COMP0080 - Graphical Models
Assignment 1
DUE: NOVEMBER 11, 2019

UCL
Machine Learning MSc
Computational Statistics and Machine Learning MSc

Author: Dorota Jagnesakova
Student ID: 16079098
E-mail: dorota.jagnesakova.19@ucl.ac.uk
Questions: 2,3

Author: Agnieszka Dobrowolska
Student ID: 16034489
E-mail: agnieszka.dobrowolska.16@ucl.ac.uk
Questions: 2,3

Author: Oliver Slumbers
Student ID: 19027699
E-mail: oliver.slumbers.19@ucl.ac.uk
Questions: 1,4

Author: Tom Grigg
Student ID: 19151291
E-mail: tom.grigg.19@ucl.ac.uk
Questions: 1,4

1 Three Cards

We first start by defining the sample space of possible values for the upper and lower faces of the drawn card, along with their respective probabilities:

$$P(Upper, Lower) = \begin{cases} \frac{1}{3} & \text{(Black, Black)} \\ \frac{1}{6} & \text{(Black, White)} \\ \frac{1}{6} & \text{(White, Black)} \\ \frac{1}{3} & \text{(White, White)} \end{cases}$$

Which can perhaps be summarised more clearly in a joint probability distribution table:

		Lower Face	
		Black	White
Upper Face	Black	$\frac{1}{3}$	$\frac{1}{6}$
	White	$\frac{1}{6}$	$\frac{1}{3}$

It is clear that if we draw a card and the upper face is black we have two possible cases to consider, either we are in (Black, Black), or (Black, White). In the following, denote BB as (Black, Black), BW as (Black, White), etc.

We can apply Bayes' rule:
Case 1: The (Black, Black) card has been drawn

$$\begin{aligned} P(BB|U = Black) &= \frac{P(U = Black|BB)P(BB)}{P(U = Black)} \\ &= \frac{(1) * \frac{1}{3}}{\frac{1}{2}} \\ &= \frac{\frac{1}{3}}{\frac{1}{2}} \\ &= \frac{2}{3} \end{aligned}$$

Case 2: The (Black, White) card has been drawn

$$\begin{aligned} P(BW|U = Black) &= \frac{P(U = Black|BW)P(BW)}{P(U = Black)} \\ &= \frac{(1) * \frac{1}{6}}{\frac{1}{2}} \\ &= \frac{\frac{1}{6}}{\frac{1}{2}} \\ &= \frac{1}{3} \end{aligned}$$

Therefore, whilst it is not possible to ascertain the colour of the lower face given only that the upper face is black, this knowledge does allow you to infer more about which card you have actually drawn. You have either drawn the all-black card, or the black-white card, and the former event is twice as likely as the latter.

2 Earthquake

(a) Exercise 1.22 Part 1.

This problem aims to infer the locations of 2 explosions using the values of energy waves, v_i, detected at each of the N sensors. We define the following assumptions:

- Assume earth is 2-dimensional.
- There are N sensors spread out evenly on the surface of the earth, located at (x_i, y_i) where $i = 1, 2, ..., N$
- Priors of the explosions are distributed uniformly, according to the spiral coordinate system, and independent.
- 2 explosions occur at unknown locations s_1, s_2.
- v_i are the noisy values observed at the i^{th} sensor, where $v_i = \frac{1}{d_i^2(1)+0.1} + \frac{1}{d_i^2(2)+0.1} + \sigma\epsilon_i$ and $d_i^2(1), d_i^2(2)$ are square distances from the explosions 1 and 2 to the i^th sensor.
- σ is the standard deviation of the Gaussian sensor noise
- $\epsilon_i \overset{iid}{\sim} N(0, 1)$.

First we expand the joint distribution:

$$\begin{aligned} P(s_1, s_2, \boldsymbol{v}) &= P(\boldsymbol{v}|s_1, s_2)P(s_1, s_2) \\ &= P(\boldsymbol{v}|s_1, s_2)P(s_1)P(s_2) \text{ by independence.} \end{aligned}$$

Then, to obtain the distribution conditional on $\boldsymbol{v}$:

$$P(s_1, s_2, \boldsymbol{v}) = P(s_1, s_2|\boldsymbol{v})P(\boldsymbol{v}) = P(\boldsymbol{v}|s_1, s_2)P(s_1)P(s_2)$$
$$P(s_1, s_2|\boldsymbol{v}) = P(\boldsymbol{v}|s_1, s_2)\frac{P(s_1)P(s_2)}{P(\boldsymbol{v})}$$

Assuming the observed sensor values are independent of each other, given explosion location:

$$P(s_1, s_2|\boldsymbol{v}) = \frac{P(s_1)P(s_2)}{P(\boldsymbol{v})}\prod_{i=1}^{N} P(v_i|d_i)$$
$$\text{with } P(v_i|d_i) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{1}{2\sigma^2}(v_i - \frac{1}{d_i^2(1)+0.1} - \frac{1}{d_i^2(2)+0.1})}$$

$$P(s_1, s_2|\boldsymbol{v}) \propto \prod_{i=1}^{N} P(v_i|d_i)$$

To obtain the posterior for just s_1 from our joint posterior, we sum over all the values of s_2. This is performed numerically in the code to obtain a probability for each s_1.

$$P(s_1|\boldsymbol{v}) = \sum_{s_2} P(s_1, s_2|\boldsymbol{v}) \tag{1}$$

To calculate the most likely location, we compute $\arg\max_{s_1} P(s_1|v)$. The resulting value for s_1 is plotted in Figure 1.

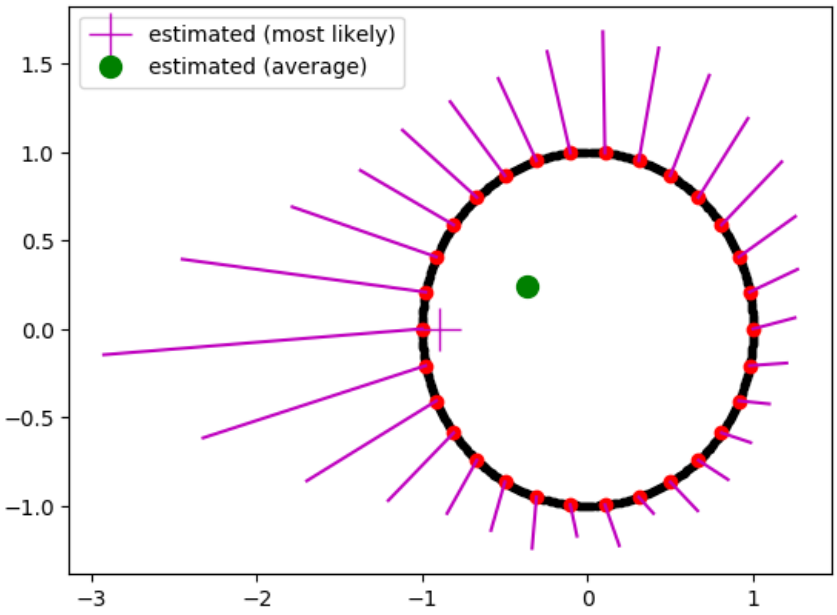

Figure 1: Posterior $P(s_1|\boldsymbol{v})$, where the highest probability is associated with the point lying at $(-0.9, 0)$

Part 2.

