Graphical Models (COMP0080) Assignment

January 2021

Team name: JJJAB

The principles of Inference and Learning were applied in the problems of double earthquakes, meeting scheduling and weather stations' data sampling. The encoding and decoding of LDPC-codes, using Loopy Belief Propagation for Binnary Symmetric Channel, was explored. Exact inference, the Mean Field Approximation and Gibbs Sampling were applied, to find the joint probability distribution of the Ising model on a 10 by 10 lattice.

Contents

1. Inference and Learning 3
1.1 Two earthquakes 3
Question 1 3
Question 2 4
Question 3 4
Question 4 5
1.2 Meeting scheduling 5
Question 1.2 (a) 5
Question 1.2 (b) 6
Bonus Question 7
1.3 Three weather stations 7
Question (a) 7
Question (b) 9
Question (c) 10
Question (d) 10
2. LDPC codes 11
2.1 Background 11
Question 1 12
Question 2 12
Question 3 14
Question 4 14
3. Mean Field Approximation and Gibbs Sampling 14
3.1. Assignment 15
Question 1 15
Question 2 16
Question 3 17
References 18
Appendix 18

1. Inference and Learning

1.1 Two earthquakes

A version of Stuart Russell's earthquake/nuclear blast detection problem, was considered. In this version, two explosions at locations s_1 and s_2 are observed. The locations of the explosions within the Earth, were investigated, based on surface measurements made by sensors, as waves of energy propagated outwards through the Earth. The observed value of the signal of sensor i, is given by

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

where ${d_i}^2(1)$ and ${d_i}^2(2)$, are the distances from explosions 1 and 2 to the sensor, respectively, σ is the standard deviation of the Gaussian sensor noise and ε_i, is noise that is drawn from a zero mean unit variance Gaussian, and is independent for each sensor, i. Data for the observed signals, v_i, were used. The coordinate system used, was a spiral coordinate system. The prior locations of the explosions were assumed to be independent and uniform.

Question 1

The belief network of the two earthquakes problem is given by

$$p(s_1, s_2, v) = p(s_1)p(s_2)\prod_{i=1}^{n} p(v_i|s_1, s_2), \tag{2}$$

where v_i is normally distributed with mean $\frac{1}{{d_i}^2(1)+0.1} + \frac{1}{{d_i}^2(2)+0.1}$ and standard deviation $\sigma = 0.2$.

By Bayes' theorem and by marginalizing s_2, the posterior distribution $p(s_1|\boldsymbol{v})$, for the location of the explosion is given by

$$p(s_1|v) = \frac{p(s_1)\sum_{s_2} p(s_2)\prod_{i=1}^{n} p(v_i|s_1, s_2)}{\sum_{s_1,s_2} p(s_1)\,p(s_2)\prod_{i=1}^{n} p(v_i|s_1, s_2)} = \frac{\sum_{s_2}\prod_{i=1}^{n} p(v_i|s_1, s_2)}{\sum_{s_1,s_2}\prod_{i=1}^{n} p(v_i|s_1, s_2)}, \tag{3}$$

where the assumption $p(s_1) = p(s_2) = \frac{1}{N}$ was made, where N is the number of possible explosion locations. The results for the posterior distributions for the location of the explosions were $p(s_1|v) \approx 0.31$ and $p(s_2|v) \approx 0.5$ and a visualization is seen in Figure 1.

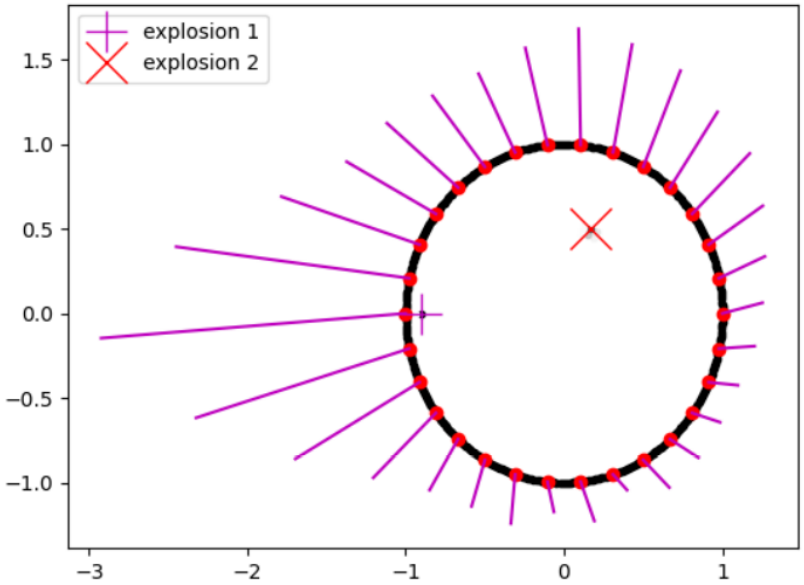

Figure 1. Posterior distributions, $p(s_1|\boldsymbol{v})$ and $p(s_2|\boldsymbol{v})$, for the locations of the explosions (darker corresponds to a higher probability).

Question 2

The hypothesis that the number of explosions occurred were 1 and 2, are denoted by $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. The probability $p(\boldsymbol{v}|\mathcal{H}_i)$, is given by

$$p(\boldsymbol{v}|\mathcal{H}_i) = \frac{p(v|s_k,\mathcal{H}_i)p(s_k|\mathcal{H}_i)}{p(v|s_k,\mathcal{H}_i)}, \tag{4}$$

where $i,k \in \{1,2\}$. Under the hypothesis $\mathcal{H}_2$, s_k takes the values $s_k = (s_1, s_2)$. Hence, $p(s_k|\mathcal{H}_2) = \frac{1}{m^2}$, where m is the number of samples. Thus,

$$p(v|\mathcal{H}_i) = p(s_k|\mathcal{H}_i)\sum_{s_k} p(v|s_k,\mathcal{H}_i) = p(s_k|\mathcal{H}_i)\sum_{s_k}\prod_{j=1}^{n} p(v_j|s_k,\mathcal{H}_i), \tag{5}$$

Hence, the value of

$$\log p(\boldsymbol{v}|\mathcal{H}_2) - \log p(\boldsymbol{v}|\mathcal{H}_1), \tag{6}$$

was found to be approximately 754.

