CS 6140: Assignment 1

Luke Davidson

October 3 2022

Exercise 1

a.) There are two main parts to this question. First, player one winning the game means that the coin landed on "Heads" a total of "n", or in this problem 8, times before "Tails" did "n", or 8, times. The second part is that the game was interrupted, meaning player one received at least "m", or in this problem 4, heads, and player two received at least "l", or 6, tails.

Combining these two aspects of the problem essentially leads to the probability that player one wins is the probability that player one receives 4 heads (n-m) before player two receives 2 (n-l) tails. That is:

$$P(P1wins) = P(Heads = 4 \cap Tails = \{0, 1\}) \tag{1.1}$$

which is equivalent to

$$P(P1wins) = P(Heads = 4 \cap Tails = 0) + P(Heads = 4 \cap Tails = 1) \tag{1.2}$$

Now we have to think of the scenarios in which these could occur for each of the terms on the right side of equation 1.2. The first term, $P(Heads = 4 \cap Tails = 0)$ is simple:

$$P(Heads = 4 \cap Tails = 0) = P(4HeadsInARow) = \frac{1}{2^4} = \frac{1}{16}$$

The second term has a few more possible scenarios. The set of scenarios would be:

$$\{THHHH, HTHHH, HHTHH, HHHTH\}$$

each having an equal probability of occurring individually, $\frac{1}{2^5}$, resulting in:

$$P(Heads = 4 \cap Tails = 1) = 4 \cdot \frac{1}{2^5} = \frac{4}{32} = \frac{1}{8}$$

Plugging these in to equation 1.2 leads to our answer:

$$P(P1wins) = \frac{1}{16} + \frac{1}{8} = \frac{3}{16}$$

b.) In terms of m, n and l, the general expression is similar to what was described in part a. In order for player one to win the game if it was to be continued essentially boils down to player one winning $n - m$ games before player two wins $n - l$ games, or player one reaching n games before player two wins an additional $n - l$ games. This means we can allow player two to win as little as l games, and as many as $n - 1$ games. Thus:

$$P(P1wins) = P(Heads = n \cap Tails = \{l, l+1, ..., n-1\}) \tag{1.3}$$

The first part, $P(Heads = n)$, is easy to compute. Similar to part a, it will be equal to:

$$P(Heads)^{NumberAdditionalWinsNeeded} = \frac{1}{2}^{n-m} \tag{1.4}$$

The second part is a little more involved. First we can see that because there is a $Tails = \{l, l+1, ..., n-1\}$ in equation 1.3, that will essentially be pulled out as a sum from l to $n - 1$, similar to:

$$P(P1wins) = \sum_{i=1}^{n-1} P(Heads = n \cap T_i) \tag{1.5}$$

We now need to come up with a way to represent the total number of possible locations of Tails, while still following the conditions that player one wins. We notice that in general, Tails can be flipped a total of "i" times, in accordance with the summation above, which can be placed in the amount of additional wins needed by player one + i - 1 places, since the last flip will have to be a heads. This can be represented by a binomial coefficient defined as:

$$\frac{m!}{n!(m-n)!} = \frac{((n-m)+i-1)!}{i!(n-m-1)!} \tag{1.6}$$

Multiply this with the probability of a tails raised to the number of tails, or $\frac{1}{2}^i$ leads to our general equation:

$$P(P1wins) = \sum_{i=1}^{n-1} (\frac{1}{2}^{n-m}) \cdot (\frac{((n-m)+i-1)!}{i!(n-m-1)!}) \cdot (\frac{1}{2}^i)$$

c.) When l and m are kept constant, the probability that player one will win as n increases will become smaller and smaller. It is essentially giving player two more and more of a chance to get tails enough times to beat player one. If player one has to win n games, where n is large, the chance they win that many games before player two becomes lower. Before this problem, I would have assumed the chance player one wins as n increases would be relatively stable, although it decreases very quickly.

Exercise 2

Answer: $P(A) \neq P(A \mid B) + P(A \mid B^c)$ for all A and B given the conditions.

To prove this, we can break up and expand the right side of this equation using the following two relationships:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{2.1}$$

$$P(A \mid B^c) = \frac{P(A \cap B^c)}{P(B^c)} = \frac{P(A) - P(A \cap B)}{1 - P(B)} \tag{2.2}$$

To prove the summation of these two expansions does not equal $P(A)$ for all A and B under the given conditions, we can consider the following example:

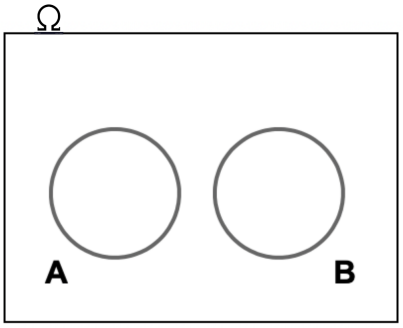


Figure 1: Visual of a discrete probability space fitting the description of Q2

In correspondence with Q2, this is a representation of a discrete probability space where $A \subseteq \Omega$ and $B \subseteq \Omega$. As one can see, $P(A \cap B) = 0$ for this probability space. This simplifies the right hand summation of equations 2.1 and 2.2 to:

$$\frac{0}{P(B)} + \frac{P(A)}{1 - P(B)} = \frac{P(A)}{1 - P(B)} \tag{2.3}$$

Equation 2.3 clearly does not equal $P(A)$ due to the denominator of $1 - P(B)$, thus making it an example of when the original equation in question does not hold.

