# EE 482 Computational Neuroscience

# Homework 1

Cemal Güven Adal

21703986

**Name: Cemal Güven Adal**
**ID: 21703986**

# Question 1

In question a single output measurement is given by:

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & 1 & -1 & 5 \\ 3 & 3 & 0 & 9 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$$

**a)** In part a of question 1 we are asked to find a general solution xn for the system Ax=0. To fasten the calculations first I have calculated the reduced row echelon form of matrix A. All row operations are shown below:

$$A = \begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & 1 & -1 & 5 \\ 3 & 3 & 0 & 9 \end{bmatrix} \qquad x = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 2 & 1 & -1 & 5 \\ 3 & 3 & 0 & 9 \end{bmatrix} \xrightarrow[R3 \leftarrow R3-3R1]{R2 \leftarrow R2-2R1} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 3 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 3 & 3 \end{bmatrix} \xrightarrow{R3 \leftarrow R3-3R2} \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

From equation 1 we can write the given output measurement as:

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = 0 \qquad (2)$$

It can be seen from the equation above x1 and x2 are the pivot variables and x3, x4 are free variables. We can write the pivot variables x1 and x2 in terms of free variables x3 and x4:

$$x1 - x3 + 2x4 = 0 \rightarrow x1 = x3 - 2x4 \quad (3)$$

$$x2 + x3 + x4 = 0 \rightarrow x2 = -x3 - x4 \; (4)$$

**Name: Cemal Güven Adal**
**ID: 21703986**

From combining equation 3 and 4 we can generate the general solution xn as:

$$xn = \begin{bmatrix} x3 - 2x4 \\ -x3 - x4 \\ x3 \\ x4 \end{bmatrix} = x3 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} + x4 \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix} \qquad (5)$$

After doing the calculations I have used python and it's libraries in order to confirm my solutions by computations. In the following code block I have implemented for my computations and explained the reason of each with comments.

```
import matplotlib.pyplot as plt #importing matplot library for plots
import numpy as np # importing numpy library for easier math operations
import math # importing math library for easier computations
import sys
```

In the next code block I have created the A array given in the question and checked its output:

```
#Defining array A which is given in the Homework question 1
A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])
A
```

```
Out[7]: array([[ 1,  0, -1,  2],
               [ 2,  1, -1,  5],
               [ 3,  3,  0,  9]])
```

Then I have created the free variables x3 , x4 and then wrote the general solution in terms of them like my hand made calculations and checked the result of the matrix multiplication with the matrix A if it results in 0:

```
#creating the free variables in order the check the hand solved solution
x3=np.random.rand()
x4=np.random.rand()
#writing general solution xn in terms of free variables x3 and x4
xn= np.array([[x3-2*x4],[-x3-x4],[x3],[x4]])
y=np.dot(A,xn)
print(y)
```

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

**b)** First I have used row operations in order to find reduced row echelon form of the augmented matrix which is created by merging of A and result matrix given in question 1:

$$\left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 2 & 1 & -1 & 5 & 4 \\ 3 & 3 & 0 & 9 & 9 \end{array}\right] \xleftarrow[R3 \leftarrow R3-3R1]{R2 \leftarrow R2-2R1} \left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 3 & 3 & 3 & 6 \end{array}\right]$$

**Name: Cemal Güven Adal**
**ID: 21703986**

$$\left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 3 & 3 & 3 & 6 \end{array}\right] \xleftarrow[R3 \leftarrow R3-3R2]{} \left[\begin{array}{cccc|c} 1 & 0 & -1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \quad (6)$$

As a result of the row operations that have been used to get reduced row echelon form of the augmented matrix we can see that x1 and x2 are the pivot variables and x3 and x4 are the free variables which we have already established in part a of the question 1. From equation 6 we can write pivot variables x1 and x2 in terms of free variables x3 and x4:

$$x1 - x3 + 2x4 = 1 \rightarrow x1 = x3 - 2x4 + 1 \quad (7)$$

$$x2 + x3 + x4 = 2 \rightarrow x2 = 2 - x3 - x4 \quad (8)$$

For finding the particular solution free variables should be assigned to 0. When free variables are assigned to 0 after finding x1 and x2 using equations 7 and 8 we can write the particular solution as:

$$xp = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} \quad (9)$$

After the calculations I have again used python confirm my solutions by computations. In the following code block I have created the array A and I have created the particular solution array which I have found in equation 9. Then I have done matrix multiplication to check I have the result given in question 1 which is: $\begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$

```
#Defining array A which is given in the Homework question 1
A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])
x_p=np.array([[1],[2],[0],[0]]) #Defining particular solution as I have found in handmade calculations
print(A.dot(x_p))
```

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

As it can be seen from above the matrix calculation gives the wanted output which shows the calculations are correct.

**c)** We can find the general solution of Ax=b by using equations which have been found in previous solutions more specifically equation 5,7,8 and 9.

$$xn = \begin{bmatrix} x3 - 2x4 + 1 \\ 2 - x3 - x4 \\ x3 \\ x4 \end{bmatrix} = x3 \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} + x4 \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} \quad (10)$$

Then I have again used python to check hand made calculations by computation. In the following code block I have defined matrix A given in the question, created free variables x3 and x4 then I have used matrix multiplication to check if the output is equal to the result given in question 1:

```
#Defining array A which is given in the Homework question 1
A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])
xall=x_p+xn #Defining all solution as adding particular and general solution
print(A.dot(xall))
```

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

As it can be seen from the above code block the output matches the result of the calculations thus calculations are true.