Question 3

Given that there is no prior preference, in other words,

$$p(\mathcal{H}_1) = p(\mathcal{H}_2) = \text{const}, \tag{7}$$

the value of $\log p(\boldsymbol{v}|\mathcal{H}_2) - \log p(\boldsymbol{v}|\mathcal{H}_1)$ is related to the probability that there are two explosions compared to one. Since $\log x$ is a monotonically increasing function for all $x > 0$, the value of $\log x$ is larger, the greater the value of x is. Also, since $\log p(v|\mathcal{H}_2) \approx -10$ and $\log p(v|\mathcal{H}_1) \approx -764$, it follows that

$p(v|\mathcal{H}_2) > p(v|\mathcal{H}_1)$, which clearly indicates that the observed sensor values are more likely, given that there were two explosions.

Question 4

For two explosions with m points in the coordinate system, the corresponding matrix contained $\frac{m^2}{2}$ entries, which correspond to the number of possible combinations of locations for the explosions. Hence, for the number of explosions occurred $K \geq 2$, the complexity of calculating $\log p(\boldsymbol{v}|\mathcal{H}_K)$ is of order $O\left(\left(\frac{m}{2}\right)^k\right)$.

1.2 Meeting scheduling

A train trip to Scotland is organized by an individual for their N friends. A meeting is arranged at the point of departure, the train ticket gates, since the organizer possesses all the tickets. The train departs at 21:05, so the organizer and the N friends need to meet prior to that time. The amount of time prior to the train's departure that the friends are told to arrive at the meeting place, considering the possibility that they will be delayed, was investigated. The solution found, satisfies the condition that the time of the meeting should be as late as possible, but also that the friends catch the train with probability 0.9.

The model for the delays used, assumes that the organizer arrives consistently on time. Each friend, i, of the total of N friends, arrives with a delay denoted by D_i. All delays are assumed to be independent and identically distributed. The probabilities for the different values of the delays are as follows:

$$P(D_i \leq 0) = 0.7, \tag{8}$$

$$P(0 \text{ min} < D_i < 5 \text{ min}) = 0.1, \tag{9}$$

$$P(5 \text{ min} \leq D_i < 10 \text{ min}) = 0.1, \tag{10}$$

$$P(10 \text{ min} \leq D_i < 15 \text{ min}) = 0.07, \tag{11}$$

$$P(15 \text{ min} \leq D_i < 20 \text{ min}) = 0.02, \tag{12}$$

$$P(20\text{min} \leq D_i) = 0.01. \tag{13}$$

Question 1.2 (a)

The time, $T_0 = T_0(N)$, at which the meeting is scheduled, was found for the values of $N = 3$, $N = 5$ and $N = 10$. In order for all the friends to arrive on time for the trip, all friends must arrive during the time interval between T_0 and 21:05, denoted as $I = [T_0, 21{:}05]$. The event that a friend, i, arrives within the interval, I, is denoted by A_i. The probability that all friends are on time, $p(A)$, where $A = \{A_i, \dots, A_N\}$, is given by

$$p(A) = \prod_{n=1}^{N} p(A_n) = p(A_i)^N, \tag{14}$$

since the delays of each friend are assumed to be independent.

The probabilities that a friend i will arrive on time, A_i, were calculated for the different possible intervals, I, and are given by

$$I = [21{:}00, 21{:}05] \Rightarrow p(A_i) = p(0 < D_i < 5) + p(D_i \leq 0) = 0.8, \quad (15)$$

$$I = [20{:}55, 21{:}05] \Rightarrow p(A_i) = 0.9, \quad (16)$$

$$I = [20{:}50, 21{:}05] \Rightarrow p(A_i) = 0.97, \quad (17)$$

$$I = [20{:}45, 21{:}05] \Rightarrow p(A_i) = 0.99, \quad (18)$$

Using Equation (14), the probabilities, $p(A)$, for the different values of N, were found and the smallest intervals I, for each N, for which $p(A) \geq 0.9$, were determined. For the value of $N = 3$, the smallest interval for which $p(A) \geq 0.9$, was determined to be $I = [20{:}50, 21{:}05]$, i.e. $T_0(3) = 20{:}50$, with the value of $p(A) = 0.91$. Similarly, for $N = 5$, the interval was found to be $I = [20{:}45, 21{:}05]$, i.e. $T_0(5) = 20{:}45$, with a corresponding value of $p(A) = 0.95$. For $N = 10$, the interval was found to be $I = [20{:}45, 21{:}05]$, i.e. $T_0(10) = 20{:}45$, with a corresponding value of $p(A) = 0.90$.

Question 1.2 (b)

The unobserved binary variables Z_i, were added to the model, as a result of a difference in punctuality among the friends. In this updated model, the probabilities given above, correspond to $P(D_i|Z_i = \text{punctual})$. The respective probabilities, in the same order, for non-punctual friends, are given by

$$P(D_i|Z_i = \text{not punctual}) = (0.5, 0.2, 0.1, 0.1, 0.05, 0.05). \quad (19)$$

The prior belief for the probability that a friend will be punctual is given by

$$p(Z_i = \text{punctual}) = \frac{2}{3}. \quad (20)$$

The probability of missing the train, $p(\text{missing the train})$, is given by

$$p(\text{missing the train}) = 1 - p(\text{no one is late}) \quad (21)$$

$$= 1 - \prod_{i=1}^{N} p(A_i) = 1 - \left(\sum_{z} p(Z_i) p(A_i|Z_i) \right)^N. \quad (22)$$

Using Equation (22), the probabilities of missing the train if the meeting times were set to 20:50, 20:45, and 20:45 for $N = 3$, $N = 5$ and $N = 10$, respectively, were found to be 0.152, 0.111 and 0.21, respectively.