Under hypothesis $\mathcal{H}_1$, we assume there is only one explosion. Hence, our model takes the same form as in the one-explosion case in Example 1.12 of the textbook. In particular, $\mathcal{H}_1$ assumes we have a corresponding one-explosion (noisy) function for our observations.

$$v_i = \frac{1}{d_i^2 + 0.1} + \sigma\epsilon_i$$

Calculating $P(v|\mathcal{H}_1)$ for our observed data values v was performed via computing the following expression:

$$\begin{aligned} P(v|\mathcal{H}_1) &= \sum_{s \in S} P(v_i|s)P(s) \\ &= P(s_0) \sum_{s \in S} \prod_{i=1}^{n} P(v_i|d_i) \end{aligned}$$

Where $P(s_0)$ is the uniform value of the probability distribution over possible explosion points s, and where the distribution $P(v_i|d_i)$ takes the form:

$$P(v_i|d_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(v_i - \frac{1}{d_i^2+0.1})}$$

$P(v|\mathcal{H}_2)$ was computed similarly:

$$\begin{aligned} P(v|\mathcal{H}_2) &= \sum_{s_1, s_2 \in S^2} P(v_i|s_1, s_2)P(s_1)P(s_2) \\ &= P(s_0)^2 \sum_{s_1, s_2 \in S^2} \prod_{i=1}^{n} P(v_i|d_i(1), d_i(2)) \end{aligned}$$

where

$$P(v_i|d_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(v_i - \frac{1}{d_i^2(1)+0.1} - \frac{1}{d_i^2(2)+0.1})}, \qquad v_i = \frac{1}{d_i^2(1) + 0.1} - \frac{1}{d_i^2(2) + 0.1} + \sigma\epsilon_i$$

The value of $\log(P(v|\mathcal{H}_2) - \log P(v|\mathcal{H}_1) \approx 8.696$ - suggesting that H_2 is magnitudes more likely than H_1 (see part 3).

Part 3.

Recall Bayes' rule, and apply it to $\mathcal{H}_1$ and $\mathcal{H}_2$:

$$P(\mathcal{H}_1|\boldsymbol{v}) = \frac{P(\boldsymbol{v}|\mathcal{H}_1)P(\mathcal{H}_1)}{P(\boldsymbol{v})}$$

$$P(\mathcal{H}_2|\boldsymbol{v}) = \frac{P(\boldsymbol{v}|\mathcal{H}_2)P(\mathcal{H}_2)}{P(\boldsymbol{v})}$$

Assuming that we have no prior preference, that is $p(\mathcal{H}_1) = p(\mathcal{H}_2)$, we can say:

$$\frac{P(\mathcal{H}_2|\boldsymbol{v})}{P(\mathcal{H}_1|\boldsymbol{v})} = \frac{P(\boldsymbol{v}|\mathcal{H}_2)}{P(\boldsymbol{v}|\mathcal{H}_1)}$$

By taking logs of both sides, we can see that the difference between logs corresponds to the ratio of probabilities (i.e. odds) between $\mathcal{H}_1$ and $\mathcal{H}_2$ given our evidence $\boldsymbol{v}$:

$$\log\left(\frac{P(\mathcal{H}_2|\boldsymbol{v})}{P(\mathcal{H}_1|\boldsymbol{v})}\right) = \log\left(\frac{P(\boldsymbol{v}|\mathcal{H}_2)}{P(\boldsymbol{v}|\mathcal{H}_1)}\right) = \log(P(\boldsymbol{v}|\mathcal{H}_2)) - \log(P(\boldsymbol{v}|\mathcal{H}_1))$$

This is a useful quantity since it increases and decreases in a strictly monotonic way with the ratio between $P(\mathcal{H}_2|\boldsymbol{v})$ and $P(\mathcal{H}_1|\boldsymbol{v})$, whilst also being sensitive to numerically small probabilities that may otherwise be difficult to handle computationally. Hence, the quantity is a useful way to compare the hypothesis that there are 2 explosions versus only 1 explosion. In fact, it can roughly be interpreted as the number of orders of magnitude by which H_2 is more likely than H_1 (and vice versa if the value is negative).

Part 4.

The computational complexity of calculating $\log(P(v|\mathcal{H}_2)$ comes from the fact that in our derivations we have to sum over all possible positions of the explosions. If there are k explosions, then there are S^k possible permutations that we have to evaluate the probability mass function over. This creates two issues: the first being that the algorithm for computing the density is necessarily $O(S^k)$ - meaning our algorithm will take exponentially more time to run as we increase the number of explosions to include in our hypothesis. The second issue is that as the number of permutations increases, the probability measure assigned to each point in S^k will also be forced to decrease exponentially on average - meaning we will soon run into numerical errors and for sufficiently large k, will be forced to use much more computationally expensive algorithms/representations for our numbers and numerical calculations.

(b) Exercise 1.23 Part 1.

Using the same inference scheme as derived in (a) Part 1., we infer the posterior distribution of $P(s_1|\boldsymbol{v})$ for a different kind of explosion sensor. This sensor measures the mean of two incoming explosions, where observed datapoints are modelled as:

$$v_i = \frac{0.5}{d_i^2(1) + 0.1} + \frac{0.5}{d_i^2(2) + 0.1} + \sigma\epsilon_i \tag{2}$$

The resulting posterior $P(s_1|\boldsymbol{v})$ is shown in Figure 2, where the highest probability is associated with the point lying at $(0.433595, -0.068747)$.

Part 2.

Under $\mathcal{H}_1$, our model takes the same form as in (a) Part 2. (Example 1.12 in the textbook). This is

$$P(v|\mathcal{H}_1) = P(s_0)\sum_{s\in S}\prod_{i=1}^{n} P(v_i|d_i)$$

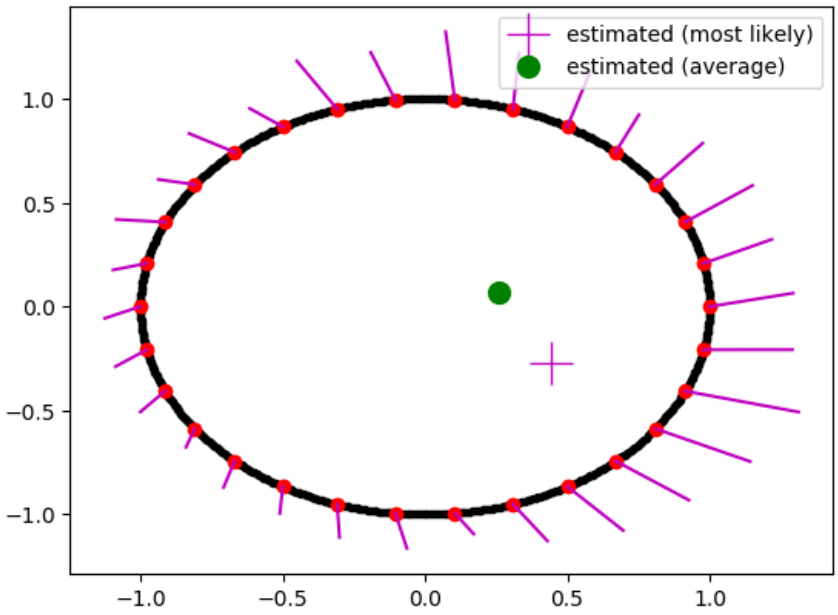

Figure 2: Posterior $P(s_1|\boldsymbol{v})$ for the mean data sensor, where the highest probability is associated with the point lying at $(0.440388, -0.269871)$

with $v_i = \frac{1}{d_i^2+0.1} + \sigma\epsilon_i$, $P(s_0)$ equal the uniform value of the probability distribution over possible explosion points s, and:

$$P(v_i|d_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(v_i - \frac{1}{d_i^2+0.1})}$$

Under $\mathcal{H}_2$, we assume there are two explosions and the noisy observations measured at each sensor i measure the mean of those explosions. The derivation of $P(v|\mathcal{H}_2)$ will therefore be the same as in (a) Part 2., with the exception of this equation for the observed values v_i. This is

$$P(v|\mathcal{H}_2) = P(s_0)^2 \sum_{s_1,s_2 \in S^2} \prod_{i=1}^{n} P(v_i|d_i(1), d_i(2))$$

with

$$P(v_i|d_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(v_i - \frac{0.5}{d_i^2(1)+0.1} - \frac{0.5}{d_i^2(2)+0.1})}, \qquad v_i = \frac{0.5}{d_i^2(1)+0.1} - \frac{0.5}{d_i^2(2)+0.1} + \sigma\epsilon_i$$

The value of $\log(P(v|\mathcal{H}_2) - \log P(v|\mathcal{H}_1) \approx 7.8217$.