**d)** Finding inverse of A is another way of solving the equation Ax=b. The given A matrix in question 1 is 3x4 matrix which is not a square matrix meaning we cant find its inverse. Thus other possibility is finding its pseudo inverse. Pseudo inverse of A is denoted as $A^{+}$. Matrix A is rank deficient thus for finding its pseudo inverse we need to use singular value decomposition.

$$A_{mxn} = U_{mxn}\Sigma_{mxn}U_{nxn}$$

$$A^{+} = V\Sigma^{+}U^{T}$$

First we should calculate either $A^{T}A$ or $AA^{T}$. Because A is a 3x4 matrix $AA^{T}$ will be a 3x3 matrix which will be easier to make calculations compared to 4x4 $A^{T}A$ matrix.

$$AA^{T} = \begin{bmatrix}1 & 0 & -1 & 2\\ 0 & 1 & 1 & 1\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}1 & 2 & 3\\ 0 & 1 & 3\\ -1 & -1 & 0\\ 2 & 5 & 9\end{bmatrix} = \begin{bmatrix}6 & 13 & 21\\ 13 & 31 & 54\\ 21 & 54 & 99\end{bmatrix} \qquad (11)$$

Then we need to find eigenvalues and eigenvectors of the vector we have found. For finding the eigenvalues we can use the formula:

$$\det(AA^{T} - \lambda I) = 0$$

$$\det(AA^{T} - \lambda I) = \begin{bmatrix}6-\lambda & 13 & 21\\ 13 & 31-\lambda & 54\\ 21 & 54 & 99-\lambda\end{bmatrix} \qquad (12)$$

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

$$-\lambda^{3} + 136\lambda^{2} - 323\lambda = 0 \qquad (13)$$

$$\lambda_1 = 68 - \sqrt{4301} \quad \lambda_2 = 68 + \sqrt{4301} \quad \lambda_3 = 0 \qquad (14)$$

**Name: Cemal Güven Adal**
**ID: 21703986**

We can find singular values of matrix A by finding the square roots of the eigenvalues of the $AA^T$:

$$\sigma_1 = \sqrt{68 - \sqrt{4301}} \qquad \sigma_2 = \sqrt{68 + \sqrt{4301}} \qquad \sigma_3 = 0 \qquad (15)$$

Finding eigen vectors are close to impossible by hand because of the complexness of the found eigenvalues thus rest will be calculated by python code. First I have checked the eigenvalues and singular values I have calculated by hand via python code:

```
A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])#creating A
Atrans=A.transpose()#taking transpose of A
AAt=np.dot(A,Atrans)#multiplying to matrices
evalues, evectors =np.linalg.eig(AAt)
evalues[2]=0
print("Eigen values:\n",np.round(evalues,3))
singvalues=np.sqrt(evalues) #finding singular values of A
print( "Singular values:\n",singvalues)
U=evectors
print("U\n",U)
```

```
Eigen values:
 [133.582   2.418   0.   ]
Singular values:
 [11.55776837  1.55498883  0.        ]
U
 [[-0.1898465  -0.70019575  0.6882472 ]
 [-0.47607011 -0.54742401 -0.6882472 ]
 [-0.85867081  0.45831524  0.22941573]]
```

As it can be seen from output eigen values and singular values check with hand made calculations. After verifying I have used linalg.svd command for U, Sigma and Vtranspose from A in python for later use:

```
U,Sigma,Vtranspose = np.linalg.svd(A,full_matrices=True)
print("U:\n",U)
print("Sigma:\n",Sigma)
print("Vtranspose:\n",Vtranspose)
```

```
U:
 [[-0.1898465   0.70019575 -0.6882472 ]
 [-0.47607011  0.54742401  0.6882472 ]
 [-0.85867081 -0.45831524 -0.22941573]]
Sigma:
 [1.15577684e+01 1.55498883e+00 4.75023945e-16]
Vtranspose:
 [[-0.32168832 -0.26407196  0.05761637 -0.90744861]
 [ 0.27016145 -0.53217213 -0.80233358  0.00815077]
 [ 0.90740247  0.07564173  0.25203967 -0.3276814 ]
 [ 0.01225465 -0.80083528  0.53797507  0.26286021]]
```

This resuts will later be used but noticing found U from eigen vectors and the U calculated U from svd command are the same shows calclations are true. Later I have created the diagonal sigma matrix from the found singular values in the code below:

```
diagonal=np.zeros((3,4))
diagonal[0,0]=Sigma[0]
diagonal[1,1]=Sigma[1]
diagonal[2,2]=Sigma[2]
print(diagonal)
```

```
[[1.15577684e+01 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.55498883e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 4.75023945e-16 0.00000000e+00]]
```

After creating the diagonal matrix I have found its pseudo inverse for finding pseudo inverse of A.

```
D_plus=np.zeros((4,3)) #taking transpose of diagonal

D_plus[0,0]=1/diagonal[0,0]
D_plus[1,1]=1/diagonal[1,1]
print(D_plus)
```

```
[[0.08652189 0.         0.        ]
 [0.         0.64309143 0.        ]
 [0.         0.         0.        ]
 [0.         0.         0.        ]]
```

After finding the pseudo inverse of sigma I have found the transpose of U and I have taken transpose of Vtranspose to finding V and I have performed matrix multiplication in order to find pseudo of A in the following code:

```
V=np.transpose(Vtranspose)
Utranspose=np.transpose(U) #Finding Utranspose from U  which have been fond from np.svd
Apseudo=V@D_plus@Utranspose#Finding A pseudo
Apseudo
```

```
array([[ 0.12693498,  0.10835913, -0.05572755],
       [-0.23529412, -0.17647059,  0.17647059],
       [-0.3622291 , -0.28482972,  0.23219814],
       [ 0.01857585,  0.04024768,  0.06501548]])
```

After finding Apseudo I have checked my result. For tis I have multiplied A pseudo with b in order to find x

```
#6.soru f
b=np.array([[1],[4],[9]])
x=Apseudo@b#minimum norm
x
```

```
array([[0.05882353],
       [0.64705882],
       [0.58823529],
       [0.76470588]])
```

Thus the result was correct and I have found A pseudo correctly by python and mathematical calculations.