Analytically, the probabilities were calculated as follows:

$$\text{For } I = [20{:}50, 21{:}05] \text{ and } N = 3, \quad p(\text{missing the train}) = 1 - \left(\frac{1}{3}0.9 + \frac{2}{3}0.97\right)^3 = 0.152,$$

$$\text{For } I = [20{:}45, 21{:}05] \text{ and } N = 5, \quad p(\text{missing the train}) = 1 - \left(\frac{1}{3}0.95 + \frac{2}{3}0.99\right)^5 = 0.111,$$

$$\text{For } I = [20{:}45, 21{:}05] \text{ and } N = 10, \quad p(\text{missing the train}) = 1 - \left(\frac{1}{3}0.95 + \frac{2}{3}0.99\right)^{10} = 0.21.$$

Bonus Question

Given that the friends, with a population of $N = 5$, missed the train, and the meeting time was set to the answer in Question 1.2 (a), $T_0(5) = 20{:}45$, the number of non-punctual friends was investigated and the posterior distribution of this number was found. In this Section, the event in which all friends arrive on time is denoted by B, the event in which the train was missed is denoted by M, and the event in which i out of the 5 friends are not punctual, is denoted by $kP = i$. The probability $p(kP = i|M)$, is given by

$$p(kP = i|M) = \frac{p(kP = i)p(M|kP = i)}{p(M)}, \tag{23}$$

for $i \in \{0,1,2,3,4,5\}$. The probability $p(M)$, which is the normalization constant of Equation (23), was not explicitly calculated.

If k out of 5 friends are not punctual, the probability that all friends arrive on time, $p(B)$, equals the product of the probabilities that all of the $N - k$ punctual friends are on time and that all of the k non punctual friends are on time. In other words,

$$p(B) = p(A_i|Z_i = \text{non punctual})^k p(A_i|Z_i = \text{punctual})^{N-k} = \left(0.95^k 0.99^{N-k}\right). \tag{24}$$

Hence, it is deduced that

$$p(M|kP = k) = 1 - p(B) = 1 - \left(0.95^k 0.99^{N-k}\right). \tag{25}$$

In order to calculate the probability $p(kP)$, a binomial distribution was used, with parameters $N = 5$ and $p = \frac{1}{3} = p(Z_i = \text{not punctual})$. The results are given by

$$p(kP = i) \approx \begin{cases} 0.132 & i = 0 \\ 0.329 & i = 1 \\ 0.329 & i = 2 \\ 0.165 & i = 3 \\ 0.041 & i = 4 \\ 0.004 & i = 5 \end{cases}, \tag{26}$$

Thus, Equation (23), becomes

$$p(kP = i|M) = \frac{p(kP = i)p(M|kP = i)}{p(M)}) \approx \begin{cases} 0.061 & i = 0 \\ 0.262 & i = 1 \\ 0.373 & i = 2 \\ 0.244 & i = 3 \\ 0.073 & i = 4 \\ 0.012 & i = 5 \end{cases}. \tag{27}$$

1.3 Three weather stations

Question (a)

Data from three weather stations, was used, without knowledge of which data sequences originate from which station. Hence, the three stations, are the hidden variables and the data sequences are the visible variables. The data sequences are assumed to follow a first order Markov chain. The set $S = \{1,2,3\}$, denotes the set of weather stations from which the sequences, denoted by

$$\omega^{(n)} = \{\omega_1, \dots, \omega_M\}, \tag{28}$$

originate, where $M = 100$ and $n \in \{1, \dots, 500\}$ and $\omega^{(n)} = s$ denotes that the sequence $\omega^{(n)}$, comes from station $s \in$ S.

The probability that $\omega^{(n)} = s$, is denoted by $\alpha(s) = p(\omega^{(n)} = s)$. The transition matrices, $A^{(s)} \in R^{3\times3}$, with $s \in$ S, are defined, with entries given by

$$A_{i,j}{}^{(s)} = p\left(\omega_{t+1}{}^{(n)} = i \middle| \omega_t{}^{(n)} = j, \omega^{(n)} = s\right), \tag{29}$$

and the associated initial probability vector $\pi_i{}^{(s)} \in R^n$ is given by

$$\pi_i{}^{(s)} = p\left(\omega_1{}^{(n)} = i \middle| \omega^{(n)} = s\right). \tag{30}$$

The graphical model for the three weather stations problem is seen in Figure 2.

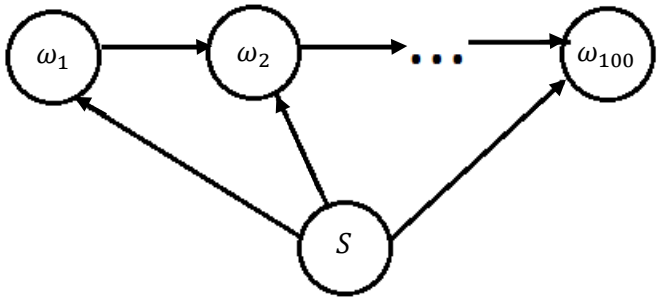

Figure 2. Graphical model for the three weather stations problem.

The Expectation Maximization, EM, algorithm consists of two steps, the Expectation step (E-step), and the Maximization step, (M-step). The steps of the EM algorithm are tabulated in Table 1.