Part 3.

The ratio of probabilities of $\mathcal{H}_2$ versus $\mathcal{H}_1$ (that is 2 explosions occurred as opposed to 1 explosion given our data v) is lower for the sensors measuring the mean of the explosions than it is for the sensors in (a). Intuitively, using sensors that measure the mean of the incoming explosions rather than their sum results in smaller values of the observations relative to the $\sigma\epsilon_i$ error term. More explicitly:

We can see why w_i will be less informative by plotting the distributions of the observed values:
Formally, the coefficient of variation $(\frac{\sigma}{\mu})$ is twice as high for the mean data - resulting in a less informative distribution over a smaller range of values.

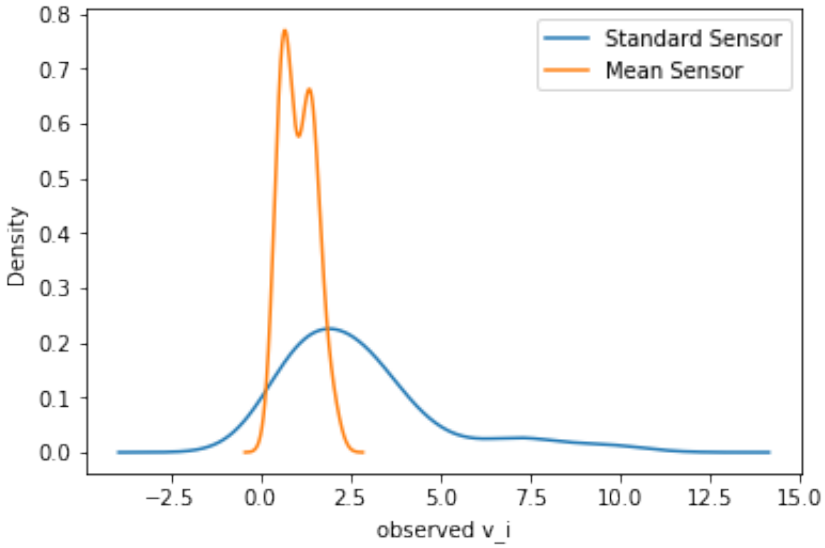

Figure 3: Kernel Density Estimates for the observed data values given different sensor types - notice that the distribution for mean sensor is much thinner.

3 Meeting Scheduling

(a) In this part of the problem, we will determine the latest possible meeting time that will give us a probability of no less than 0.9 of catching the train. We know everyone needs to arrive at our meeting point by 21:05. We will calculate the minimum number of extra minutes $d_N \in (5n \mid n \in \mathbb{N}_0)$ necessary to subtract from 21:05 in order to catch the train with probability 0.9.

We are given a set of independent identically distributed (iid) non-negative random variables $D_i, i = 1, .., N$ representing the delay of each of our N friends respectively. We only work with multiples of five, i.e. $D_i \in (5n \mid n \in \mathbb{N}_0)$. From the information given to us, we can calculate the probability distribution function for these random variables:

$$F_D(d) = P(D \leqslant d) = \begin{cases} 0.7 & \text{if } d = 0 \\ 0.8 & \text{if } d = 5 \\ 0.9 & \text{if } d = 10 \\ 0.97 & \text{if } d = 15 \\ 0.99 & \text{if } d = 20 \\ 1 & \text{if } d > 20 \end{cases}, d \in S$$

with $S = (5n \mid n \in \mathbb{N}_0)$. We want to find a minimum $d \in S$ such that $\prod_{i=1}^{N} P(D_i \leqslant d) \geqslant 0.9$. Since our random variables are iid, we can write this as $d_N = min\{d \in S \mid P(D_i \leqslant d)^N \geqslant 0.9\}$.

- For N=3, we want $d_3 = min\{d \in S \mid P(D_i \leqslant d)^3 \geqslant 0.9\}$. This is $d_3 = 15$, i.e. our meeting time should be 20:50. Then we will catch our train with probability $0.97^3 \approx 0.9127$
- For N=5, we want $d_5 = min\{d \in S \mid P(D_i \leqslant d)^5 \geqslant 0.9\}$. This is $d_5 = 20$, which corresponds to a 20:45 meeting time. In this case we will catch our train with probability ≈ 0.951
- For N=10, we want $d_{10} = min\{d \in S \mid P(D_i \leqslant d)^{10} \geqslant 0.9\}$. This is $d_{10} = 20$. We get $0.99^{10} \approx 0.9044$ probability of catching our train. Thus, the meeting time will still be 20:45.

(b) We are given a second random variable, Z_i, associated with the delay of our friends. Z_i takes on one of two values, *punctual (p)* or *not punctual (np)*, with probabilities 2/3 and 1/3 respectively. Therefore, when estimating the delay of our friends, we are now dealing with a joint distribution $F_{D,Z}(d, z)$.
We can compute the conditional distribution $F_{D|Z}(d \mid Z = z)$:

$$P(D \leqslant d \mid Z = p) = \begin{cases} 0.7 & \text{if } d = 0 \\ 0.8 & \text{if } d = 5 \\ 0.9 & \text{if } d = 10 \\ 0.97 & \text{if } d = 15 \\ 0.99 & \text{if } d = 20 \\ 1 & \text{if } d > 20 \end{cases}, P(D \leqslant d \mid Z = np) = \begin{cases} 0.5 & \text{if } d = 0 \\ 0.7 & \text{if } d = 5 \\ 0.8 & \text{if } d = 10 \\ 0.9 & \text{if } d = 15 \\ 0.95 & \text{if } d = 20 \\ 1 & \text{if } d > 20 \end{cases}$$

To obtain the marginal distribution of D, we sum the joint distribution function over Z. Applying Bayes Theorem,

$$\begin{aligned} F_D(d) &= \sum_Z F_{D,Z}(d, z) \\ &= \sum_Z F_{D|Z}(d \mid z) F_Z(z) \\ &= P(D \leqslant d \mid Z = p)P(Z = p) + P(D \leqslant d \mid Z = np)P(Z = np) \end{aligned}$$

We aim to calculate the probabilities of missing our train given the three cases of N from part **(a)** and the meeting times we picked. To catch the train, **everyone** needs to arrive by 21:05. We assume that if $\geqslant$ one person arrives later, we will miss the train (the probability of this is equal to 1 - P(everyone arrives by 21:05)).

- For N=3 and meeting time 20:50, the maximum allowed delay is 15 minutes. The probability we miss our train is therefore $1 - P(D \leqslant 15)^3 = 1 - (\frac{2}{3} \times 0.97 + \frac{1}{3} \times 0.9)^3 \approx 0.1516$.

- For N=5 and meeting time 20:45, the maximum allowed delay is 20 minutes. The probability we miss our train is therefore $1 - P(D \leqslant 20)^5 = 1 - (\frac{2}{3} \times 0.99 + \frac{1}{3} \times 0.95)^5 \approx 0.1113$.
- For N=10 and meeting time 20:45, the maximum allowed delay is again 20 minutes. The probability we miss our train is therefore $1 - P(D \leqslant 20)^{10} = 1 - (\frac{2}{3} \times 0.99 + \frac{1}{3} \times 0.95)^{10} \approx 0.2103$.