**e)** For finding sparsest solution vectors which have the most number of zers must be found in the solution set. In part b for finding the particular solution I had assigned free variables $x_3$ and $x_4$ 0 which resulted in:

$$\begin{bmatrix}1\\2\\0\\0\end{bmatrix}$$

This resulted in only two non zero entries. Then in python I have validated my solution:

```
sparsolution1 = [[1],[2],[0],[0]]
A@sparsolution1
```

```
array([[1],
       [4],
       [9]])
```

From the output we can see multiplication equals b which means it means sparsolution1 is a solution for the system which has 2 zero entries.

**f)**Given Ax=b has infinitely many solution which is the case in question 1 as discussed before, least norm solution is the solution which $||x||$ is small as possible. To find the least norm solution we can multiply both sides of the given equation which is Ax=b in the question with $A^+$ which results in the least norm solution:

$$x^* = A^+ b \qquad (16)$$

From part d which $A^+$is found I have multiplied it with be in order to find $x^*$.

$$x^* = \begin{bmatrix} 0.127 & 0.108 & -0.557 \\ -0.235 & -0.176 & 0.176 \\ -0.362 & -0.284 & 0.232 \\ 0.018 & 0.040 & 0.065 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$$

$$x^* = \begin{bmatrix} 0.058 \\ 0.647 \\ 0.588 \\ 0.764 \end{bmatrix}$$

Then after hand made calculations I have used python to validate the result.

```
b=np.array([[1],[4],[9]])
x=Apseudo@b#minimum norm
x
```

```
array([[0.05882353],
       [0.64705882],
       [0.58823529],
       [0.76470588]])
```

As it can be seen from the result calculations hold and least norm solution is found.

# Question 2

**a)** In the question its been stated that activation of Broca given language and not activation follow a Bernoulli Distribution. From research in literature we have multiple data to work with. We can use Binomial Distribution in order to calculate the likelihood because of multiple random variables we have from Bernoulli distribution

$$P(data|x) = \begin{Bmatrix} x \; if \; data = 1 \\ (1-x) \; if \; data = 0 \end{Bmatrix} \qquad (17)$$

**Name: Cemal Güven Adal**
**ID: 21703986**

For using binomial disturbution for calculating we can use the formula:

$$P(i) = \binom{n}{i} p^i (1-p)^{n-i} \quad (18)$$

For part a equation becomes:

$$P(data|x_l) = \binom{869}{103} (x_l)^{103} (1-x_l)^{766} \quad (19)$$

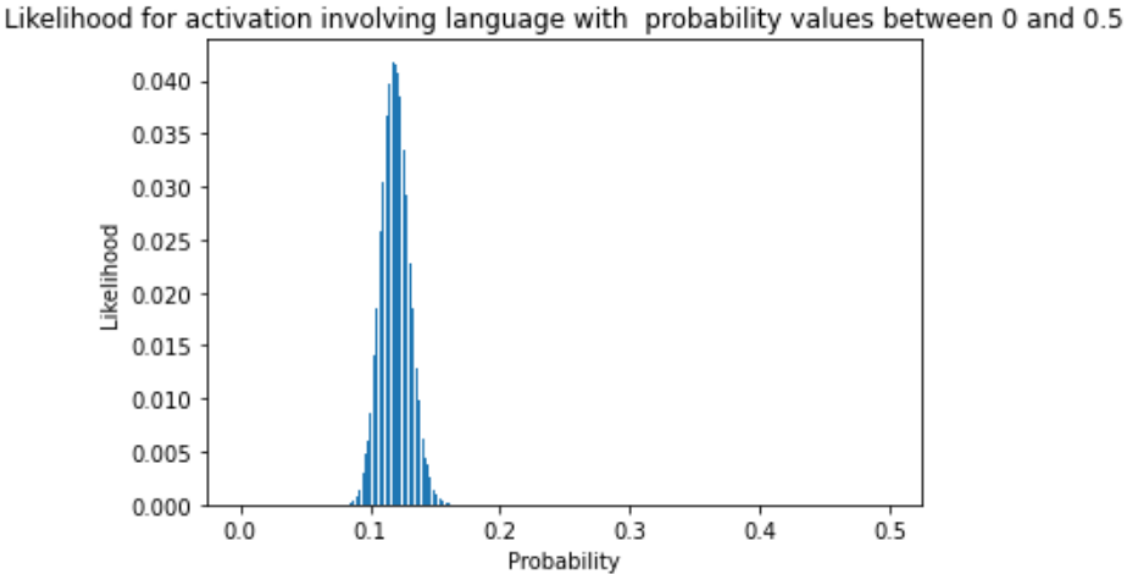
$$P(data|x_{nl}) = \binom{2353}{199} (x_{nl})^{199} (1-x_{nl})^{766} \quad (20)$$

After writing the Bernoulli distubution for the experiment by hand I have used python for computing the function for $x_l$ and $x_{nl}$ values between 0 and 1. The plots are limited in x axis in order to get a better visual , limits are deifned on top of the plot and chosen as when there is nearly 0 likelihood after reaching certain possibility.