Exercise 3

a.) The individual PMF's for X and Y are respectively seen below:

		Y			
	p(X|Y)	1	2	3	4
X	1	0.25	0.25	0.25	0.25
	2	0.25	0.25	0.25	0.25
	3	0.25	0.25	0.25	0.25
	4	0.25	0.25	0.25	0.25

		Y			
	p(Y|X)	1	2	3	4
X	1	1	0	0	0
	2	0.5	0.5	0	0
	3	0.3333333333	0.3333333333	0.3333333333	0
	4	0.25	0.25	0.25	0.25

Figure 2: Individual PMF's of X and Y, respectively

Combining these two PMF's lead to the joint PMF, seen below:

		Y			
	p(X,Y)	1	2	3	4
X	1	0.25	0	0	0
	2	0.125	0.125	0	0
	3	0.08333333333	0.08333333333	0.08333333333	0
	4	0.0625	0.0625	0.0625	0.0625

Figure 3: Joint PMF, p(X,Y)

We can confirm this by summing all of the values, which should, and does, = 1.

b.) The marginal PMF for Y is defined as

$$P_Y(y) = \sum_{x_i \in R_X} P_{XY}(x_i, y) \tag{3.1}$$

This essentially becomes the summation of each column in Figure ???:

Marginal PMF of Y	0.5208333333	y = 1
	0.2708333333	y = 2
P_Y(y)	0.1458333333	y = 3
	0.0625	y = 4

Figure 4: Marginal PMF of Y

c.) Conditional PMF of X where Y = 2 is defined as:

$$P_{X|Y=2}(x) = \frac{p_{XY}(x, 2)}{p_Y(2)} \tag{3.2}$$

which results in:

Conditional PMF of X	0	x = 1
	0.4615384615	x = 2
where Y = 2	0.3076923077	x = 3
	0.2307692308	x = 4

Figure 5: Conditional PMF of X, where Y = 2

d.) Mary buying at least 2 but no more than 3 books means that Y = 2 or 3. We can calculate conditional mean and variance of these two scenarios by using the following two equations:

$$\mu_{X|Y=y} = \sum_x x \cdot g(x \mid y) \tag{3.3}$$

$$\sigma^2_{Y|x} = \sum_x [x - \mu_{X|y}]^2 \cdot g(x \mid y) \tag{3.4}$$

where

$$g(x \mid y) = \frac{p(x,y)}{p_Y(y)} \tag{3.5}$$

$g(x \mid y)$:

		Y			
	g(x|y)	1	2	3	4
X	1	0.48	0	0	0
	2	0.24	0.4615384615	0	0
	3	0.16	0.3076923077	0.5714285714	0
	4	0.12	0.2307692308	0.4285714286	1

Figure 6: $g(x \mid y)$

Using these equations, our conditional mean of X for both Y = 2 and Y = 3 are:

	Y	
	2	3
	0	0
	0.9230769231	0
	0.9230769231	1.714285714
	0.9230769231	1.714285714
cond mean x =	2.769230769	3.428571429

Figure 7: Conditional Mean of X for Y=2 and Y=3

and thus a calculation of conditional variance:

e.) To calculate Mary's total expected expenditure for her trip, we want to know the total expected cost of each book. This will lead to our relationship:

xi	xi - mu_x,y=2	<-- squared	<-- * g(xi|y=2)	
1	-1.769230769	3.130177515	0	
2	-0.7692307692	0.5917159763	0.2730996814	
3	0.2307692308	0.05325443787	0.01638598088	
4	1.230769231	1.514792899	0.3495675922	
			0.6390532544	= cond var x, y = 2
xi	xi - mu_x,y=3	<-- squared	<-- * g(xi|y=3)	
1	-2.428571429	5.897959184	0	
2	-1.428571429	2.040816327	0	
3	-0.4285714286	0.1836734694	0.1049562682	
4	0.5714285714	0.3265306122	0.139941691	
			0.2448979592	= cond var x, y = 3

Figure 8: Conditional Variance of X for Y=2 and Y=3

$$TotalCost = \sum_{i=1}^{n} C_i \tag{3.6}$$

The cost of the "ith" book is a function of Y, such that we want $E(C \mid Y)$. Relating back to part d, we know the conditional means of X, total hours spent in the book store, in relation to Y, total books bought. Knowing the mean cost of each book, we can relate them all to solve for her expected total cost given hours in the book store:

$$TotalCost = \sum_{Y=\{2,3\}} \mu_X \cdot 3 = 2.769 \cdot 3 = \$8.31 \tag{3.7}$$

Exercise 4

We start by identifying the conditions in which $X = Y_0$ and $X = Y_1$. Referencing the original equation,

$$X = Z \cdot Y_1 + (1 - Z) \cdot Y_0 \tag{4.1}$$

it is evident that for $X = Y_1$, $Z = 1$. Similarly, for $X = Y_0$, $Z = 0$. Thus we can make the two key relations:

$$P(X = Y_1) = P(Z = 1) = \alpha$$
$$P(X = Y_0) = P(Z = 0) = 1 - \alpha$$

because the relationship $P(X = Y_1), P(X + Y_0) = \alpha, 1 - \alpha$ is defined in the problem statement. We can now expand our original equation (4.1) for any arbitrary selection of X, defined x:

$$P(X = x) = P(Z = 1) \cdot P(x = Y_1) + P(Z = 0) \cdot P(x = Y_0) \tag{4.2}$$

where $P(X = x)$ is the density of X, and $P(x = Y_1)$ and $P(x = Y_0)$ are the densities of Y_1 and Y_0, respectfully. Replacing $P(Z = 1)$ and $P(Z = 0)$ with the relationships defined above, we prove that the density of X is a mixture of the densities of Y_1 and Y_0 with α and $1 - \alpha$ as the respective proportions:

$$P(X = x) = \alpha \cdot P(x = Y_1) + (1 - \alpha) \cdot P(x = Y_0) \tag{4.3}$$

Exercise 5

a.) ML estimate of λ

ML estimate of λ is defined as:

$$f_{ML} = argmax(p(D \mid f)) \tag{5.1}$$

For a Poisson distribution,

$$p(D \mid f) = p(X \mid \lambda) = \frac{\lambda^x \cdot e^{-\lambda}}{x!} \tag{5.2}$$

where X is the set of data points in the problem. Because X is a set, the likelihood of λ can be defined as

$$L(\lambda) = \prod_{i=1}^{n} p(x_i \mid \lambda) \tag{5.3}$$

Inputting equation 5.2 in to equation 5.3 yields

$$L(\lambda) = \prod_{i=1}^{n} \frac{\lambda^{x_i} \cdot e^{-\lambda}}{x_i!} = \frac{e^{-n\lambda} \cdot \lambda^{\sum_{i=1}^{n} x_i}}{\prod_{i=1}^{n} x_i!} \tag{5.4}$$