BONUS We take our answer for N=5 from part **(a)**: $d_5 = 20$ minutes (i.e. 20:45 meeting time). The probability of missing our train in this case is equal to $1 - P(D \leqslant 20)^5$ (1 - probability everyone will be there by 21:05).

We assume that we missed the train. For this specific case of N=5 and a 20:45 meeting time, we will now explore how this new information affects our belief about the number of our non-punctual friends K. Let us define the posterior distribution of K for our case:

$$P(K = k \mid \text{missed train}) = \frac{P(\text{missed train} \mid K = k)P(K)}{P(\text{missed train})}$$

with $k \in [0, 5]$. Using the conditional distribution from **(b)**,

- the likelihood $P(\text{missed train} \mid K = k) = 1 - [(1 - \frac{k}{5}) \times 0.99 + \frac{k}{5} \times 0.95]^5$,
- prior (for N=5) $P(K) = \frac{1}{3} \times 5 = \frac{5}{3}$, and
- $P(\text{missed train}) = 1 - 0.99^5$.

This posterior distribution gives us the probability that k of our 5 friends are not punctual, given that we set the meeting time for 20:45 and missed our train.

4 Dunwich Hamlet

We have attached our code (commented and explained in accordance with this report) as Appendix Q4.

(a) Let $c_n = a_n + b_n$ where a_n is the number of season ticket holders who attend a game and b_n is the number of 'normal' fans who attend a game. Fixing a and b allows us to write $a_n|a \sim \text{Bin}(a, p_a)$ and $b_n|b \sim \text{Bin}(b, p_b)$, and we can express the distribution $P(c_n = x|a, b)$ as a convolution of the distributions over the possible sums of a_n and b_n for fixed a and b.

$$P(c_n = x|a, b) = \sum_{k=0}^{a} P(a_n = k|a)P(b_n = x - k|b)$$

Which can be written explicitly using the probability mass functions of a_n and b_n as:

$$P(c_n = x|a, b) = \sum_{k=0}^{a} \binom{a}{k} p_a^k (1 - p_a)^{a-k} \binom{b}{x-k} p_b^{x-k} (1 - p_b)^{b+k-x}$$

The expectation of the sum of two random variables is the sum of their expectations, so:

$$\begin{aligned} E[c_n|a, b] &= E[a_n|a, b] + E[b_n|a, b] \\ &= E[a_n|a] + E[b_n|b] \\ &= ap_a + bp_b \end{aligned}$$

Similarly, by independence:

$$\begin{aligned} Var[c_n|a, b] &= Var[a_n|a] + Var[b_n|b] \\ &= ap_a(1 - p_a) + bp_b(1 - p_b) \end{aligned}$$

(b) Consider the quantity $d_n - c_n$, which represents the number of family members attending the game (equivalently, the number of ticket holders who brought a family member with them). It's clear that $(d_n - c_n) \sim \text{Bin}(c_n, p_d)$. Therefore, the distribution of $d_n|c_n$ takes the form of a "shifted binomial distribution", that is: $P(d_n = k|c_n) = P(d_n - c_n = k - c_n|c_n)$. This probability mass function can be written explicitly as:

$$P(d_n = x|c_n) = \binom{c_n}{x - c_n} p_d^{x-c_n} (1 - p_d)^{c_n - (x - c_n)}$$

The expected value can also be expressed using the fact that $(d_n - c_n) \sim \text{Bin}(c_n, p_d)$:

$$\begin{aligned} E[d_n|c_n] &= E[c_n|c_n] + E[d_n - c_n|c_n] \\ &= c_n + c_n p_d \end{aligned}$$

(c)

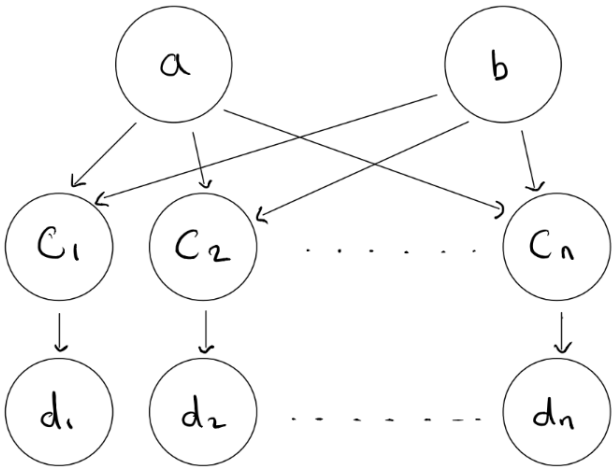

Figure 4: 4c. A belief network, encoding dependencies in our model

(d) We start by decomposing the joint conditional probability through the use of Bayes theorem and the independence statements made clear by our graphical representation:

$$\begin{aligned} P(a,b|d_1,...,d_n) &= \frac{P(a,b,d_1,...,d_n)}{P(d_1,..,d_n)} \\ &= \frac{P(d_1,...,d_n|a,b)P(a,b)}{P(d_1,..,d_n)} \\ &= \frac{P(d_1|a,b)P(d_2|a,b)...P(d_n|a,b)P(a,b)}{P(d_1,..,d_n)} \\ &\propto \prod_i P(d_i|a,b) \end{aligned}$$

Where the last proportionality holds due to the uniform prior on a and b, and the constant probability for the evidence. We can now express each $P(d_i|a,b)$ term using the distributions we arrived at in *a)* and *b)*.

$$\begin{aligned} P(d_i|a,b) &= \sum_{c_i} P(d_i,c_i|a,b) \\ &= \sum_{c_i} P(d_i|c_i,a,b)P(c_i|a,b) \\ &= \sum_{c_i} P(d_i|c_i)P(c_i|a,b) \text{ (due to conditional independence of } a,b \text{ and } d_i \text{ given } c_i) \end{aligned}$$

Where the summation is over all possible values of c_i for a given a and b (i.e. $c_i \in \{0,\ldots,a+b\}$).

We can put all this together to obtain the following result about the joint posterior of a and b given the evidence:

$$P(a,b|d_1,...,d_n) \propto \prod_i \left[\sum_{c_i} P(d_i|c_i)P(c_i|a,b) \right]$$

Since the RHS is computationally feasible, we are able to compute the joint posterior matrix for the given parameters by fixing n and calculating the RHS for all pairs of a and b, and then normalizing. We then marginalize

by summing matrix rows and columns to get $P(a|d_1, ..., d_n)$ and $P(b|d_1, ..., d_n)$, respectively.

Printed below are the plots of the posterior distributions for n = 1,2,...,10. As n increases and the sequence of evidence increases in length, the plots become more red and less transparent.