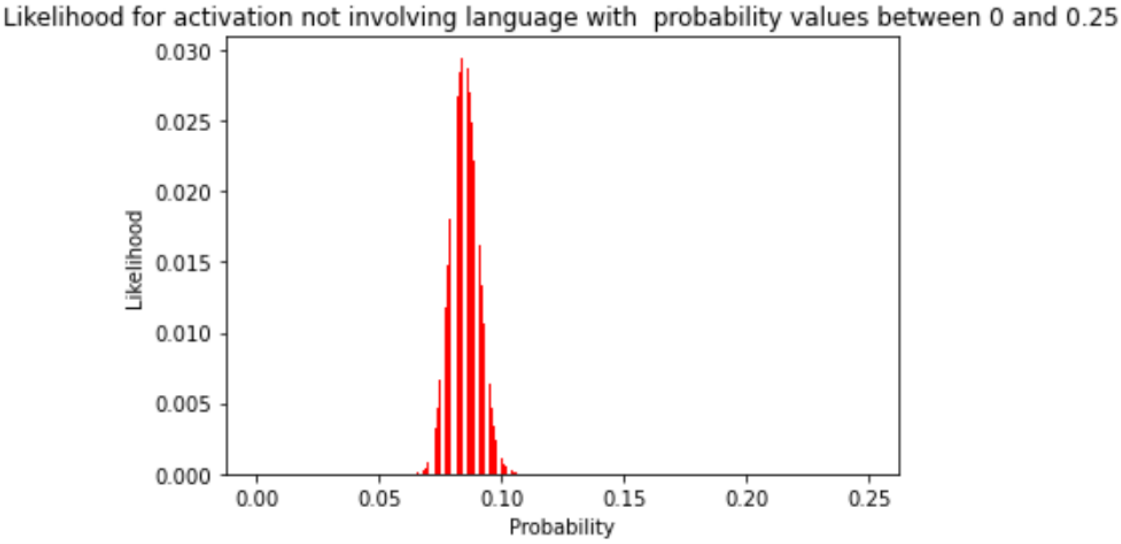
```
#writing function for computing bernoulli
def bernoulli(n,i,p):
    combination=(math.factorial(n))/(math.factorial(i)*math.factorial(n-i))
    ber_result=combination*(p**i)*((1-p)**(n-i))
    return ber_result

possible=np.linspace(0,1,1001) #xl and xnl values between 0 and 1
like=bernoulli(869,103,possible)#language
like1=bernoulli(2353,199,possible)#no language
#Activation plots during language

#plots of likelihood function for involving language 0
plt.figure()

plt.bar(possible[0:500],like[0:500],width=0.0007)#plot between 0 and 0.5
plt.title("Likelihood for activation involving language with  probability values between 0
and 0.5")
plt.xlabel("Probability")
plt.ylabel("Likelihood")
#Activation plots during no language
plt.figure()

plt.bar(possible[0:250],like1[0:250],width=0.0006,color='r')#plot between 0 and 0.25
plt.title("Likelihood for activation not involving language with  probability values between
0 and 0.25")
plt.xlabel("Probability")
plt.ylabel("Likelihood")
```

Plots for likelihood functions for activation with language and activation with no language can be seen below:

**Name: Cemal Güven Adal**
**ID: 21703986**




Plot 1: Likelihood distribution for activation involving language




Plot 2: Likelihood distribution for activation not involving language

**b)** In this question I am expected to find $x_l$ and $x_{nl}$ which maximizes likelihood functions. To do this I have used maximum likelihood estimation method. First I have used the equation 18 then took his derivative respect to p in order to find the value for equation to be 0 for finding max value:

$$P(i) = \binom{n}{i} p^i (1-p)^{n-i} \qquad (18)$$

$$\log\big(P(i)\big) = \log\binom{n}{i} + i\log(p) + (n-i) * \log(1-p) \qquad (21)$$

$$\frac{\partial}{\partial p}\big(P(i)\big) = \frac{i}{p} + \frac{(n-i)}{1-p} \qquad (22)$$

$$\frac{i}{p} + \frac{(n-i)}{1-p} = 0 \quad (23)$$

$$p = \frac{i}{n} \quad (24)$$

For xl:

$$x_{l(mle)} = \frac{103}{869} = 0.1185 \quad (25)$$

$$x_{nl(mle)} = \frac{199}{2353} = 0.0845 \quad (26)$$

After this I have used python in order to find mle values for xl and xnl:

```
#b findig MLE
a=np.argmax(like)
b=np.argmax(like1)
print("MLE of activation with language",possible[a])
print("MLE of activation with no language",possible[b])
```

```
MLE of activation with language 0.11900000000000001
MLE of activation with no language 0.085
```

Python result and calculations check thus the MLE calculations for $x_l$ and $x_{nl}$ are true.

**c)** Bayes rule can be used for finding P(x|data) which is asked in the question:

$$P(x|data) = \frac{P(data|x)P(x)}{P(data)} \quad (27)$$

$$P(data) = \sum P(data|x)P(x) \quad (28)$$

Prior P(x) is given as 1 in the question. Thus equations to find posterior pdf become simpler:

$$P(x|data) = \frac{P(data|x)}{P(data)}$$

$$P(data) = \sum P(data|x)$$

For finding cumulative distribution function I have used posterior distribution function. From cumulative distribution I have found the upper and lower confidence bounds by finding the probability values which gives 0.025 and 0.975 in the cumulative distribution function. Because this is a discrete cumulative distribution I have given an interval around these values for finding corresponding probability values because there may not be the exact values in the cdf. Python code can be seen below:

```
#c
#finding sum of likelihood
suml1=0
suml2=0
cdf=np.zeros(1001)
cdf1=np.zeros(1001)
conf=np.zeros(2)
conf1=np.zeros(2)
```