Taking the log of this function, or the log likelihood (denoted $LL(\lambda)$), simplifies it to

$$LL(\lambda) = log(e^{-n\lambda}) + log(\lambda^{\sum_{i=1}^{n} x_i)} - log(\prod_{i=1}^{n} x_i!) \tag{5.5}$$

which further simplifies to

$$LL(\lambda) = -n\lambda + (\sum_{i=1}^{n} x_i) \cdot log(\lambda) - log(\prod_{i=1}^{n} x_i!) \tag{5.6}$$

We now want to maximize this equation to find $\hat{\lambda}$. We will integrate with respect to λ then set equal to 0 at $\lambda = \hat{\lambda}$:

$$\frac{d(LL(\lambda))}{d\lambda} = -n + \frac{1}{\lambda} \sum_{i=1}^{n} x_i \tag{5.7}$$

It is now trivial that setting equal to 0 and solving for $\lambda = \hat{\lambda}$ yields

$$\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5.8}$$

In our problem, there are 72 accidents over the next 8 days, so $n = 8$ and $\sum_{i=1}^{n} x_i = 72$, yielding a max likelihood estimate of

$$\hat{\lambda}_{ML} = \frac{72}{8} = 9$$

b.) The max a posteriori, or MAP, estimate is similar to above, except we are multiply a $p(f)$ in to equation 5.1, so it becomes

$$f_{MAP} = argmax(p(D \mid f) \cdot p(f)) = argmax(\frac{e^{-\lambda}\lambda^{x}}{x!} \cdot \frac{e^{-\lambda/2}}{2}) \tag{5.9}$$

Equation 5.4 now becomes

$$L(\lambda) = \frac{e^{-1.5n\lambda} \cdot \lambda^{\sum_{i=1}^{n} x_i}}{2\prod_{i=1}^{n} x_i!} \tag{5.10}$$

Taking the log and differentiating with respect to λ yields

$$\frac{-3n}{2} + \frac{\sum_{i=1}^{n} x_i}{\lambda} \tag{5.11}$$

Finally, setting equal to 0 and solving for $\hat{\lambda}$ yields

$$\hat{\lambda}_{MAP} = \frac{2\sum_{i=1}^{n} x_i}{3n} = \frac{2 \cdot 72}{3 \cdot 8} = 6$$

c.) The Bayes estimate takes the MAP estimate one step further. Equation 5.10 is very important for a Bayes estimate. The first step is to find the posterior distribution:

$$p(f \mid D) = \frac{p(D \mid f) \cdot p(f)}{\int_0^{\infty} p(D \mid f) \cdot p(f) df} \tag{5.12}$$

where $p(D \mid f) \cdot p(f)$ will equal $L(\lambda)$ in equation 5.10. Plugging this in to equation 5.12 yields

$$p(f \mid D) = \frac{\frac{e^{-1.5n\lambda} \cdot \lambda^{\sum_{i=1}^{n} x_i}}{2\prod_{i=1}^{n} x_i!}}{\int_0^{\infty} \frac{e^{-1.5n\lambda} \cdot \lambda^{\sum_{i=1}^{n} x_i}}{2\prod_{i=1}^{n} x_i!} df} \tag{5.13}$$

In our example, we know $n = 8$ and $\sum_{i=1}^{n} x_i = 72$. This simplifies the numerator of equation 5.10 to

$$e^{-12\lambda} \cdot \lambda^{72} \tag{5.14}$$

We can now relate this to a gamma distribution with parameters $\alpha = 73$ and $\beta = 12$, where we know the mean is equal to α/β. Thus:

$$\hat{\lambda}_{Bayes} = \frac{\alpha}{\beta} = \frac{73}{12} = 6.083$$

Exercise 6

a.) This MAP estimate begins with finding the likelihood function, $L(\theta)$, a function of both the distribution of $X_{i=1}^{n}$ and Θ:

$$L(\theta) = \prod_{i=1}^{n} P(X \mid \theta, \sigma_0^2) \cdot P(\theta \mid \mu, \sigma^2) \tag{6.1}$$

where, due to the Gaussian distribution,

$$P(X \mid \theta, \sigma_0^2) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\frac{-(X-\theta)^2}{2\sigma_0^2}} \tag{6.2}$$

$$P(\theta \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(\theta-\mu)^2}{2\sigma^2}} \tag{6.3}$$

Plugging these in to equation 6.1 results in:

$$L(\theta) = \prod_{i=1}^{n} \frac{1}{2\pi\sqrt{\sigma^2\sigma_0^2}} \cdot e^{\frac{-(X_i-\theta)^2}{2\sigma_0^2} - \frac{(\theta-\mu)^2}{2\sigma^2}} \tag{6.4}$$

Similar to problem 5, we will now take the log of this likelihood, and notice the product operator will distribute an "n" to the first term, and sum across X_i in the exponent of "e". This will lead to:

$$LL(\theta) = -n \cdot \log\left(2\pi\sqrt{\sigma^2\sigma_0^2}\right) + \sum_{i=1}^{n} \frac{-(X_i-\theta)^2}{2\sigma_0^2} - \frac{(\theta-\mu)^2}{2\sigma^2} \tag{6.5}$$

Now we differentiate with respect to θ:

$$\frac{dLL(\theta)}{d\theta} = \sum_{i=1}^{n} \frac{-(X_i-\theta)}{\sigma_0^2} - \frac{(\theta-\mu)}{\sigma^2} \tag{6.6}$$

Similar to above, we notice this summation adds an "n" factor to both the first and second term, simplifying to:

$$\frac{dLL(\theta)}{d\theta} = \frac{n\bar{X} - n\theta}{\sigma_0^2} + \frac{n\theta - n\mu}{\sigma^2} \tag{6.7}$$

where the $n\sum_{i=1}^{n} X_i$ term simplifies to the sample mean, $\bar{X}$. Equating this to 0 and solving for θ_{MAP} leads to our answer:

$$\theta_{MAP} = \frac{\mu\sigma_0^2 - \bar{X}\sigma^2}{\sigma_0^2 - \sigma^2}$$

b.) Similar to Exercise 5, we will solve for the Bayes estimate using the equation

$$p(f \mid D) = \frac{p(D \mid f) \cdot p(f)}{\int_0^\infty p(D \mid f) \cdot p(f) df} \tag{6.8}$$