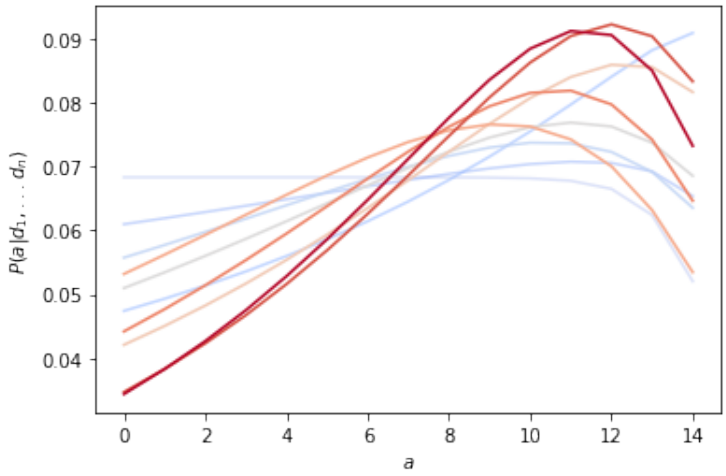

Figure 5: Plots of $P(a|d_1, ..., d_n)$ for n=1,2,...,10

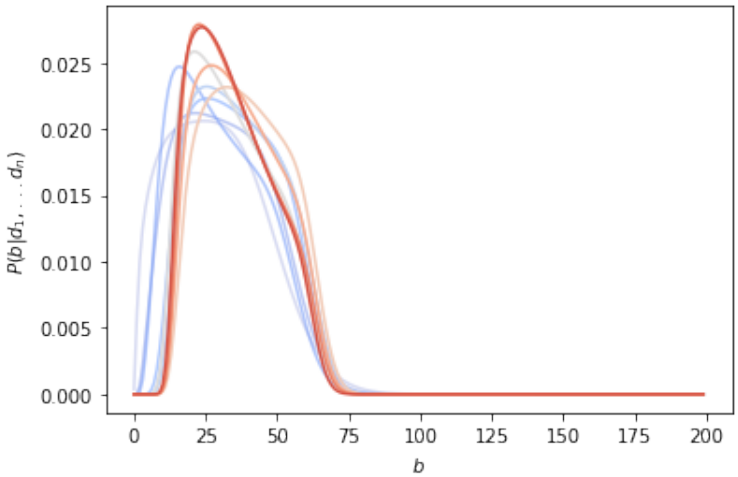

Figure 6: Plots of $P(b|d_1, ..., d_n)$ for n=1,2,...,10

Additionally we plot the full-evidence ($n = 10$) posteriors which are used to calculate our ML/MAP estimates.

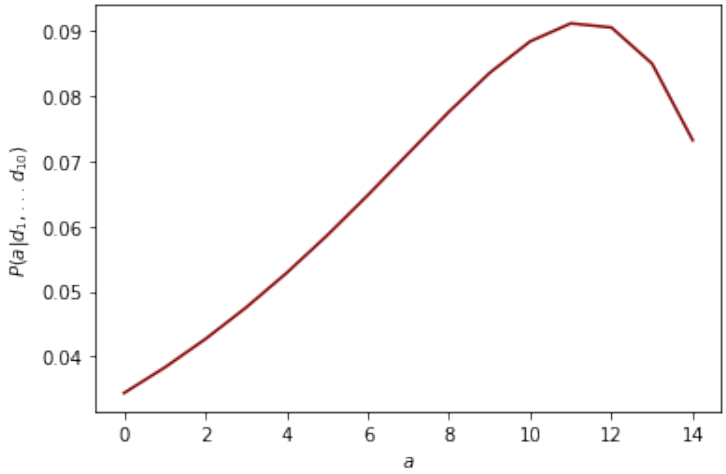

Figure 7: Plot of $P(a|d_1, ..., d_n)$ for the full evidence posterior.

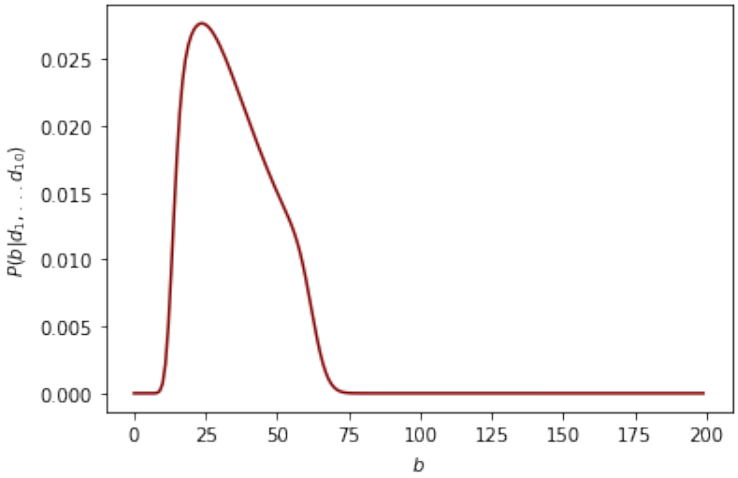

Figure 8: Plot of $P(b|d_1, ..., d_n)$ for the full evidence posterior

Our ML/MAP estimates (which coincide due to our uniform priors on a and b) are as follows:

$$\text{a} = 11$$
$$\text{b} = 24$$

Appendix Q2

November 11, 2019

0.1 Exercise 1.22

part 1.

[ ]:
```
function earthquakes_exercise()

  ###############################################################
  # Get clean data obtained at sensor i

    function value(x_true,y_true,x_sensor,y_sensor)
      d_1 = (x_true[1] - x_sensor)^2 + (y_true[1] - y_sensor)^2
      d_2 = (x_true[2] - x_sensor)^2 + (y_true[2] - y_sensor)^2

      return ((1/(0.1 + d_1)) + (1/(0.1 + d_2)))
    end

  ###############################################################
  # Define the coordinate system

    S = 2000 # number of points on the spiral
    rate = 25 # angular rate of spiral
    sd = 0.2 # standard deviation of the sensor Gaussian noise

    x = zeros(S); y = zeros(S)
    for s = 1:S
        theta = rate*2*pi*s/S;  r = s/S
        x[s] = r*cos(theta); y[s] = r*sin(theta)
    end

    plot(x,y,".", label = "")

  ###############################################################
  # Define the locations of the detection stations on the surface
  # Also define what value on each sensor would be generated by an explostion at␣
↪internal location s

    N = 30 # number of stations
```

```
  x_sensor = zeros(N); y_sensor = zeros(N)
  v = zeros(S^2,N)
  for sensor=1:N
      theta_sensor=2*pi*sensor/N
      x_sensor[sensor]=cos(theta_sensor); y_sensor[sensor]=sin(theta_sensor)
      i = 0
      for s1 = 1:S
        for s2 = 1:S
          i += 1
          v[i,sensor] = value([x[s1] x[s2]],[y[s1]
↪y[s2]],x_sensor[sensor],y_sensor[sensor]) # explosion value for some value
↪function
        end
      end
  end

 ################################################################
 # Store all the possible locations of the 2 explosions and their permutations

  position_combo = zeros(S*S, 4)
  i = 0
  for s1 in 1:S
    for s2 in 1:S
      i += 1
      position_combo[i, :] = [x[s1] x[s2] y[s1] y[s2]]
    end
  end

 ################################################################
 # Load energy values detected at each sensor
  val = [1.5575865910322284, 2.0316965014178545, 2.527159472728958, 2.
↪6723924798859993, 3.0556883949222513, 3.226100565264301, 3.413836775759006, 2.
↪934969359401483, 2.574515504193705, 2.6488388964756617, 2.9253870488198985, 3.
↪1964831294088745, 4.565820133697982, 7.400903824656596, 9.633267837723208, 7.
↪016102363491975, 4.508193821658606, 2.7329525442253204, 1.7254938107215634, 1.
↪4010698300325652, 1.4277427130171043, 0.8691896336292224, 1.2026529289752554,
↪0.5831154459014418, 1.1207179346239071, 1.0105518841821948, 0.
↪8807933908444603, 0.8835180770749729, 1.0899061329336812 ,1.2699960898845595];

 ################################################################
 # Perform inference p(s1, s2| observed sensor values) given these sensor values

  logp = zeros(S^2)
  for s = 1:S^2
      for sensor = 1:N
          logp[s] += -0.5*(val[sensor]-v[s,sensor]).^2/(sd^2) # Gaussian
↪distribution
```

```
        end
    end

# Compute the probabilities
    p = exp.(logp.-maximum(logp)) # do exponentiation (and avoid over/underflow)
    p = p./sum(p); # normalise

#############################################################
# Sum over s2 to infer P(s1|*v*)

    p_s1 = zeros(S)
    for n = 0:S-1 # in these splits, s1 is constant and only s2 is changing
    constant_s1 = p[(n*S+1):(n+1)*S]
    constant_sum = sum(constant_s1)
    p_s1[n+1] = constant_sum
    end

# Find the greatest probability for s1
    max_p, max_ind = findmax(p_s1)

# Find the associated locations
    s1_loc = [x[max_ind], y[max_ind]];
    println("Most likely location of s1: $s1_loc")

#############################################################
# Generate plots

figure()
    for s=1:S
        plot(x[s],y[s],".",color=[1,1,1])
    end

    for theta=0:0.01:2*pi
        plot(cos(theta),sin(theta),".",color=[0,0,0])
    end

        sf=0.2
    for sensor=1:N
        plot(x_sensor[sensor],y_sensor[sensor],"o",color=[1,0,0])
        theta_sensor=2*pi*sensor/N
        x_sensor_nm=(1+sf*val[sensor])*cos(theta_sensor+0.05)
        y_sensor_nm=(1+sf*val[sensor])*sin(theta_sensor+0.05)
        plot([x_sensor[sensor], x_sensor_nm],[y_sensor[sensor],␣
↪y_sensor_nm],"-m")
    end
```

```
    plot(x[max_ind],y[max_ind],"m+",markersize=20,label="estimated (most␣
↪likely)")
    plot(sum(p_s1.*x),sum(p_s1.*y),"go",markersize=10,label="estimated␣
↪(average)")
    legend()

end
```