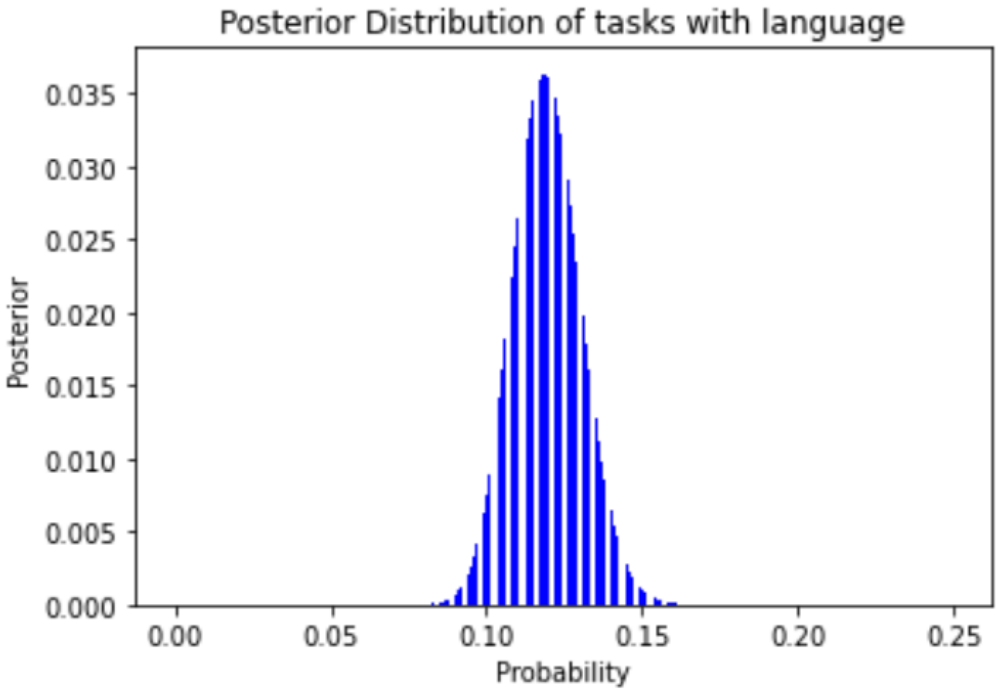
```
suml1 = np.sum(like)


suml2=np.sum(like1)

#finding posteriorpfg

postpdf=like/suml1
postpdf1=like1/suml2

#plot for posterior distribution with language

plt.figure()
plt.bar(possible[0:250],postpdf[0:250],width=0.0006,color='b')#plot between 0 and 0.25
plt.title("Posterior Distribution of tasks with language")
plt.xlabel("Probability")
plt.ylabel("Posterior")

#posterior plot of no language

plt.figure()
plt.bar(possible[0:250],postpdf1[0:250],width=0.0006,color='r')#plot between 0 and 0.25
plt.title("Posterior Distribution of tasks with no language")
plt.xlabel("Probability")
plt.ylabel("Posterior")
for i in range(0, len(like)):
  cdf[i]=np.sum(postpdf[0:i+1])
for i in range(0, len(like1)):
  cdf1[i]=np.sum(postpdf1[0:i+1])

#cdf plot language

plt.figure()
plt.bar(possible,cdf,width=0.0007,color='b')
plt.title("Cumulative Distribution of tasks with language")
plt.xlabel("Probability")
plt.ylabel("CDF")

#cdf plot no language


plt.figure()
plt.bar(possible,cdf,width=0.0007,color='r')
plt.title("Cumulative Distribution of tasks with no language")
plt.xlabel("Probability")
plt.ylabel("CDF")
print(cdf)


#calculating confidence bounds
```

**Name: Cemal Güven Adal**
**ID: 21703986**

```
#for language
for i in range(0,len(cdf)):
    if 0.023<cdf[i]<0.026:
        conf[0]=possible[i]
    elif 0.973<cdf[i]<0.975:
        conf[1]=possible[i]

#for no language
for i in range(0,len(cdf)):
    if 0.018<cdf1[i]<0.030:
        conf1[0]=possible[i]
    elif 0.970<cdf1[i]<0.980:
        conf1[1]=possible[i]
print("conidence interval for tasks involving lanuage","lower bound:",conf[0],"    upper
bound:", conf[1])
print("conidence interval for tasks not involving lanuage","lower bound:",conf1[0],"
upper bound:", conf1[1])
```

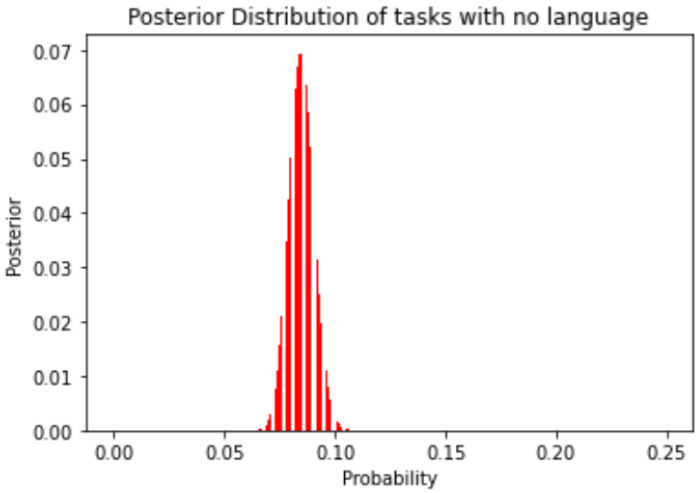
```
conidence interval for tasks involving lanuage lower bound: 0.098     upper bound: 0.14100000000000001
conidence interval for tasks involving lanuage lower bound: 0.073     upper bound: 0.096
```

As it can be seen from the output of the code the lower bound for tasks involving language is 0.098 , upper bound is 0.141. For tasks not involving language lower bound is 0.073 and upper bound is 0.096.

