_upper:
        xl_upper = np.round(data_points[i], 3)
        found_xl_upper = True

# booleans to find the bounds of the 95% CI of tasks
# involving language, using CDF
xnl_lower = -1
found_xnl_lower = False
xnl_upper = -1
found_xnl_upper = False

# computing the CDF of the posterior function of tasks
# not involving language and holding the CI bounds
for i in range (0, len(data_points)):
    if i == 0:
        xnl_posterior_cdf[i] = xnl_posterior[i]
    else:
        xnl_posterior_cdf[i] = xnl_posterior_cdf[i-1] +␣
 ↪xnl_posterior[i]

    if xnl_posterior_cdf[i] > 0.025 and not found_xnl_lower:
        xnl_lower = np.round(data_points[i], 3)
        found_xnl_lower = True
    elif xnl_posterior_cdf[i] > 0.975 and not found_xnl_upper:
        xnl_upper = np.round(data_points[i], 3)
```

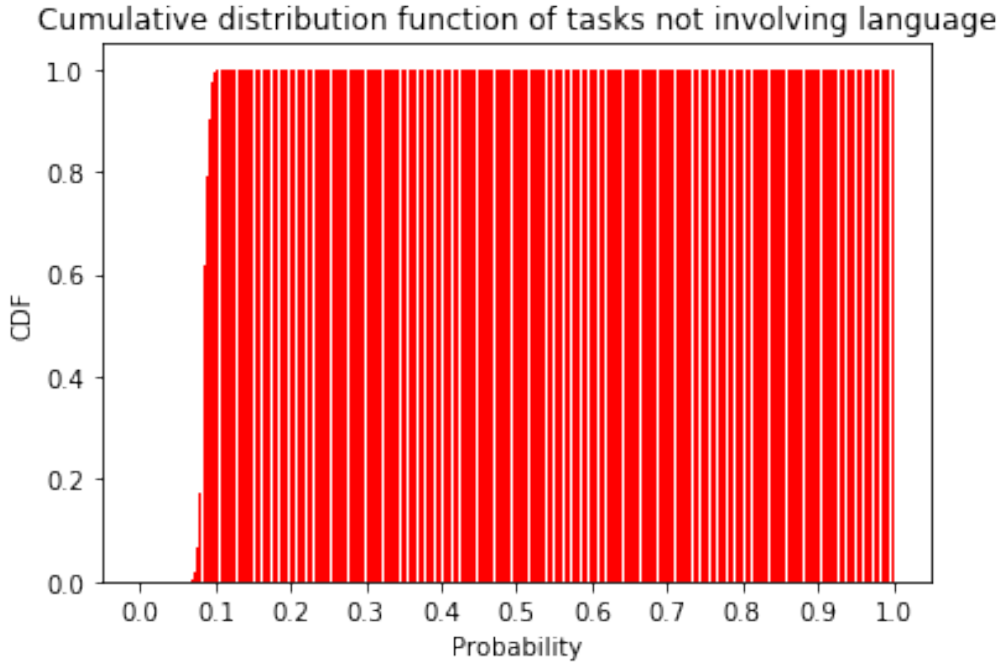
```
        found_xnl_upper = True


# Plotting the figures below
plt.figure()
plt.xlim(0, 200)
plt.xticks(np.arange(0, 201, 20), np.around(np.arange(0, 0.201, 0.
 ↪02),2))
plt.title("Posterior distribution of tasks not involving language")
plt.xlabel("Probability")
plt.ylabel("P(X|data) (Posterior)")
plt.bar(np.arange(len(data_points)), xl_posterior, color='g')

plt.figure()
plt.xlim(0, 200)
plt.xticks(np.arange(0, 201, 20), np.around(np.arange(0, 0.201, 0.
 ↪02),2))
plt.title("Posterior distribution of tasks not involving language")
plt.xlabel("Probability")
plt.ylabel("P(X|data) (Posterior)")
plt.bar(np.arange(len(data_points)), xnl_posterior, color='r')

plt.figure()
plt.xticks(np.arange(0, 1001, 100), np.around(np.arange(0, 1.001,␣
 ↪0.1),2))
plt.title("Cumulative distribution function of tasks involving␣
 ↪language")
plt.xlabel("Probability")
plt.ylabel("CDF")
plt.bar(np.arange(len(data_points)), xl_posterior_cdf, color='g')

plt.figure()
plt.xticks(np.arange(0, 1001, 100), np.around(np.arange(0, 1.001,␣
 ↪0.1),2))
plt.title("Cumulative distribution function of tasks not involving␣
 ↪language")
plt.xlabel("Probability")
plt.ylabel("CDF")
plt.bar(np.arange(len(data_points)), xnl_posterior_cdf, color='r')
plt.show(block=False)

print("%95 Confidence Interval Bounds for tasks involving␣
 ↪language")
print("\tLower: ", xl_lower, "\tHigher: ", xl_upper, "\n")
```