[3]:
```
using Random
using PyPlot
earthquakes_exercise()
```

0.2 Exercise 1.22

0.2.1 part 2.

Hypothesis 1: 1 EXPLOSION

[ ]:
```
##############################################################
    # Define the coordinate system

    S = 2000 # number of points on the spiral
    rate = 25 # angular rate of spiral
    sd = 0.2 # standard deviation of the sensor Gaussian noise

    x = zeros(S); y = zeros(S)
    for s = 1:S
        theta = rate*2*pi*s/S;  r = s/S
        x[s] = r*cos(theta); y[s] = r*sin(theta)
    end

    ############################################################
    # Load energy values detected at each sensor
    val = [1.5575865910322284, 2.0316965014178545, 2.527159472728958, 2.
↪6723924798859993, 3.0556883949222513, 3.226100565264301, 3.413836775759006, 2.
↪934969359401483, 2.574515504193705, 2.6488388964756617, 2.9253870488198985, 3.
↪1964831294088745, 4.565820133697982, 7.400903824656596, 9.633267837723208, 7.
↪016102363491975, 4.508193821658606, 2.7329525442253204, 1.7254938107215634, 1.
↪4010698300325652, 1.4277427130171043, 0.8691896336292224, 1.2026529289752554,␣
↪0.5831154459014418, 1.1207179346239071, 1.0105518841821948, 0.
↪8807933908444603, 0.8835180770749729, 1.0899061329336812 ,1.2699960898845595];

    ############################################################
```

```
   # Compute sensor locations and all possible v-values, for all s1, s2 at each␣
↪sensor.

   function value(x_true,y_true,x_sensor,y_sensor)
      return 1/(0.1+ (x_true-x_sensor)^2 + (y_true-y_sensor)^2)
   end

   # Define the locations of the detection stations on the surface
   # Also define what value on each sensor would be generated by an explostion␣
↪at internal location s
   N = 30 # number of stations
   x_sensor = zeros(N); y_sensor = zeros(N)
   v = zeros(S,N)
   for sensor=1:N
     theta_sensor=2*pi*sensor/N
     x_sensor[sensor]=cos(theta_sensor); y_sensor[sensor]=sin(theta_sensor)
     for s=1:S
         v[s,sensor]=value(x[s],y[s],x_sensor[sensor],y_sensor[sensor]) #␣
↪explosion value
     end
   end

   ##############################################################
   ## Compute log probabilities

   zum = 0
     for s = 1:S
       log_prod = 0
       for sensor = 1:N
           log_prod += -0.5*(val[sensor]-v[s,sensor])^2/(sd^2)
       zum += exp(log_prod)
           # compute the product of probabilities for each individual sensor
     end
   end
   log_1ex = log(zum)
```

Hypothesis 2: 2 EXPLOSIONS

[ ]:
```
##############################################################
   # Redefine the value function

   function value(x_true,y_true,x_sensor,y_sensor)
     d_1 = (x_true[1] - x_sensor)^2 + (y_true[1] - y_sensor)^2
     d_2 = (x_true[2] - x_sensor)^2 + (y_true[2] - y_sensor)^2

     return ((1/(0.1 + d_1)) + (1/(0.1 + d_2)))
   end
```

```
   ############################################################
   # Define the coordinate system

   S = 2000 # number of points on the spiral
   rate = 25 # angular rate of spiral
   sd = 0.2 # standard deviation of the sensor Gaussian noise

   x = zeros(S); y = zeros(S)
   for s = 1:S
       theta = rate*2*pi*s/S;  r = s/S
       x[s] = r*cos(theta); y[s] = r*sin(theta)
   end

   ############################################################
   # Define the locations of the detection stations on the surface
   # Also define what value on each sensor would be generated by an explostion␣
↪at internal location s

   N = 30 # number of stations
   x_sensor = zeros(N); y_sensor = zeros(N)
   v = zeros(S^2,N)
   for sensor=1:N
       theta_sensor=2*pi*sensor/N
       x_sensor[sensor]=cos(theta_sensor); y_sensor[sensor]=sin(theta_sensor)
       i = 0
       for s1 = 1:S
         for s2 = 1:S
           i += 1
           v[i,sensor] = value([x[s1] x[s2]],[y[s1]␣
↪y[s2]],x_sensor[sensor],y_sensor[sensor]) # explosion value for some value␣
↪function
         end
       end
   end

   ############################################################
   # Compute log probabilities

   zum2 = 0
     for s = 1:S^2
       log_prod2 = 0
       for sensor = 1:N
           log_prod2 += -0.5*(val[sensor]-v[s,sensor])^2/(sd^2)
       zum2 += exp(log_prod2)

     end
```

```
    end

    log_2ex = log(zum2)
```