When plugging in equations 6.2 and 6.3, this yields

$$p(\theta \mid X, \sigma, \sigma_0, \mu) = \frac{\frac{1}{2\pi\sqrt{\sigma^2\sigma_0^2}} \cdot e^{\frac{-(X_i-\theta)^2}{2\sigma_0^2} - \frac{(\theta-\mu)^2}{2\sigma^2}}}{\int_0^\infty \frac{1}{2\pi\sqrt{\sigma^2\sigma_0^2}} \cdot e^{\frac{-(X_i-\theta)^2}{2\sigma_0^2} - \frac{(\theta-\mu)^2}{2\sigma^2}} d\theta} \tag{6.9}$$

At this stage, we can look at the bottom term being integrated and notice that when derived with respect to θ and solved for from 0 to infinity, it will become a function of σ, σ_0, $\sum_{i=1}^{n} X_i$, and μ. This essentially makes it a constant that does not relate to theta. Thus, when we take the log of this likelihood equation, it will disappear just the same as a constant, leaving just the numerator. Maximizing the numerator is the same as we did in the above problem, finding θ_{MAP}, thus the Bayes estimate will be the same as the MAP estimate:

$$\theta_{Bayes} = \frac{\mu\sigma_0^2 - \bar{X}\sigma^2}{\sigma_0^2 - \sigma^2}$$

Exercise 7

a.) For the case where we only use D_x, the parameters can be found using the following logic. The two main parts of the logic defined in the EM algorithm and the E-step and the M-step. The combination of these two steps allow us to converge to parameter estimations. The first step, or the E-step, involves calculating the pseudo-posteriors. Denoted $\gamma_{Z_i=k}$, these can be calculated using the following equation:

$$\gamma_{Z_i=k} = P(Z_i = k \mid X_i) = \frac{P(X_i \mid Z_i = k) \cdot P(Z_i = k)}{P(X_i)} = \frac{w_k \cdot N(x_i, \mu_k, \sigma_k)}{\sum_c w_c \cdot N(x_c, \mu_c, \sigma_c)} \tag{7.1}$$

where k denotes the number of distributions involved in the problem, N is the normal distribution, w denotes the weight on each distribution, x_i denotes each data point in the data set, and μ_c and σ_c are the updated estimated parameters of the normal distribution. The first time this is run, these parameters are initialized to some value.

The second step, or the M-step, involves calculating updated parameters based off of the pseudo-parameters found in the E-step. Specifically, we are interested in finding updates in the parameters μ, σ, and w. The equations for these parameters can be derived from the auxiliary function:

$$Q(\theta, \theta^{(t)}) = E[log(P(Z \mid \theta), \theta^{(t)})] \tag{7.2}$$

where θ is defined as the set of parameters we are estimating. In the M-step, our goal is to maximize this function to find better parameters. Maximizing and setting derivatives = 0, we get:

$$Q(\theta, \theta^{(t)}) = \sum_{k=1}^{M} \sum_{i=1}^{N} log\gamma_k P(Z_{ki}, \theta^{(t)}) + \sum_{k=1}^{M} \sum_{i=1}^{N} logP(x_i \mid \theta_k) \cdot P(Z_{ki}, \theta^{(t)}) \tag{7.3}$$

Maximizing this function with respect to each parameter we want to find leads to the equations for each parameter update:

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \gamma_{Z_i=k} \cdot X_i}{N_k} \tag{7.4}$$

$$\hat{\sigma}_k^2 = \frac{\sum_{i=1}^{N} \gamma_{Z_i=k} \cdot (X_i - \mu_k)^2}{N_k} \tag{7.5}$$

$$\hat{w}_k = \frac{N_k}{N} \tag{7.6}$$

where

$$N_k = \sum_{i=1}^{N} \gamma_{Z_i=k}$$

Once we calculate these updated parameters, we pass them back in to the E-step and repeat until we reach some terminating condition. This terminating condition could be convergence within a certain error tolerance, or simply a max number of iterations. These equations are derived from the laws of probability relating to the auxiliary function and log likeliness.

b.) When incorporating D_Y and *beta*, equation 7.3 changes slightly to:

$$Q(\theta,\theta^{(t)}) = \sum_{k=1}^{M} log(\alpha \cdot N(x_k \mid \mu_1,\sigma_1^2) + (1-\alpha)\cdot N(x_k \mid \mu_2,\sigma_2^2)) + \sum_{j=1}^{N} log(\beta \cdot N(y_j \mid \mu_1,\sigma_1^2) + (1-\beta)\cdot N(y_j \mid \mu_2,\sigma_2^2)) \tag{7.7}$$

We can now use Lagrange multipliers and the relationship in equation 7.1 to find α and β. Similar to equation 7.6, this results in:

$$\alpha = \frac{M_k}{M} \tag{7.8}$$

$$\alpha = \frac{N_k}{N} \tag{7.9}$$

where...

$$M_k = \sum_{i=1}^{m} \frac{w_k \cdot N(x_i,\mu_k,\sigma_k)}{\sum_c w_c \cdot N(x_c,\mu_c,\sigma_c)} \tag{7.10}$$

$$N_k = \sum_{i=1}^{n} \frac{w_k \cdot N(y_i,\mu_k,\sigma_k)}{\sum_c w_c \cdot N(y_c,\mu_c,\sigma_c)} \tag{7.11}$$

Maximizing these and setting derivatives of desired parameters =0 leads to our equations for parameter updates:

$$\hat{\mu}_k = \frac{1}{M_k+N_k} \cdot \left(\sum_{i=1}^{M} x_i \cdot \left(\frac{w_k \cdot N(x_i,\mu_k,\sigma_k)}{\sum_c w_c \cdot N(x_c,\mu_c,\sigma_c)}\right) + \sum_{j=1}^{N} y_j \cdot \left(\frac{w_k \cdot N(y_j,\mu_k,\sigma_k)}{\sum_c w_c \cdot N(y_c,\mu_c,\sigma_c)}\right)\right. \tag{7.12}$$

$$\hat{\sigma}_k^2 = \frac{1}{M_k+N_k} \cdot \left(\sum_{i=1}^{M} (x_i-\mu_k)^2 \cdot \left(\frac{w_k \cdot N(x_i,\mu_k,\sigma_k)}{\sum_c w_c \cdot N(x_c,\mu_c,\sigma_c)}\right) + \sum_{j=1}^{N} (y_j-\mu_k)^2 \cdot \left(\frac{w_k \cdot N(y_j,\mu_k,\sigma_k)}{\sum_c w_c \cdot N(y_c,\mu_c,\sigma_c)}\right)\right. \tag{7.13}$$

c.) The source code I created to simulate the EM algorithm for parts a and b is attached below:

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

class EM():
    def __init__(self, mn, mu1_actual, mu2_actual, sig1_actual, sig2_actu
        """
        Pass in actual parameter values that the parameter estimates shou
        """
        self.mn = mn
        self.mu1_actual = mu1_actual
        self.mu2_actual = mu2_actual
        self.sig1_actual = sig1_actual
        self.sig2_actual = sig2_actual
```

```
        self.alpha_actual = alpha_actual
        self.beta_actual = beta_actual
        self.LL = np.empty((0,1))

    def initialize(self, mu1_init, mu2_init, sig1_init, sig2_init, alpha_
        """
        Initialize parameters to initial parameter estimates
        """
        self.mu1_est = mu1_init
        self.mu2_est = mu2_init
        self.sig1_est = sig1_init
        self.sig2_est = sig2_init
        self.alpha_est = alpha_init
        self.beta_est = beta_init

    def PD_d(self, d, ab, k) -> float:
        """
        Args:
            d: float, x or y value
            ab: string; "alpha" or "beta"
            k: int; 1 or 2 (which mixture)
        Returns:
            float; value of pD(d) for the given data point
        """
        if ab == "alpha":
            if k == 1:
                return self.alpha_est*(math.exp((-(d-self.mu1_est)**2)/(2
            if k == 2:
                return (1-self.alpha_est)*(math.exp((-(d-self.mu2_est)**2
        elif ab == "beta":
            if k == 1:
                return self.beta_est*(math.exp((-(d-self.mu1_est)**2)/(2*
            if k == 2:
                return (1-self.beta_est)*(math.exp((-(d-self.mu2_est)**2)

    def generateNormal(self, num, mode):
        """
        Args:
            num: int; 1 or 2
            mode: string; "est" or "actual"
        Returns:
            np.array; normal distribution based on given inputs
        """
        if num == 1:
            if mode == "est":
                return np.random.normal(self.mu1_est, self.sig1_est, self
            elif mode == "actual":
                return np.random.normal(self.mu1_actual, self.sig1_actual
```

```
        elif num == 2:
            if mode == "est":
                return np.random.normal(self.mu2_est, self.sig2_est, self
            elif mode == "actual":
                return np.random.normal(self.mu2_actual, self.sig2_actual

    def generateData(self, xy):
        """
        Generates initial data based on actual values
        Args:
            xy: string; "x" or "y"
        Returns:
            np.array: x or y data based on input
        """
        if xy == 'x':
            x_dist_1 = self.alpha_actual*self.generateNormal(1, "actual")
            x_dist_2 = (1-self.alpha_actual)*self.generateNormal(2, "actu
            self.x_data = np.concatenate((x_dist_1, x_dist_2)).flatten()
        elif xy == 'y':
            y_dist_1 = self.beta_actual*self.generateNormal(1, "actual")
            y_dist_2 = (1-self.beta_actual)*self.generateNormal(2, "actua
            self.y_data = np.concatenate((y_dist_1, y_dist_2)).flatten()

    def E_step(self):
        # Initialize arrays to hold pseudo posteriors
        self.og_gammas = np.empty((len(self.x_data),2))
        self.gammas = np.empty((len(self.x_data),2))
        for i in range(len(self.x_data)):
            for mode in [1, 2]:
                self.og_gammas[i, mode-1] = em.PD_d(self.x_data[i], "alph
        self.gammas[:,0] = self.og_gammas[:,0]/np.sum(self.og_gammas, axi
        self.gammas[:,1] = self.og_gammas[:,1]/np.sum(self.og_gammas, axi

    def M_step(self):
        N_1, N_2 = np.sum(self.gammas, axis=0)
        self.mu1_est = (1/N_1)*np.sum(np.multiply(self.gammas[:,0], self.
        self.mu2_est = (1/N_2)*np.sum(np.multiply(self.gammas[:,1], self.
        self.sig1_est = math.sqrt((1/N_1)*np.sum((self.gammas[:,0])*np.sq
        self.sig2_est = math.sqrt((1/N_2)*np.sum((self.gammas[:,1])*np.sq
        self.alpha_est = N_1/self.mn
        self.beta_est = N_2/self.mn

    def computeLogLikelihood(self):
        self.LL = np.append(self.LL, np.sum(np.log(np.sum(self.og_gammas,



num_tests = 1000
```

```
max_steps = 50
results = np.empty((0,6))
results_main = np.empty((0,6))

for i in range(num_tests):
    em = EM(1000, 10, 12, 2, 0.5, 0.5, 0.7)
    em.initialize(1, 1, 1, 1, 0.1, 0.6)
    em.generateData('x')
    em.generateData('y')

    for step in range(max_steps):
        em.E_step()
        em.M_step()
        results = np.append(results, np.array([[em.mu1_est, em.mu2_est, e
        em.computeLogLikelihood()
    results_main = np.append(results_main, np.array([[em.mu1_est, em.mu2_
    print(F"[INFO]: {i+1} / {num_tests} complete.")

# Post process data
averages = np.mean(results_main, axis=0)
stddevs = np.std(results_main, axis=0)

print(averages)
print(stddevs)