Algorithm 1 EM Algorithm. Input: a distribution $p(w|\theta)$ and dataset W. Returns ML candidate $\theta = \{A^{(s)}, \pi^{(s)}, \alpha(s)\}_{s\in S}$

Initialize θ^0
while θ not converged **do** $m \leftarrow m + 1$
  for $n = 1$ to N **do** ▷ This is the E-step
    $q_m^n(s^{(n)}|w^{(n)}) = p(s^{(n)}|w^{(n)}, \theta^{m-1})$
  end for
    update θ^m ▷ This is the M-step
  end while

Table 1. Outline of the steps of the EM algorithm for the three weather stations problem.

The update of the parameters is performed as follows:

$$A_{i,j}^{(s),\text{new}} = \frac{\sum_{n=1}^{N} p(w^{(n)} = s|x^{(n)}, \theta^{\text{old}}) \sum_{m=1}^{M} I[w_m^{(n)} = i] I[w_{m-1}^{(n)} = j]}{\sum_{r}^{3} \sum_{n=1}^{N} p(w^{(n)} = s|x^{(n)}, \theta^{\text{old}}) \sum_{m=1}^{M} I[w_m^{(n)} = r] I[w_{m-1}^{(n)} = j]} \tag{31}$$

$$\pi_i^{(s),\text{new}} = \frac{\sum_{n=1}^{N} p(w^{(n)} = s|w^{(n)}, \theta^{\text{old}}) I[w_1^{(n)} = i]}{\sum_{r}^{3} \sum_{n=1}^{N} p(w^{(n)} = s|w^{(n)}, \theta^{\text{old}}) I[w_1^{(n)} = r]} \tag{32}$$

$$\alpha(s)^{\text{new}} = \frac{\sum_{n=1}^{N} p(s|w^{(n)}, \theta^{\text{old}})}{\sum_{s} \sum_{n=1}^{N} p(s|w^{(n)}, \theta^{\text{old}})} \tag{33}$$

Question (b)

The EM algorithm was implemented using the data from the three stations. The probability that $\omega^{(n)} = s$, denoted as, $\alpha(s)$, was found to be

$$\alpha(s) = \begin{pmatrix} 0.192 \\ 0.508 \\ 0.290 \end{pmatrix}. \tag{34}$$

The initial probability vectors were given by

$$\pi^{(1)} = \begin{pmatrix} 0.427 \\ 0.573 \\ 0 \end{pmatrix}, \tag{35}$$

$$\pi^{(2)} = \begin{pmatrix} 0.461 \\ 0.331 \\ 0.208 \end{pmatrix}, \tag{36}$$

and

$$\pi^{(3)} = \begin{pmatrix} 0.192 \\ 0.419 \\ 0.389 \end{pmatrix}. \tag{37}$$

The transition matrices were given by

$$A^{(1)} = \begin{pmatrix} 0.389 & 0.241 & 0.545 \\ 0.148 & 0.515 & 0.018 \\ 0.463 & 0.244 & 0.437 \end{pmatrix}, \tag{38}$$

$$A^{(2)} = \begin{pmatrix} 0.071 & 0.137 & 0.511 \\ 0.495 & 0.226 & 0.437 \\ 0.435 & 0.637 & 0.052 \end{pmatrix}, \tag{39}$$

and

$$A^{(3)} = \begin{pmatrix} 0.060 & 0.130 & 0.418 \\ 0.140 & 0.354 & 0.428 \\ 0.801 & 0.546 & 0.155 \end{pmatrix}. \tag{40}$$

The value of the Log-likelihood was found to be -96.356. Finally, the ten rows of the posterior distribution of $s \in$ S, were given by

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.98 & 0.02 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0.006 & 0.994 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0.003 & 0.997 \\ 0 & 0.012 & 0.988 \end{pmatrix}. \tag{41}$$

Question (c)

The learned parameters were heavily influenced by the initial guess. The choice of initial values is of great importance, as it heavily influences the speed of convergence of the algorithm and its ability to locate the global maximum. As stated in [1], it is advisable to use a mixed strategy, by starting from several different initial guesses, making a small number of iterations, without necessarily examining convergence, and then, running until convergence from the point with the largest likelihood after these initial iterations, using a strict criterion. Such an approach, helps in reducing the amount of time spent in areas of a flat likelihood, away from the global maximum. The initialization strategy we used, was to sample uniformly all of the parameters and then normalize. The computational issue we encountered was floating point underflow which was caused by calculating the values of $p(s_i|\omega)$. These values were extremely small, which resulted in them being represented as zero. This problem was mitigated by calculating $\sum_j \log p(s_i|\omega_j{}^{(n)})$, for the values of $n \in \{1, \dots, 500\}$, and then normalizing.

Question (d)

The problem arising from uniform initial parameter distributions is seen via the update functions, which become

$$A_{i,j}^{s,\,\text{new}} = \frac{\sum_{n=1}^{N} \frac{1}{3} \sum_{m=1}^{M} I[w_m^{(n)} = i][w_{m-1}^{(n)} = j]}{\sum_s \sum_{n=1}^{N} \sum_{m=1}^{M} I[w_m^{(n)} = s] I[w_{m-1}^{(n)} = j]} \tag{42}$$

$$\pi_i^{s,\text{new}} = \frac{\sum_{n=1}^{N} \frac{1}{3} I[w_1^{(n)} = i]}{\sum_s \sum_{n=1}^{N} \frac{1}{3} I[w_1^{(n)} = s]} \tag{43}$$

$$\alpha(h)^{\text{new}} = \frac{1}{3} \tag{44}$$

Hence, an identical update is applied, which results in identical distribution for each state of s, and identical parameter estimates of all of the parameters. The results with uniformly distributed initial guesses, were found to be

$$\pi^{(s)} = \begin{pmatrix} 0.374 \\ 0.404 \\ 0.222 \end{pmatrix}, \tag{45}$$

and

$$A^{(s)} = \begin{pmatrix} 0.159 & 0.145 & 0.487 \\ 0.306 & 0.283 & 0.351 \\ 0.534 & 0.572 & 0.163 \end{pmatrix}, \tag{46}$$

for all $s \in$ S.