Compute difference between log probabilities

[ ]:
```
diff = log_2ex - log_1ex
```

1 Exercise 1.23

1.0.1 Part 1.

[ ]:
```
function earthquakes_means_exercise()

  ###############################################################
  # Compute the values at each sensor

    function value(x_true,y_true,x_sensor,y_sensor)
      d_1 = (x_true[1] - x_sensor)^2 + (y_true[1] - y_sensor)^2
      d_2 = (x_true[2] - x_sensor)^2 + (y_true[2] - y_sensor)^2

      return ((0.5/(0.1 + d_1)) + (0.5/(0.1 + d_2)))
    end

  ###############################################################
  # Define the coordinate system

    S = 2000 # number of points on the spiral
    rate = 25 # angular rate of spiral
    sd = 0.2 # standard deviation of the sensor Gaussian noise

    x = zeros(S); y = zeros(S)
    for s = 1:S
        theta = rate*2*pi*s/S;  r = s/S
        x[s] = r*cos(theta); y[s] = r*sin(theta)
    end

    plot(x,y,".", label = "")

  ###############################################################
  # Define the locations of the detection stations on the surface
  # Also define what value on each sensor would be generated by an explostion at␣
↪internal location s

    N = 30 # number of stations
    x_sensor = zeros(N); y_sensor = zeros(N)
    v = zeros(S^2,N)
```

```
  for sensor=1:N
      theta_sensor=2*pi*sensor/N
      x_sensor[sensor]=cos(theta_sensor); y_sensor[sensor]=sin(theta_sensor)
      i = 0
      for s1 = 1:S
        for s2 = 1:S
          i += 1
          v[i,sensor] = value([x[s1] x[s2]],[y[s1]␣
↪y[s2]],x_sensor[sensor],y_sensor[sensor]) # explosion value for some value␣
↪function
        end
      end
  end

 ################################################################
 # Store all the possible locations of the 2 explosions and their permutations

  position_combo = zeros(S*S, 4)
  i = 0
  for s1 in 1:S
    for s2 in 1:S
      i += 1
      position_combo[i, :] = [x[s1] x[s2] y[s1] y[s2]]
    end
  end

 ################################################################
 # Load energy values detected at each sensor
  val = [1.304925002, 1.449069926, 1.265043093, 0.953817726, 1.369084037, 1.
↪313618071, 1.62530874, 1.190647749, 1.325482095, 0.67531377, 0.877754324, 0.
↪596489574, 0.816668218, 0.551164317, 0.634682775, 0.626785942, 0.605713388, 0.
↪394278619, 0.607659282, 0.591969992, 0.755237378, 0.827011395, 0.539143056, 1.
↪036494465, 1.414815944, 1.569445198, 1.821017663, 2.034303308, 1.536641007, 1.
↪476378713]

################################################################
 # Perform inference p(s1, s2| observed sensor values) given these sensor values

  logp = zeros(S^2)
  for s = 1:S^2
      for sensor = 1:N
          logp[s] += -0.5*(val[sensor]-v[s,sensor]).^2/(sd^2) # Gaussian␣
↪distribution
      end
  end

 # Compute the probabilities
```

```
  p = exp.(logp.-maximum(logp)) # do exponentiation (and avoid over/underflow)
  p = p./sum(p); # normalise

# Find the greatest probability for s1,s2
  maxp,maxind = findmax(p)

# Find the associated locations
  opti_positions = position_combo[maxind, :] # format: x1, x2, y1, y2

 ############################################################
# Sum over s2 to infer P(s1|*v*)

p_s1 = zeros(S)
for n = 0:S-1 # in these splits, s1 is constant and only s2 is changing
  constant_s1 = p[(n*S+1):(n+1)*S]
  constant_sum = sum(constant_s1)
  p_s1[n+1] = constant_sum
end

# Find the greatest probability for s1
max_p, max_ind = findmax(p_s1)

# Find the associated locations
s1_loc = [x[max_ind], y[max_ind]];
println("Most likely location of s1: $s1_loc")

############################################################
# Generate plots

figure()
  for s=1:S
      plot(x[s],y[s],".",color=[1,1,1])
  end

  for theta=0:0.01:2*pi
      plot(cos(theta),sin(theta),".",color=[0,0,0])
  end

      sf=0.2
  for sensor=1:N
      plot(x_sensor[sensor],y_sensor[sensor],"o",color=[1,0,0])
      theta_sensor=2*pi*sensor/N
      x_sensor_nm=(1+sf*val[sensor])*cos(theta_sensor+0.05)
      y_sensor_nm=(1+sf*val[sensor])*sin(theta_sensor+0.05)
      plot([x_sensor[sensor], x_sensor_nm],[y_sensor[sensor],␣
↪y_sensor_nm],"-m")
  end
```

```
    plot(x[max_ind],y[max_ind],"m+",markersize=20,label="estimated (most␣
↪likely)")
    plot(sum(p_s1.*x),sum(p_s1.*y),"go",markersize=10,label="estimated␣
↪(average)")
    legend()

end
```

[8]:
```
using PyPlot
using Random
earthquakes_means_exercise()
```

```
Most likely location of s1: [0.4403886448888879, -0.2698705086757887]
```

2 Exercise 1.23

2.0.1 Part 2.

Hypothesis 1: 1 EXPLOSION

[ ]:
```
#############################################################
    # Define the coordinate system

    S = 2000 # number of points on the spiral
    rate = 25 # angular rate of spiral
    sd = 0.2 # standard deviation of the sensor Gaussian noise

    x = zeros(S); y = zeros(S)
    for s = 1:S
        theta = rate*2*pi*s/S;  r = s/S
        x[s] = r*cos(theta); y[s] = r*sin(theta)
    end

    ############################################################
    # Load energy values detected at each sensor
    val = [1.304925002, 1.449069926, 1.265043093, 0.953817726, 1.369084037, 1.
↪313618071, 1.62530874, 1.190647749, 1.325482095, 0.67531377, 0.877754324, 0.
↪596489574, 0.816668218, 0.551164317, 0.634682775, 0.626785942, 0.605713388, 0.
↪394278619, 0.607659282, 0.591969992, 0.755237378, 0.827011395, 0.539143056, 1.
↪036494465, 1.414815944, 1.569445198, 1.821017663, 2.034303308, 1.536641007, 1.
↪476378713]

    ############################################################
    # Compute sensor locations and all possible v-values, for all s1, s2 at each␣
↪sensor.
```

```
    function value(x_true,y_true,x_sensor,y_sensor)
       return 1/(0.1+ (x_true-x_sensor)^2 + (y_true-y_sensor)^2)
    end

    # Define the locations of the detection stations on the surface
    # Also define what value on each sensor would be generated by an explostion␣
↪at internal location s
    N = 30 # number of stations
    x_sensor = zeros(N); y_sensor = zeros(N)
    v = zeros(S,N)
    for sensor=1:N
      theta_sensor=2*pi*sensor/N
      x_sensor[sensor]=cos(theta_sensor); y_sensor[sensor]=sin(theta_sensor)
      for s=1:S
          v[s,sensor]=value(x[s],y[s],x_sensor[sensor],y_sensor[sensor]) #␣
↪explosion value
      end
    end


    ################################################################
    # Compute log probabilities

    zum = 0
    for s = 1:S
    log_prod = 0
        for sensor = 1:N
            log_prod += -0.5*(val[sensor]-v[s,sensor])^2/(sd^2)
        zum += exp(log_prod)

        end
    end
    log_1ex_mean = log(zum)
```

Hypothesis 2: 2 EXPLOSIONS

[ ]:
```
################################################################
    # Get clean data obtained at sensor i

    function value(x_true,y_true,x_sensor,y_sensor)
      d_1 = (x_true[1] - x_sensor)^2 + (y_true[1] - y_sensor)^2
      d_2 = (x_true[2] - x_sensor)^2 + (y_true[2] - y_sensor)^2

      return ((0.5/(0.1 + d_1)) + (0.5/(0.1 + d_2)))
    end

    ################################################################
```

```
   # Define the coordinate system

   S = 2000 # number of points on the spiral
   rate = 25 # angular rate of spiral
   sd = 0.2 # standard deviation of the sensor Gaussian noise

   x = zeros(S); y = zeros(S)
   for s = 1:S
       theta = rate*2*pi*s/S;  r = s/S
       x[s] = r*cos(theta); y[s] = r*sin(theta)
   end

   ############################################################
   # Define the locations of the detection stations on the surface
   # Also define what value on each sensor would be generated by an explostion␣
↪at internal location s
   N = 30 # number of stations
   x_sensor = zeros(N); y_sensor = zeros(N)
   v = zeros(S^2,N)
   for sensor=1:N
       theta_sensor=2*pi*sensor/N
       x_sensor[sensor]=cos(theta_sensor); y_sensor[sensor]=sin(theta_sensor)
       i = 0
       for s1 = 1:S
         for s2 = 1:S
           i += 1
           v[i,sensor] = value([x[s1] x[s2]],[y[s1]␣
↪y[s2]],x_sensor[sensor],y_sensor[sensor]) # explosion value for some value␣
↪function
         end
       end
   end

   ############################################################
   # Store all the possible locations of the 2 explosions and their permutations

   position_combo = zeros(S*S, 4)
   i = 0
   for s1 in 1:S
     for s2 in 1:S
       i += 1
       position_combo[i, :] = [x[s1] x[s2] y[s1] y[s2]]
     end
   end

   ############################################################
   # Load energy values detected at each sensor
```

```
   val = [1.304925002, 1.449069926, 1.265043093, 0.953817726, 1.369084037, 1.
↪313618071, 1.62530874, 1.190647749, 1.325482095, 0.67531377, 0.877754324, 0.
↪596489574, 0.816668218, 0.551164317, 0.634682775, 0.626785942, 0.605713388, 0.
↪394278619, 0.607659282, 0.591969992, 0.755237378, 0.827011395, 0.539143056, 1.
↪036494465, 1.414815944, 1.569445198, 1.821017663, 2.034303308, 1.536641007, 1.
↪476378713]

   ##############################################################
   # Compute log probabilities

   zum2 = 0
   for s = 1:S^2
       log_prod2 = 0
       for sensor = 1:N
       log_prod2 += -0.5*(val[sensor]-v[s,sensor])^2/(sd^2)
       zum2 += exp(log_prod2)
       end
   end
   log_2ex_mean = log(zum2)
```