# Plots
x_range = range(max_steps)
est_labels = ["mu1 est", "mu2 est", "sigma1 est", "sigma2 est", "alpha es
actual = [em.mu1_actual, em.mu2_actual, em.sig1_actual, em.sig2_actual, e
actual_labels = ["mu1 actual", "mu2 actual", "sigma1 actual", "sigma2 act
colors = ["r", "b", "g", "c", "m"]
for i in range(5):
    plt.plot(x_range, [actual[i]]*len(x_range), colors[i], label = actual
    plt.plot(x_range, results[:,i], colors[i], label = est_labels[i])
plt.plot(x_range, em.LL, label = "Log Likelihood")
plt.legend()
plt.show()
```

To quickly explain this code, first, for every test (1000 total) an EM class is created, with actual parameter values that are going to be found during the E step and M step. Next, initial parameter values are created for the data distribution parameters, using the initialize() function. Then X and Y data are created based on the actual parameters used to create the instance of the class, using the generateData() method. Next is the main section of the EM algorithm. I used my terminal state to be a maximum numper of steps, or iterations. Then, for each iteration, The E and M steps are executed, passing in the old parameters and creating the new parameters using the equations from equations 7.{1, 4, 5, 6, 8, 9, 12, 13}. I then save the results for each test to be later post processed in to averages and standard deviations. The last section is a plotting function I used to visualize some results.

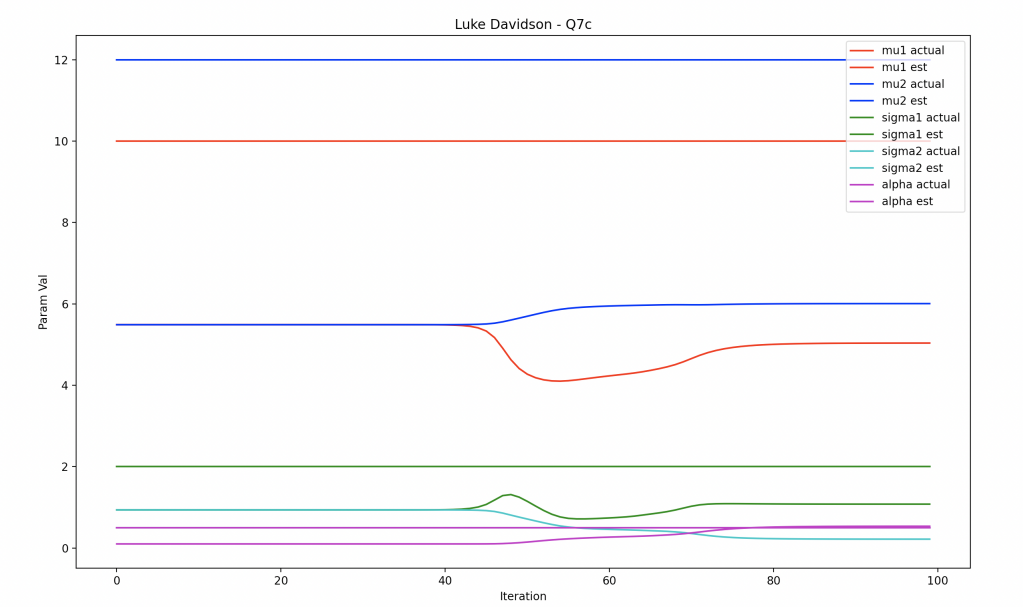


Figure 9: Parameter Estimates with iterations = 100

I received some interesting results. Attached is a graph of a result of one of my simulations:

We can see in figures 9 and 10 that the parameter values do seem to converge to a value, although not the correct anticipated value. The only parameter to converge correctly was the alpha value. This could be due to computational error. This is anticipated to be better in part b due to the use of multiple data sets and parameters used.

In figures 11 and 12, we can see that the log likelihood also converges to a value, which is anticipated to happen once the parameters are near their estimation.

For 100 tests, some of the mean values I calculated are seen in figure 13 below.

The first list printed out represents means, in the order $[\mu_1, \mu_2, \sigma_1, \sigma_2, \alpha, \beta]$. The second list represents the standard deviations. Similar to the results seen above for a single test, the value that seemed to converge the best is alpha, converging to 0.493, when the true value was 0.5.

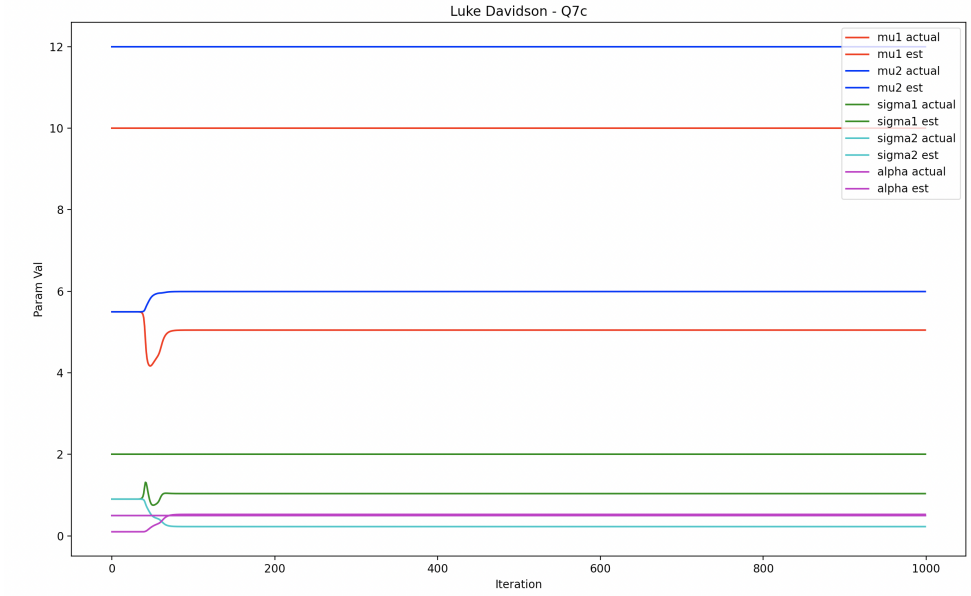


Figure 10: Parameter Estimates with iterations = 1000

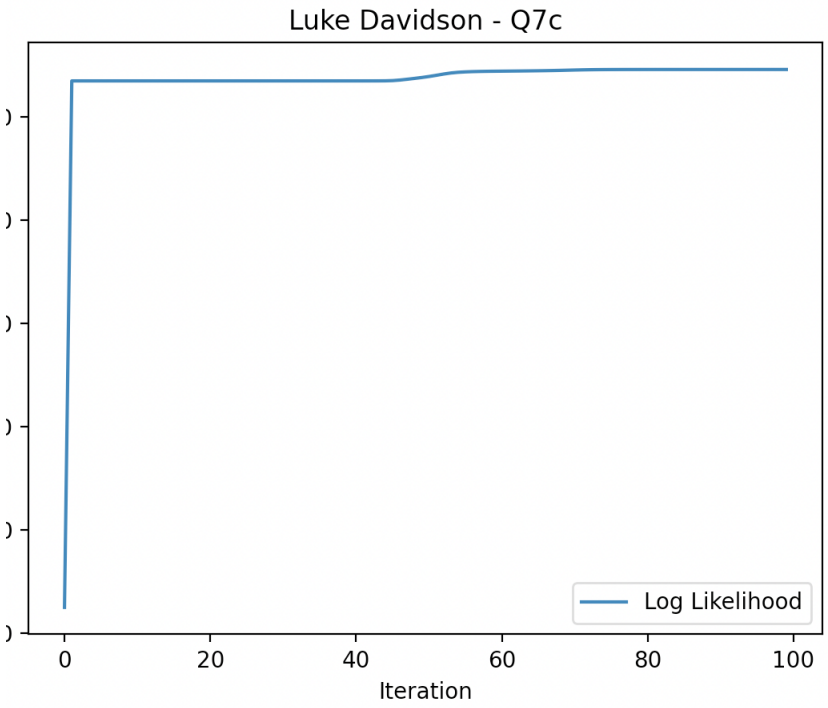


Figure 11: Log Likelihood with iterations = 100

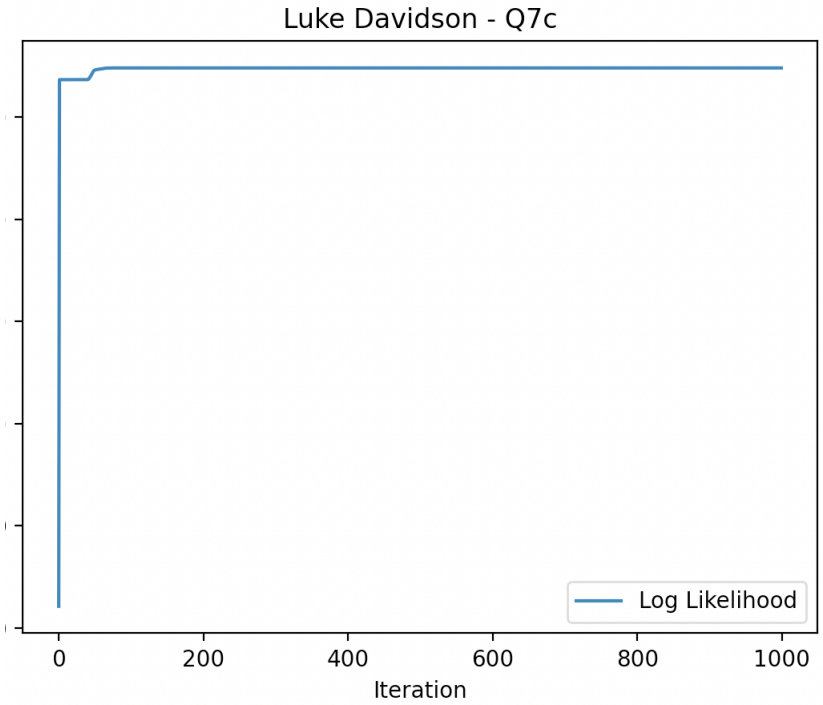


Figure 12: Log Likelihood with iterations = 1000