```
print("%95 Confidence Interval Bounds for tasks not involving␣
 ↪language")
print("\tLower: ", xnl_lower, "\tHigher: ", xnl_upper)
```







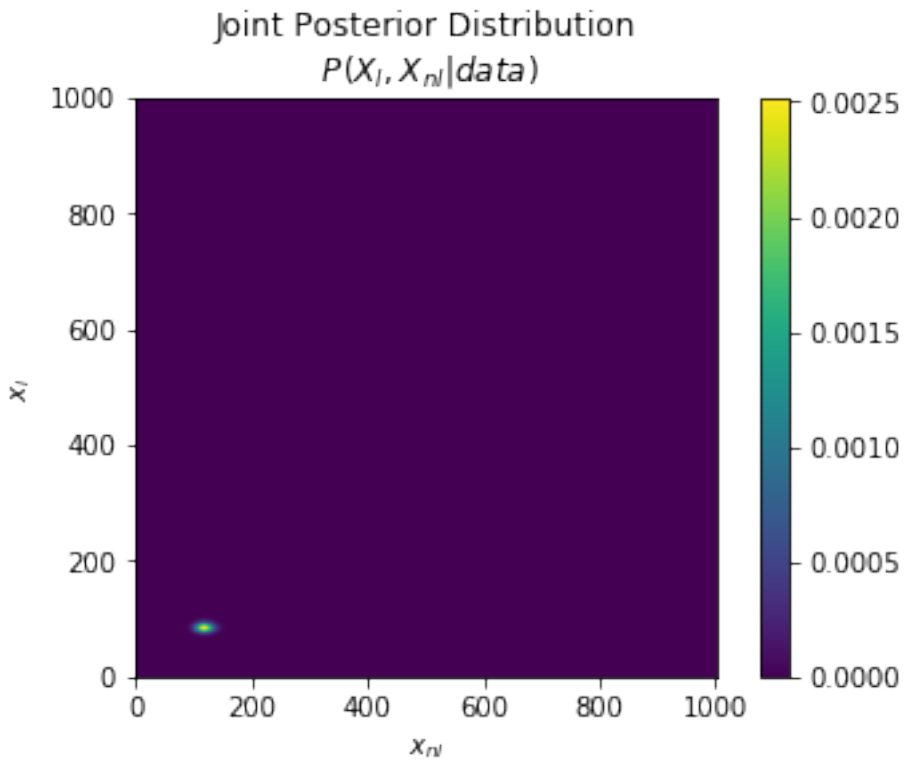



```
%95 Confidence Interval Bounds for tasks involving language
        Lower:  0.099   Higher:  0.142

%95 Confidence Interval Bounds for tasks not involving language
        Lower:  0.074   Higher:  0.096
```

d) It has been expected from us to find the joint posterior distribution $P(X_l, X_{nl}|data)$ over the probabilities x_l and x_{nl}. To compute the joint posterior distribution, I had to multiply **xl_posterior** and **xnl_posterior** transpose as matrices, to obtain a joint distribution. As we multiply transpose of **xnl_posterior**, vertical values will denote **xnl_posterior** and horizontal values will denote **xl_posterior**.

After we computed the joint distribution, we are expected to compute $P(X_l > X_{nl}|data)$ and $P(X_{nl} \geq X_l|data)$. Code for computing the probabilities and displaying the image of the joint distribution is as follows

```
# we put our posterior distributions in matrix form for matrix
# multiplication in order to find the joint distributions
xl_matrix = np.matrix(xl_posterior)
xnl_matrix = np.matrix(xnl_posterior).transpose()

joint_matrix = np.matrix(xnl_matrix * xl_matrix)

# plotting the joint posterior distribution
plt.figure()
plt.imshow(joint_matrix, origin='lower')
plt.colorbar()
plt.xlabel("$x_l$")
plt.ylabel("$x_{nl}$")
plt.title("Joint Posterior Distribution\n $P(X_l, X_{nl}|data)$")
plt.show(block=False)

xl_greater = 0
xnl_greater = 0

# computing the P(X_l > X_nl | data) and P(X_nl >= X_l | data)
for i in range(len(xl_posterior)):
    for j in range(len(xnl_posterior)):
        if i > j:
            xnl_greater += joint_matrix[i,j]
        else:
            xl_greater += joint_matrix[i,j]

print("P(X_l > X_nl | data) = ", np.around(xl_greater, 4))
print("P(X_nl >= X_l | data) = ", np.around(xnl_greater, 4))
```




```
P(X_l > X_nl | data) =  0.9984
P(X_nl >= X_l | data) =  0.0016
```

e) It has been asked to compute the P(*language*|*activation*) using our previous findings from **part b** and provided that P(*language*) = (0.5). To do that again we have to use the Bayes' Rule. We can rewrite the Bayes' Rule as:

$$P(language|activation) = \frac{P(activation|language)P(language)}{P(activation)} \tag{7}$$

For this computation we have to compute the value of $P(activation)$. To to that we have to use the additivity axiom and our previous findings from **part b**. We can write the $P(activation)$ using the additivity axiom, as follows:

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

Code of the expected calculation and its output is provided below.

```
xl_prob = xnl_prob = .5

# using the Bayes rule we find the P(language|activation) by␣
 ↪reverse inference
activation_prob = np.argmax(x_l)/1000 * xl_prob + np.argmax(x_nl)/
 ↪1000 * xnl_prob
```

```
reverse_inference = (np.argmax(x_1)/1000 * xl_prob) /␣
 ↪activation_prob
print("P(language|activation) = ", np.around(reverse_inference, 4))
```

```
P(language|activation) =  0.5833
```

From the output we can infer that reverse inference is convenient in our case. With respect to the output that we have achieved, we can say that; when there is an activity in the Broca's area, it is a task which involves language with probability 58.3%. From this result, we can also state that if there same amount of experiments were conducted, for both language involving and not involving tasks, language involving tasks will have a higher amount.

References

[1] https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/lecture-33-left-and-right-inverses-pseudoinverse/

[2] https://faculty.math.illinois.edu/mlavrov/docs/484-spring-2019/ch4lec4.pdf