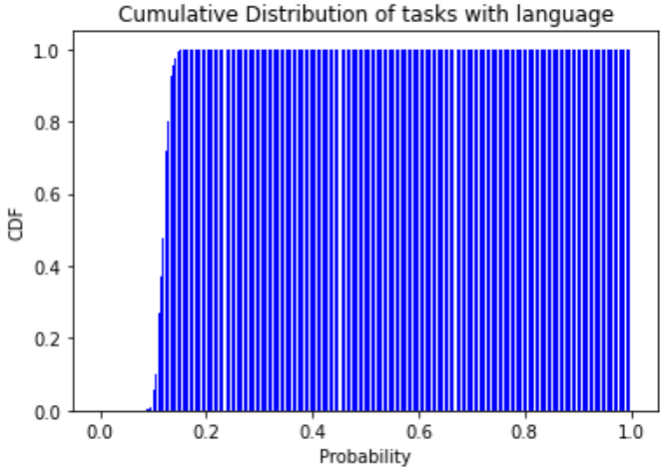



Plot 3:Posterior Distribution of  tasks involving language

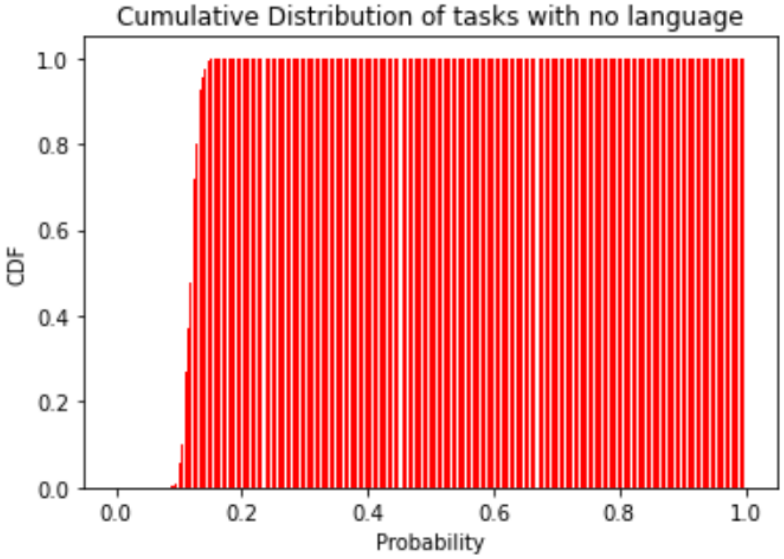
**Name: Cemal Güven Adal**
**ID: 21703986**




Plot 4:Posterior Distribution of  tasks not involving language




Plot 5: Cumulative Distribution of tasks involving language

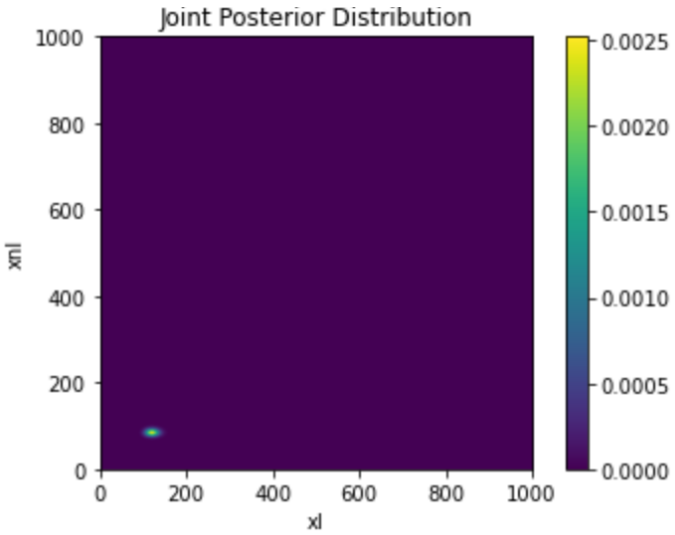



Plot 6: Cumulative Distribution of tasks not involving language

**Name: Cemal Güven Adal**
**ID: 21703986**

**d)** It is been asked to joint posterior distribution $P(X_l,X_{nl}|data)$. For finding joint distribution posterior distribution functions need to be multiplied. But both of them is in the form of a row vector at the moment thus matrix multiplication cant be applied. In order to resolve this either one of them can be chosen to be taken its transpose of. I have chosen $x_{nl}$. With this multiplication horizonal side of the image plotted will be for $x_l$ and vertical side will be for $x_{nl}$. Python code for multiplication and plotting the image can be seen below:

```
postpdf1tr=np.transpose(postpdf1)
postpdf1tr=postpdf1tr.reshape(1001,1)
postpdf=postpdf.reshape(1,1001)
jointpdf=np.dot(postpdf1tr,postpdf)

plt.figure()
plt.imshow(jointpdf, origin='lower')
plt.title("Joint Posterior Distribution")
plt.xlabel("xnl")
plt.ylabel("xl")
plt.colorbar()
```