```
[INFO]: 999 / 1000 complete.
[INFO]: 1000 / 1000 complete.
[4.98201725 5.99950102 0.99342162 0.25404417 0.49287521 0.50712479]
[0.08935558 0.01481887 0.0486298  0.02421763 0.03104167 0.03104167]
```

Figure 13: Means and Standard Deviations for m and n = 1000, and 1000 tests

Exercise 8

a.) The volume of a hypercube and hypersphere are given as:

$$V_{hypercube}(n) = r^n \tag{8.1}$$

$$V_{hypersphere}(n) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} \cdot r^n \tag{8.2}$$

The ratio of a hypersphere with radius r and hypercube with side $2r$ is given as:

$$\frac{\frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} \cdot r^n}{(2r)^n} \tag{8.3}$$

The r^n terms cancel, leaving

$$\frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1) \cdot 2^n} \tag{8.4}$$

The below image shows the values of this ratio as n increases, via WolframAlpha:

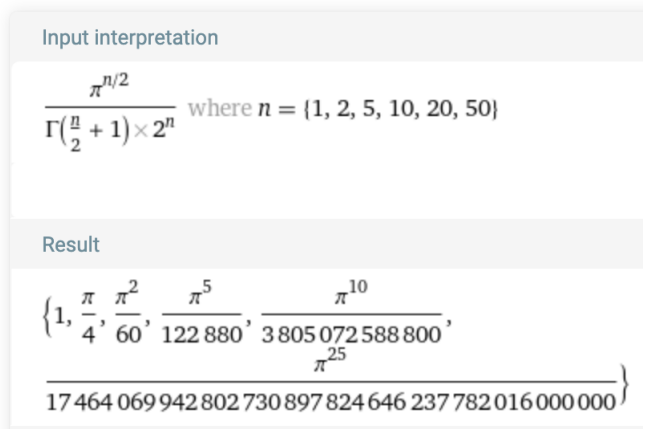


Figure 14: Value of Equation 8.4 as n increases

As one can see, as n increases, representing high dimension spaces, the ratio approaches 0. This represents that the volume of the hypercube is much greater than the volume of the hypersphere. Now computing the "ratio of the distance from the center of the hypercube to one of the corners divided by the distance to one of the sides", we get:

$$\frac{\sqrt{n \cdot r^2}}{r} = \sqrt{n} \tag{8.5}$$

Clearly, as n increases, so will this ratio, representing that the distance from the center to a corner will increase faster than the distance to one of the sides. Both of these together show that the corners will become "spikes" due to the long shapes with little volume.

b.) "The fraction of the volume of the sphere which lies at the distance between $r\varepsilon$ and r from the center, where $0 < \varepsilon < r$", is:

$$\frac{V(r) - V(r-\varepsilon)}{V(r)} = \frac{r^d - (r-\varepsilon)^d}{r^d} \tag{8.6}$$

Evaluated at $\varepsilon = \{0.01r, 0.5r\}$ and $r = \{1, 2, 3, 10, 100\}$ leads to:

		epsilon	
		0.01r	0.5r
r	1	0.010000	0.500000
	2	0.019900	0.750000
	3	0.029701	0.875000
	10	0.095618	0.999023
	100	0.633968	1.000000

Figure 15: Equation 8.6 evaluated at the given values

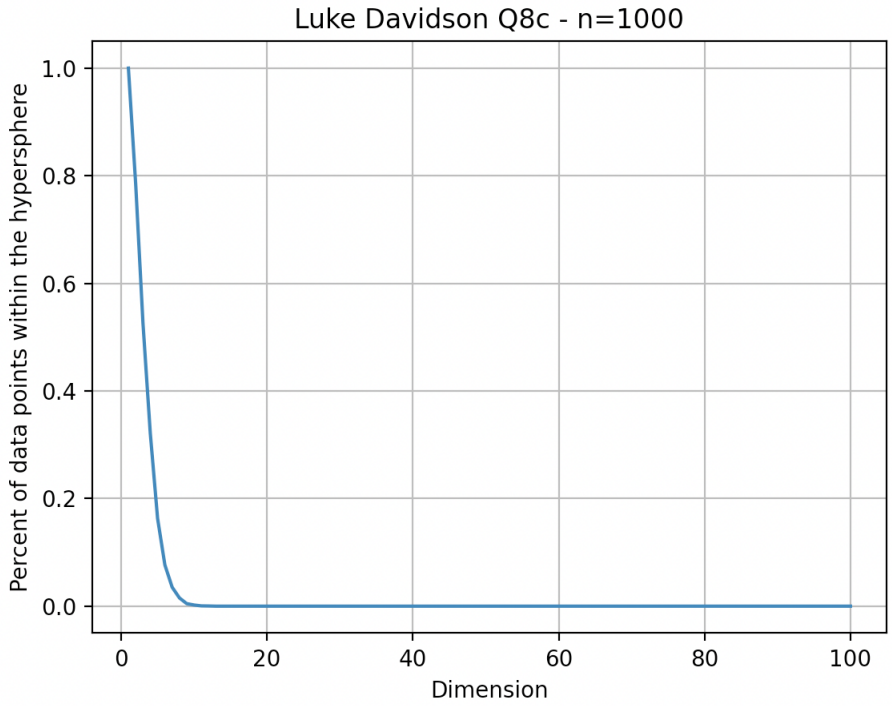
This shows that as r increases, the percentage of points within the small shell increases, and in fact approaches 1, meaning they are concentrating in that small shell.

c.) Code for simulating the percentage of random points that lie within hypersphere or radius r, within a hyper cube with length 2r:

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

n = 10
d = range(1, 101)
percents = []
percents_master = []

def GenerateDataPoints(n, d):
    data = np.random.rand(n, d)
    return data

def PercentInHypersphere(d):
    percent = math.pi**(d/2)/(math.gamma(((d/2)+1))*2**d)
    return percent

for dim in d:
    percent = PercentInHypersphere(dim)
    data = GenerateDataPoints(n, dim)
    percents = []
    for points in range(n):
        count = 0
        for pt in data[points, :]:
            if pt < percent:
                count += 1
        percents.append(count/dim)
    print(percents)
```

```
        percents_master.append(sum(percents)/n)

plt.plot(d, percents_master)
plt.grid()
plt.xlabel("Dimension")
plt.ylabel("Percent of data points within the hypersphere")
plt.title(f"Luke Davidson Q8c - n={n}")
plt.show()
```