Compute the difference between log probabilities

[ ]:
```
diff_mean = log_2ex_mean - log_1ex_mean
```

Appendix Q4

November 11, 2019

[ ]:
```
a_min = 0
a_max = 15
b_min = 0
b_max = 200
p_a = 0.99
p_b = 0.3
p_d = 0.5
D = [22, 27, 26, 32, 31, 25, 35, 26, 28, 23]
```

[ ]:
```
all_as = list(range(a_min, a_max + 1))
all_bs = list(range(b_min, b_max + 1))
```

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
```

0.0.1 Utility functions

[ ]:
```
# Function for visualising distributions (used for debugging)

def plot_discrete_dist(dist, x_min, x_max):
    xs = list(range(x_min, x_max))
    f_xs = list(map(lambda x: dist(x), xs))
    plt.plot(xs, f_xs)
    plt.show();
```

0.0.2 Computation and Plots

Our strategy will be to compute the joint conditional distribution $P(a, b|d_1, d_2, \ldots, d_n)$ and then marginalise to get $P(a|d_1, d_2, \ldots, d_n$ and $P(b|d_1, d_2, \ldots, d_n)$.

From our derivation, we know that $P(a, b|d_1, d_2, \ldots, d_n) \propto \Pi_{i=0}^{n} P(d_i|a, b) = \Pi_{i=0}^{n} \left[\sum_{c_i} P(d_i|c_i)P(c_i|a, b)\right]$, so let's create functions for the relevant terms:

First, we showed that $d_n - c_n \sim \text{Bin}(c_n, p_d)$ and hence $P(d_n = k|c_n) = \binom{c_n}{k-c_n} p_d^{k-c_n}(1 - p_d)^{2c_n-k}$ which can be represented as a constant-shifted binomial.

[ ]:
```
from scipy.stats import binom as binom_dist

def P_d_COND_c(d, c):
    return binom_dist.pmf(d - c, c, p_d)
```

We also showed that $P(c = x|a, b) = \sum_{k=0}^{a} \binom{a}{k}\binom{b}{x-k} p_a^k p_b^{x-k}(1-p_a)^{a-k}(1-p_b)^{b-(x-k)}$.

[ ]:
```
from scipy.special import binom as n_choose_k

def P_c_COND_a_b(c, a, b):
    res = 0
    for k in range(0, a + 1):
        a_term = n_choose_k(a, k)*(p_a**k)*((1 - p_a)**(a-k))
        b_term = n_choose_k(b, c - k)*(p_b**(c - k))*((1 - p_b)**(b-(c - k)))
        res += (a_term * b_term)
    return res
```

Now we can write $P(d_i|a, b)$ for each i as $\sum_{c_i} P(d_i|c_i)P(c_i|a, b)$

[ ]:
```
# since we're going to be recomputing some of these terms as we use more and more␣
 ↪data,
# we may as well store values in a lookup table indexed on i to speed things up.
dist_d_COND_a_b = np.full((len(D), a_max, b_max), np.nan)

def P_d_COND_a_b(i, a, b):
    d = D[i]
    if np.isnan(dist_d_COND_a_b[i][a][b]):
        zum = 0
        for c in range(0, a + b):
            zum += P_d_COND_c(d, c)*P_c_COND_a_b(c, a, b)
        dist_d_COND_a_b[i][a][b] = zum

    return dist_d_COND_a_b[i][a][b]
```

Finally we can compute the distribution $P(a, b|d_1, d_2, \ldots, d_n) \propto \Pi_{i=0}^{n} P(d_i|a, b)$ by iterating over possible values of a and b, and then normalizing.

[ ]:
```
from tqdm import tqdm
from itertools import product

def P_a_b_COND_ds(a, b, ds):
    dist = np.zeros(shape=(a_max, b_max))

    all_as = list(range(a))
    all_bs = list(range(b))

    for a, b in tqdm(list(product(all_as, all_bs))):
        prod = 1.

        # take the product of independent data d conditional on a, b
        for i in range(len(ds)):
            prod *= P_d_COND_a_b(i, a, b)

        dist[a][b] = prod

    # normalize!
    dist = dist / dist.sum()
    return dist
```

Compute the posterior distribution for $n = 1, \ldots, 10$

[ ]:
```
dists_a_b_COND_ds = []
for i in range(len(D)):
    dists_a_b_COND_ds.append(P_a_b_COND_ds(a_max, b_max, D[:i + 1]))
```

Now all that's left to do is marginalise to find the posteriors $P(a|d_1, d_2, \ldots, d_n)$ and $P(b|d_1, d_2, \ldots, d_n)$

[ ]:
```
dists_a_COND_ds = list(map(lambda x: x.sum(axis=1), dists_a_b_COND_ds))
dists_b_COND_ds = list(map(lambda x: x.sum(axis=0), dists_a_b_COND_ds))
```

Let's plot and see how our posteriors evolve as as we increase n:

[ ]:
```
for i in range(len(dists_a_COND_ds)):
    dist_a = dists_a_COND_ds[i]
    plt.plot(range(len(dist_a)), dist_a, alpha=min((i+1)*0.2, 1), c=cm.
 ↪coolwarm((i+1)*0.1))
    plt.xlabel('$a$')
    plt.ylabel('$P(a|d_1,...d_n$)')

plt.show()

for i in range(len(dists_b_COND_ds)):
    dist_b = dists_b_COND_ds[i]
    plt.plot(range(len(dist_b)), dist_b, alpha=min((i+1)*0.2, 1), c=cm.coolwarm(i*0.
 ↪1));
    plt.xlabel('$b$')
    plt.ylabel('$P(b|d_1,...d_n$)')

plt.show()
```

(Here, light blue and transparent $\rightarrow$ opaque and red as n increases and we use more evidence.)

Finally let's plot our full-evidence posteriors and calculate ML/MAP estimates for the parameters.

[ ]:
```
dist_a_COND_D = dists_a_COND_ds[-1]
dist_b_COND_D = dists_b_COND_ds[-1]
```

[ ]:
```
plt.plot(range(len(dist_a_COND_D)), dist_a_COND_D, c='maroon')
plt.xlabel('$a$')
plt.ylabel('$P(a|d_1,...d_{10}$)')
plt.show()
plt.plot(range(len(dist_b_COND_D)), dist_b_COND_D, c='maroon')
plt.xlabel('$b$')
plt.ylabel('$P(b|d_1,...d_{10}$)')
plt.show()
```

ML/MAP estimates (i.e. ML = MAP due to uniform prior)

[ ]:
```
print('a:', dist_a_COND_D.argmax())
print('b:', dist_b_COND_D.argmax())
```

Fun plot for the joint distribution! (Note: the MAP esitmate for the joint distribution does not coincide with the marginal estimates! - Independence does not imply conditional independence!)

[ ]:
```
plt.imshow(dists_a_b_COND_ds[-1], cmap='hot', interpolation='nearest')
plt.show()
```