Plot 7: Joint Posterior Distribution of xl and xnl

I have used origin lower command in order to reverse the first found plot which had the reversed origin because of the matrix multiplication. As we can see from small green blue shape from the picture probabilities gathered around that location. From the plot 3 and plot 4 we can see posterior distribution of xl is wider compared to xnl thus we can see from the figure xl side of the are which probabilities are gathered widened compared to xnl(vertical part). Having a small gathering of the high points into one location shows that the maximum likelihood estimation we had done in part b is with less error. If high point are was large we could be less sure about the maximum likelihood estimation. For finding $P(Xl > Xnl|data)$ and $P(Xl \leq Xnl|data)$ points I have performed the nested loop. Code and result can be seen below:

```
#P(Xl > Xnl|data)and P(Xl > Xnl|data) finding from matrix
for i in range(0,len(postpdf1)):
    for k in range(0,len(postpdf1)):
```

```
    if i<k:
        postgreater=postgreater+jointpdf[i,k]
    else:
        post1greater=post1greater+jointpdf[i,k]
print(" P(Xl > Xnl|data):",postgreater)
print("P(Xl < Xnl|data):",post1greater)
```

```
 P(Xl > Xnl|data): 0.9978520275861292
P(Xl < Xnl|data): 0.002147972413864097
```

From the result P(Xl > Xnl|data)=0.99785 and P(Xl < Xnl|data)=0.00214.

**e)** In the question P(language) given as 0.5. To calculate P(language|activation) we need to use bayes rule. From total probability theorem we can write P(activation) as it can be seen from equation 30. P(nolanguage) is 0.5 from 1-P(language). Then equation becomes:

$$P(language|activation) = \frac{P(activation|language)P(language)}{P(activation)} \quad (29)$$

$$P(activation) = P(activation|language)P(language) + P(activation|nolanguage)P(nolanguage) \quad (30)$$

$$P(language|activation) = \frac{0.1185*0.5}{(0.1185+0.0845)*0.5} = 0.58374 \quad (31)$$

For validating my calculations I have performed the computations on python. Code can be seen below:

```
#f
mle1=a
mle2=b
plang=0.5
langact=mle1*plang/((mle1+mle2)*plang)
print(langact)
```

```
0.5833333333333334
```

Output is same with my calculations thus P(language|activation) = 0.5833. From the result we can deduce that when there is an activity Broca are there is a %58 chance which this task involves language and %42 chance that this task involves no language. From this it can be said that reverse inference can be used yet when using we shouldn't be so confident in it. This is because using language task and no language task during activation is close to one another even though language task is more likely.

**Name: Cemal Güven Adal**
**ID: 21703986**

## References

Greenberg, M. D. (1988). *Advanced engineering mathematics*. Englewood Cliffs, NJ: Prentice-Hall Internat.

Bertsekas, D. P., & Tsitsiklis, J. N. (2008). *Introduction to probability: Dimitri P. Bertsekas and John N. Tsitsikis*. Belmont, Massachussets: Athena Scientific.

## Appendices

### Code

```
#!/usr/bin/env python
# coding: utf-8



import matplotlib.pyplot as plt #importing matplot library for plots
import numpy as np # importing numpy library for easier math operations
import math # importing math library for easier computations
import sys


def CemalGuven_Adal_21703986_hw1(question):
    if question == '1' :
        #--------------------------------------------------------------------------Question1----------
----------------------------------------------------
        #Question 1 A----------------------------------------
        #Defining array A which is given in the Homework question 1
        A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])
        A




        #creating the free variables in order the check the hand solved solution
        x3=np.random.rand()
        x4=np.random.rand()
        #writing general solution xn in terms of free variables x3 and x4
        xn= np.array([[x3-2*x4],[-x3-x4],[x3],[x4]])
        y=np.dot(A,xn)
        print(y)

        #Question1 B-----------------------------------------------



        #Defining array A which is given in the Homework question 1
        A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])
        x_p=np.array([[1],[2],[0],[0]]) #Defining particular solution as I have found in
handmade calculations
```