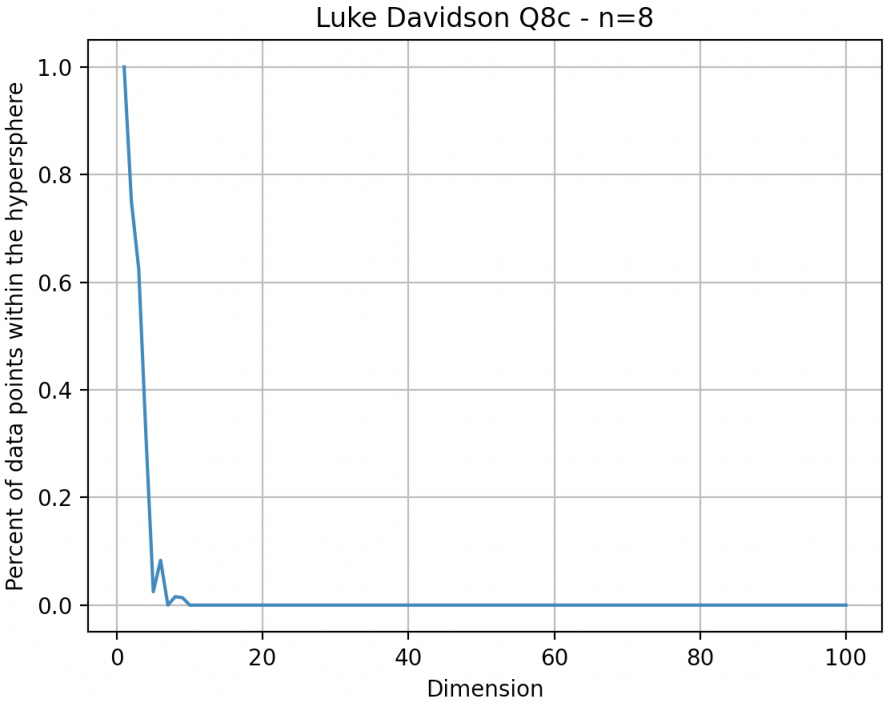
This code results in the following plot for n = 1000:



As one can see, this proves our answer in part a. As the dimension increases, the percentage of random points within a hypercube of side 2r that also lie within a hypersphere of radius r quickly decreases. Nearing 0% at just around a dimension of 10. This graph was recreated using "n" data points that are "d" dimensions. The data created is thus an nxd matrix. Each of these data points was tested to see if it lied within the hypersphere, and then an average percentage was taken for each dimension from 1 to 100.

If we use a small "n" value, for example 8, the resulting graph looks like below.

Using a smaller "n" value greatly increases the variability in the results, as seen in the inconsistency in the graph near dimensions 5-12. A varied "r" will lead to similar results, as the volume percentage of a hypersphere and hypercube do not depend on "r", as seen in equation 8.4.


Luke Davidson Q8c - n=8
Percent of data points within the hypersphere
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Dimension