2. LDPC codes

2.1 Background

In this Section, the addition and multiplication operations are operations in $\mathbb{F}_2$, which means that they are as follows: $0 + 1 = 1 + 0 = 1$, while $1 + 1 = 0 + 0 = 0$, $1 \cdot 0 = 0 \cdot 1 = 0 \cdot 0 = 0$ and $1 \cdot 1 = 1$. A transmitted message consists of blocks of length K, each given by $t \in \{0,1\}^K$. The code is a subset of 2^K codewords of length N. The ratio $\frac{K}{N}$, is called the rate of the code. If all the codewords form a K-dimensional linear subspace of $\{0,1\}^N$, the code is said to be linear. A parity check matrix, H, is a matrix of full-rank size $(N - K, N)$, that defines such a subspace. The codewords are thus the solutions of $Hx = 0$.

Given a parity check matrix, H, a generator matrix, G, is found. The columns of G consist of the basis vectors of the codeword subspace, if known. Thus, the codewords will be given by $x = Gt$. Systematic encoding refers to the situation where all bits of t are coped to the specific location of the transmitted message x. Gaussian elimination is a way of constructing a systematic encoder, G. Up to the permutation of columns, the echelon form of H is equivalent to $[PI_{N-K}]$, and G is selected to be $[I_K P]^T$, since $HGt = 0$ for every t.

Decoding is performed using Loopy Belief Propagation. The transmitted and received vectors are denoted by x and y, respectively, and the noise model specifies the conditional probability distributions $P(y|x)$. In the case of the Binary Symmetric Channel model, each bit is independently flipped with probability p, so

$$P(y|x) = \prod_{n=1}^{N} p(y_n|x_n) = \prod_{n=1}^{N} p^{x_n - y_n}(1-p)^{x_n - y_n + 1}. \tag{47}$$

Thus,

$$p(x, y) = p(y|x)p(x), \tag{48}$$

where $p(x)$ is a uniform prior distribution over all valid codewords, given by,

$$p(x) \propto \mathbb{I}[Hx = 0]. \tag{49}$$

2.2 Questions

Question 1

A function was written so that for a given input matrix, H, it returns a systematic encoding matrix G for it, as well as a matrix, $\widehat{H}$, which is equal to H up to a column permutation and $\widehat{H}Gt = 0$ for all t.

The function was tested using the matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \tag{50}$$

and the outputs were

$$\widehat{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \tag{51}$$

and

$$G = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}. \tag{52}$$

Question 2

A factor-graph was drawn for the matrix given in Equation (50), and it is shown in Figure 3. The probability that the variable $x_n = i$, given that parity check m is satisfied, denoted by ${q^i}_{mn}$, and the probability that the parity check m is satisfied, given that $x_n = i$, denoted as ${r^i}_{mn}$, are given by

$$q^i_{mn} = \Pr(x_n = i | r_{\{N(m)\setminus m\},n}) := \mu_{s_n \to f_m} \tag{53}$$

$$r^i_{mn} = \Pr(\textstyle\sum_{j \in N(m)} x^j == 0 | x^n = i, x^{n'} \sim q_{mn'}) := \mu_{f_m \to s_m} \tag{54}$$

The update rule for ${r^i}_{mn}$, is given by

$$r^i_{mn} = \textstyle\sum_{\mathbf{x}_{\{\mathbf{n}' \in \mathbf{N}(\mathbf{m})\setminus\{\mathbf{n}\}\}}} \Pr(\sum_{x^j \in \mathbf{x}_{\mathbf{n}'}} x^j == 0 \mod 2 | x_n = i) * \prod_{n' \in N(m)\setminus\{n\}} q^{x_{n'}}_{mn'} \tag{55}$$

A more efficient computation of the update rule given by Equation (55), is achieved by defining $\delta x = x^1 - x^0$, and noting that

$$\delta r_{mn} = (-1)^{z_m} \textstyle\prod_{n' \in N(m)\setminus\{n\}} \delta q_{mn'} \tag{56}$$

Thus, it is possible to recover

$$r^i_{mn} = \tfrac{1}{2}(1 + (-1)^i * \delta r_{mn}) \tag{57}$$

The update rule for ${q^i}_{mn}$, is given by

$$q^i_{mn} \propto p^i_n \prod_{m' \in M(n) \setminus \{m\}} r^i_{m'n} \tag{58}$$

where the expression given requires normalization. The marginal probability will be such that

$$q^i_n \propto \prod_{m \in M(n)} r^i_{mn} = q^i_{mn} * r^i_{mn} \tag{59}$$

The distribution is given by

$$\frac{1}{Z} \prod_m \phi_m : \phi_m = \Pr(\sum_{i \in Dom(\phi_m)} x^i == 0 \mod 2) \tag{60}$$

where the domain of m is given by the columns in the Hamming matrix of row m, that are equal to 1. The marginal distribution is given by

$$p(x_i) = \sum_{x \setminus_i} \frac{1}{Z} \prod_m \phi_m \tag{61}$$

And the joint distribution is given by

$$p(y, x) = p(y|x)p(x) \tag{62}$$

where

$$p(y_i|x_i) = \begin{cases} p = 0.1 & \text{if} y_i == x_i \\ 1 - p = 0.9 & \text{if} y_i \neq x_i \end{cases} \text{ and } p(y|x) = \prod_i p(y_i|x_i) \tag{63}$$