**Name: Cemal Güven Adal**
**ID: 21703986**

```
print(A.dot(x_p))

#Question1 C----------------------------------------------


#Defining array A which is given in the Homework question 1
A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])
xall=x_p+xn #Defining all solution as adding particular and general solution
print(A.dot(xall))


#Question1 D----------------------------------------------

A= np.array([[1,0,-1,2],[2,1,-1,5],[3,3,0,9]])#creating A
Atrans=A.transpose()#taking transpose of A
AAt=np.dot(A,Atrans)#multiplying to matrices
evalues, evectors =np.linalg.eig(AAt)
evalues[2]=0
print("Eigen values:\n",np.round(evalues,3))
singvalues=np.sqrt(evalues) #finding singular values of A
print( "Singular values:\n",singvalues)
U=evectors
print("U\n",U)


AtA=np.dot(Atrans,A)#multiplying to matrices
evalues, evectors =np.linalg.eig(AtA)
evalues[2:]=0
print(np.real(np.round(evalues,3)))
singvalues=np.sqrt(evalues) #finding singular values of A
print(np.real(np.round(singvalues,3)))
V=evectors
print(np.real(np.round(evectors,3)))




U,Sigma,Vtranspose = np.linalg.svd(A,full_matrices=True)
print("U:\n",U)
print("Sigma:\n",Sigma)
print("Vtranspose:\n",Vtranspose)



diagonal=np.zeros((3,4))
diagonal[0,0]=Sigma[0]
diagonal[1,1]=Sigma[1]
diagonal[2,2]=Sigma[2]
print(diagonal)
```

```
    aplus=U@diagonal@Vtranspose
    Sigma=np.round(Sigma,1)
    print(np.round(aplus,1))



    D_plus=np.zeros((4,3)) #taking transpose of diagonal

    D_plus[0,0]=1/diagonal[0,0]
    D_plus[1,1]=1/diagonal[1,1]
    print(D_plus)




    #check
    D_plus@diagonal




    V=np.transpose(Vtranspose)
    Utranspose=np.transpose(U) #Finding Utranspose from U  which have been fond from
np.svd
    Apseudo=V@D_plus@Utranspose#Finding A pseudo
    Apseudo




    #6.soru f
    b=np.array([[1],[4],[9]])
    x=Apseudo@b#minimum norm
    x



    sparsolution1 = [[1],[2],[0],[0]]
    A@sparsolution1


  elif question == '2' :

    #------------------QUESION2-------------------------------------------------------------------
----------------------------

    #writing function for computing bernoulli
    def bernoulli(n,i,p):
```

```
        combination=(math.factorial(n))/(math.factorial(i)*math.factorial(n-i))
        ber_result=combination*(p**i)*((1-p)**(n-i))
        return ber_result

    possible=np.linspace(0,1,1001) #xl and xnl values between 0 and 1
    like=bernoulli(869,103,possible)#language
    like1=bernoulli(2353,199,possible)#no language
    #Activation plots during language

    #plots of likelihood function for involving language 0
    plt.figure()

    plt.bar(possible[0:500],like[0:500],width=0.0007)#plot between 0 and 0.5
    plt.title("Likelihood for activation involving language with  probability values between
0 and 0.5")
    plt.xlabel("Probability")
    plt.ylabel("Likelihood")
    #Activation plots during no language
    plt.figure()

    plt.bar(possible[0:250],like1[0:250],width=0.0006,color='r')#plot between 0 and 0.25
    plt.title("Likelihood for activation not involving language with  probability values
between 0 and 0.25")
    plt.xlabel("Probability")
    plt.ylabel("Likelihood")




    #b findig MLE
    a=np.argmax(like)
    b=np.argmax(like1)
    print("MLE of activation with language",possible[a])
    print("MLE of activation with no language",possible[b])




    #c
    #finding sum of likelihood
    suml1=0
    suml2=0
    cdf=np.zeros(1001)
    cdf1=np.zeros(1001)
    conf=np.zeros(2)
    conf1=np.zeros(2)
    suml1 = np.sum(like)
```

```
    suml2=np.sum(like1)

    #finding posteriorpfg

    postpdf=like/suml1
    postpdf1=like1/suml2

    #plot for posterior distribution with language

    plt.figure()
    plt.bar(possible[0:250],postpdf[0:250],width=0.0006,color='b')#plot between 0 and
0.25
    plt.title("Posterior Distribution of tasks with language")
    plt.xlabel("Probability")
    plt.ylabel("Posterior")

    #posterior plot of no language

    plt.figure()
    plt.bar(possible[0:250],postpdf1[0:250],width=0.0006,color='r')#plot between 0 and
0.25
    plt.title("Posterior Distribution of tasks with no language")
    plt.xlabel("Probability")
    plt.ylabel("Posterior")
    for i in range(0, len(like)):
      cdf[i]=np.sum(postpdf[0:i+1])
    for i in range(0, len(like1)):
      cdf1[i]=np.sum(postpdf1[0:i+1])

    #cdf plot language

    plt.figure()
    plt.bar(possible,cdf,width=0.0007,color='b')
    plt.title("Cumulative Distribution of tasks with language")
    plt.xlabel("Probability")
    plt.ylabel("CDF")

    #cdf plot no language



    plt.figure()
    plt.bar(possible,cdf,width=0.0007,color='r')
    plt.title("Cumulative Distribution of tasks with no language")
    plt.xlabel("Probability")
    plt.ylabel("CDF")
    print(cdf)



    #calculating confidence bounds
    #for language
```

```
    for i in range(0,len(cdf)):
        if 0.023<cdf[i]<0.026:
            conf[0]=possible[i]
        elif 0.973<cdf[i]<0.975:
            conf[1]=possible[i]

    #for no language
    for i in range(0,len(cdf)):
        if 0.018<cdf1[i]<0.030:
            conf1[0]=possible[i]
        elif 0.970<cdf1[i]<0.980:
            conf1[1]=possible[i]
    print("conidence interval for tasks involving lanuage","lower bound:",conf[0]," upper bound:", conf[1])
    print("conidence interval for tasks not involving lanuage","lower bound:",conf1[0]," upper bound:", conf1[1])




    postpdf1tr=np.transpose(postpdf1)
    postpdf1tr=postpdf1tr.reshape(1001,1)
    postpdf=postpdf.reshape(1,1001)
    jointpdf=np.dot(postpdf1tr,postpdf)

    plt.figure()
    plt.imshow(jointpdf, origin='lower')
    plt.title("Joint Posterior Distribution")
    plt.xlabel("xl")
    plt.ylabel("xnl")
    plt.colorbar()
    postgreater=0
    post1greater=0
    #P(Xl > Xnl|data)and P(Xl > Xnl|data) finding from matrix
    for i in range(0,len(postpdf1)):
        for k in range(0,len(postpdf1)):
            if i<k:
                postgreater=postgreater+jointpdf[i,k]
            else:
                post1greater=post1greater+jointpdf[i,k]
    print(" P(Xl > Xnl|data):",postgreater)
    print("P(Xl < Xnl|data):",post1greater)
```

**Name: Cemal Güven Adal**
**ID: 21703986**

```
    #f
    mle1=a
    mle2=b
    plang=0.5
    langact=mle1*plang/((mle1+mle2)*plang)
    print(langact)



question=input("enter question number")
CemalGuven_Adal_21703986_hw1(question)
```