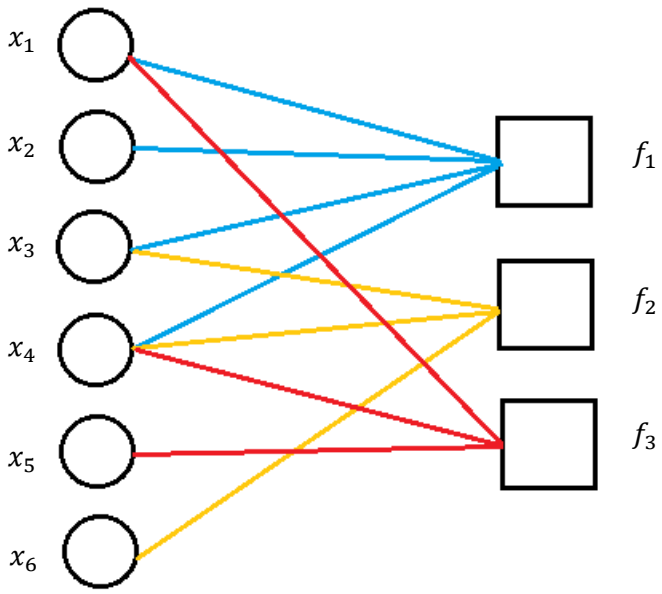

Figure 3. Factor-graph of matrix given in Equation (50).

Question 3

An LDPC-decoder was written, as described in Section 2.1. The function receives a parity check matrix, $\widehat{H}$, a word y, a noise ration p and the maximum number of iterations. The default number of iterations was set to 20. The output of the function is a decoded vector. In addition, it returns 0 if the decoding was successful and -1 if the decoding was unsuccessful after the maximum number of operations.

The parity check matrix, H_1, contained in H1.txt, the vector, y_1, contained in y1.txt, as well as a noise ratio value of $p = 0.1$, were used to check the function. The algorithm converged after 8 iterations and the output of the function was the following series of binary digits:

[0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0
0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 0 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 1 0
0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 0 1 0 0 1
1 1 1 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1
0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 0 1 1
1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 1 1
0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1
0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 1 0 1 0 1
0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1
1 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1
1 0 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 1 0 1 1 0 1 0 0 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1
1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1
0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0 0
0 0 0 0 1 1 0 1 1 0 1 1 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0
1 0 1 0 1 1 0 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1
0 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 1 1 1 0]

Question 4

The original English message was recovered by reading off the first 248 bits of the 252-bit message, and by treating them as a sequence of 31 ASCII symbols. The original English message was found to be

Happy Holidays! Dmitry&David :)

3. Mean Field Approximation and Gibbs Sampling

The Ising model on the $n \times n$ lattice was considered, with potentials given by

$$P(x) = Z^{-1} \prod_{i>j} \varphi(x_i, x_j), \tag{64}$$

where

$$\varphi(x_i, x_j) = e^{\beta\, \mathbb{I}[x_i = x_j]}, \tag{65}$$

for i a neighbor of j on a lattice and $i > j$.

3.1. Assignment

The joint probability distribution of the top and bottom nodes of the rightmost column of the 10×10 lattice, was computed. In other words, the probability table for $P(x_{1,10}, x_{10,10})$, was found, where $x_{i,j}$ is the node in the i-th row and j-th column. The values of β investigated were $\beta = 4$, $\beta = 1$ and $\beta = 0.01$.

Question 1

Firstly, the joint probability distribution was found by performing exact inference, by treating each column as one variable with 2^n states and performing message passing on the induced factor-graph. To start, we stack the 10 by 10 lattice and change variable domain from binary to 2^{10}. Then, we change original potential to column potential by multiplication. The column potential is the product of all potentials between two columns. This, includes all potentials that connect the two columns (horizontal potential in lattice graph) and all potentials that connect rows within the two columns (vertical potential in lattice graph). After calculating all column potentials, we perform sum product algorithm on this new singly-connected graph and marginalize over the column node X_{10}. So now we have an expression for $p(X_{10})$ so we can just calculate all possible $P(X_{10})$. Then, to find $P(x_{1,10}, x_{10,10})$, we simply sum over all possible $X_{2,10} \dots X_{9,10}$ value. For example, if we want to find $P(x_{1,10} = 1, x_{10,10} = 1)$, we sum over all probabilities in which $x_{1,10} = 1$ and $x_{10,10} = 1$ and all possible value for $X_{2,10} \dots X_{9,10}$.

The Equation used for calculating potentials between nodes in the lattice:

$$\varphi(x_i, x_j) = \frac{1}{Z} \exp\left(-\beta \sum_{i>j} \mathbb{I}(x_i = x_j) \right). \tag{66}$$

To calculate column potentials in the singly-connected graph, we simply find the product between all potentials calculated using above equation in the adjacent two columns in lattice graph. For example, the potential between the first two columns, $\varphi(X_1, X_2)$, equals the expression:

$$\varphi(x_{1,1}, x_{1,2})\varphi(x_{1,1}, x_{2,1})\varphi(x_{2,2}, x_{1,2})\varphi(x_{2,2}, x_{2,1}) \dots \varphi(x_{9,1}, x_{9,2})\varphi(x_{9,1}, x_{10,1})\varphi(x_{9,2}, x_{10,2})\varphi(x_{10,1}, x_{10,2}),$$

where capital X represents a column node and lower case x represents lattice node.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.48201	0.01799
$x_{1,10} = -1$	0.01799	0.48201

Table 2. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 4$, calculated using the exact inference.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.25638	0.24362
$x_{1,10} = -1$	0.24362	0.25638

Table 3. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 1$, calculated using the exact inference.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.48201	0.01799
$x_{1,10} = -1$	0.01799	0.48201

Table 4. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 0.1$, calculated using the exact inference.

Question 2

The Mean Field Approximation and coordinate ascent were used. The joint probability of the lattice is given by the expression

$$p(\boldsymbol{x}) = \frac{1}{Z}\exp\left(-\beta \sum_{i>j} \mathbb{I}\left(x_i = x_j\right)\right). \tag{67}$$

However, due to the complex and computationally intensive nature of the distribution (there are 81 dependencies between neighbors), the joint probability distribution was approximated by a separable distribution $\log(q(\boldsymbol{x}|\boldsymbol{a})) \propto \langle \boldsymbol{a}, \boldsymbol{x} \rangle$. In other words, the marginal distributions are assumed to be independent from one another in this new distribution, $q(x)$. Since x is binary, any general distribution of $q(x_i)$, is given by

$$q(x_i) = \begin{cases} q_i & \text{if } x_i == -1 \\ 1 - q_i & \text{if } x_i == 1 \end{cases} \tag{68}$$

The free energy of this distribution is given by

$$E_{q_x} \log \frac{p(x)}{q(x)} = H(q) - \sum_x \beta q(x) \sum_{j>i} \mathbb{I}(x_i = x_j) - \log Z \tag{69}$$

which is rewritten as

$$E_{q_x} \log \frac{p(x)}{q(x)} = H(q) - \sum_x \beta q(x) \sum_{j>i} \frac{(x_i * x_j)+1}{2} \tag{70}$$

by exploiting the fact that

$$\sum_{j>i} \frac{(x_i * x_j)+1}{2} = 1 \Leftrightarrow x_i = x_j, \sum_{j>i} \frac{(x_i * x_j)+1}{2} = 0 \Leftrightarrow x_i \neq x_j \tag{71}$$

Coordinate ascent is performed in order to maximize the free energy. Hence, Equation (70) was differentiated with respect to q_i, and the derivative was set to zero, in order to arrive at the expression

$$q_i = \frac{1}{1+e^{-\beta \sum_{m \in N(i)} (\bar{x}m)}} = \frac{e^{\beta \sum_{m \in N(i)} (\bar{x}m)}}{1+e^{\beta \sum_{m \in N(i)} (\bar{x}m)}} \tag{72}$$

where $\bar{x}_j = E_{q_j} x_j = 2q_j - 1$

The results for the joint probability distributions, $P(x_{1,10}, x_{10,10})$, for the values of β of 4, 1, and 0.1, are tabulated in Tables 5, 6 and 7, respectively.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.99933	0.00034
$x_{1,10} = -1$	0.00034	1.13679×10^{-7}

Table 5. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 4$, calculated using the Mean Field Approximation method.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.20869	0.24862
$x_{1,10} = -1$	0.24862	0.20869

Table 6. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 1$, calculated using the Mean Field Approximation method.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.25000	0.25000
$x_{1,10} = -1$	0.25000	0.25000

Table 7. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 0.1$, calculated using the Mean Field Approximation method.

Question 3

Finally, Gibbs sampling was used for the task of Questions 1 and 2 of Section 3. The 10 by 10 lattice was initiated randomly with the binary values 1 and -1. The probability of a given sample, x_i, to be drawn and have the value of 1, is given by

$$P(x_i = 1 | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_{100}) = \frac{\prod_j \varphi(x_i = 1, x_j)}{\sum_{x_i} \prod_j \varphi(x_i, x_j)}, \tag{73}$$

where the index j loops over the neighbors of node x_i in the lattice. Thus, the probability of x_i being drawn and having the value of -1 is given by

$$P(x_i = -1 | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_{100}) = 1 - P(x_i = 1 | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_{100}). \tag{74}$$

Using Equations (73) and (74), a new lattice was sampled from the randomly initialized lattice. The new lattice was used to sample the next lattice and the process was repeated for 100000 iterations, to ensure the algorithm converged. From the 100000 sampled lattices, the joined probability distribution, $P(x_{1,10}, x_{10,10})$, was found. The results for the joint probability distributions, $P(x_{1,10}, x_{10,10})$, for the values of β of 4, 1, and 0.1, are tabulated in Tables 8, 9 and 10, respectively.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.00000	0.00069
$x_{1,10} = -1$	0.00046	0.99885

Table 8. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 4$, calculated using the Gibbs sampling method.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.28000	0.22501
$x_{1,10} = -1$	0.21417	0.28082

Table 9. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 1$, calculated using the Gibbs sampling method.

	$x_{10,10} = 1$	$x_{10,10} = -1$
$x_{1,10} = 1$	0.24591	0.25093
$x_{1,10} = -1$	0.25169	0.25147

Table 10. Results for the probability table of $P(x_{1,10}, x_{10,10})$, for $\beta = 0.01$, calculated using the Gibbs sampling method.

References

[1] Dimitris Karlis, Evdokia Xekalaki, Choosing initial values for the EM algorithm for finite mixtures.

Appendix

1.1 Code for two earthquakes

```
using DelimitedFiles
using PyPlot
function earthquake_exercise_setup()

    figure()
    S=2000
    rate=25
    sd=0.2
    N=30

    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

    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])
        end
    end

    eeData = readdlm("earthquakeExerciseData.csv")

    logp_two = zeros(S,S)
```

```
    for s1 = 1:S
        for s2 = 1:S
            for sensor = 1:N
                logp_two[s1,s2] += -0.5*(eeData[sensor] - v[s1, sensor] -
v[s2, sensor])^2/(sd^2)
            end
        end
    end

    p_two = exp.(logp_two.-maximum(logp_two))

    p_two = p_two/sum(p_two)
    maxp, maxind = findmax(p_two)
    figure()
    prob = sum(p_two, dims=2)
    maxp1, maxpind1 = findmax(prob)
    for s=1:length(x)
        plot(x[s], y[s],".", color=(1-(prob[s]/maxp1))*[1,1,1])
    end

    for t=0:0.01:2*pi
        plot(cos(t), sin(t), ".", 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*eeData[sensor])*cos(theta_sensor+0.05)
        y_sensor_nm=(1+sf*eeData[sensor])*sin(theta_sensor+0.05)
        plot([x_sensor[sensor], x_sensor_nm],[y_sensor[sensor],
y_sensor_nm],"-m")
    end
    plot(x[maxind[2]], y[maxind[2]], "m+", markersize=20, label =
"explosion 1")
    plot(x[maxind[1]], y[maxind[1]], "rx", markersize=20, label =
"explosion 2")
    legend()
    savefig("explosions.png")

    logp_one = zeros(S)
    for s=1:S
        for sensor=1:N
            logp_one[s] += -0.5*(eeData[sensor]-v[s,sensor])^2/(sd^2)
        end
    end

    # Hypothesis ratio calculation
    max_two = maximum(logp_two)
    p_two = exp.(logp_two.-max_two)
    logp2 = log(sum(p_two))+max_two
    max_one = maximum(logp_one)
    p_one = exp.(logp_one.-max_one)
    logp1 = log(sum(p_one))+max_one
```

```
    print(logp2-logp1)
end

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

earthquake_exercise_setup()
```

1.3 Code for three weather stations

```
import numpy as np

df = np.genfromtxt('meteo1.csv', dtype='int')
alpha = np.random.uniform(0, 1, size = (3)); alpha = alpha / np.sum(alpha)
init_vector = np.random.uniform(0, 1, size = (3,3)); init_vector =
init_vector / np.sum(init_vector, axis = 0).reshape(1, -1)
mat = np.random.uniform(0, 1, size = (3,3,3)); mat = mat / np.sum(mat,
axis = 0).reshape(1, 3, 3)
# EM loop
for iteration in range(15):
    # E - step
    q_h = np.zeros((df.shape[0], 3))
    for i in range(df.shape[0]):
        df_sequence = df[i, :]
        visibleF = df_sequence[0]
        visibleT = df_sequence[1:]
        visibleL = df_sequence[:-1]
        q_h[i, :] = np.log(alpha) + np.log(init_vector[visibleF, :]) +
np.sum(np.log(mat[visibleT, visibleL, :]), axis = 0)
    q_h -= np.max(q_h, axis = 1).reshape(-1, 1)
    qA = np.exp(q_h)
    qA = qA / np.sum(qA, axis = 1).reshape(-1, 1)
    # M-step
    alpha = np.sum(qA, axis = 0) / np.sum(qA)
    for i in range(init_vector.shape[0]):
        init_vector[i, :] = np.sum((df[:, 0] == i).reshape(-1, 1) * qA,
axis = 0)
    init_vector = init_vector / np.sum(init_vector, axis = 0).reshape(1, -
1)
    for i in range(mat.shape[0]):
        for j in range(mat.shape[1]):
            tran = (df[:, 1:] == i) * (df[:, :-1] == j)
            c = np.sum(tran, axis = 1)
            mat[i,j, :] = np.sum(c.reshape(-1, 1) * qA, axis = 0)
    mat = mat / np.sum(mat, axis = 0)
```

2. Code for LDPC codes

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
"""
Created on Mon Dec 20 11:21:22 2021
"""

import numpy as np
import sympy


def Generator(H):
    H_rref, index = sympy.Matrix(H).rref()
    H_rref = np.array(H_rref.tolist()) % 2
    K = max(index)
    I = H_rref[:,:K+1]
    P = H_rref[:,K+1:]
    H_hat = np.concatenate((P,I),axis = 1)
    G = np.concatenate((I,P), axis = 0)

    return H_hat, G


def horizontal_step(H, M):
    factor_var_matrix = np.zeros(H.shape)
    for i in range(len(H)):
        result = 1
        neighbors = np.where(H[i,:]==1)[0]
        for j in neighbors:
            other_neighbors = neighbors[neighbors != j]
            result = np.prod(np.tanh(0.5* M[i,other_neighbors]))
            factor_var_matrix[i,j] = np.log((1+result)/(1-result))

    return factor_var_matrix


def vertical_step(H, M, l, fm):
    m, n = M.shape
    for j in range(n):
        neighbors = np.where(H[:,j] == 1)[0]
        for i in neighbors:
            other_neighbors = neighbors[neighbors != i]
            M[i,j] = np.sum(fm[other_neighbors,j]) + l[j]

    return M


def LDPC(H, y, p, max_iteration=20):
    likelihood = []
    for i in range(len(y)):
        if(y[i] == 1):
            likelihood.append(np.log(p/(1-p)))
        else:
            likelihood.append(np.log((1-p)/p))

    M = H * np.array(likelihood).T
    sucess = -1
```

```
    for i in range(1, max_iteration):
        fm = horizontal_step(H, M)
        M = vertical_step(H, M, likelihood, fm)
        c = likelihood + np.sum(M, axis = 0)
        decoded = np.where(c<0, 1, 0)

        if(np.all(H.dot(decoded) % 2 == 0)):
            sucess = 0
            break

    return decoded, sucess, i


def to_english(message):
    text = []
    i = 0
    while(i < 248):

text.append(chr(int(''.join(message[i:i+8].astype(int).astype(str)), 2)))
        i = i + 8
    text = ''.join(text)

    return text


if __name__ == '__main__':

    H = np.array([[1,1,1,1,0,0],
                  [0,0,1,1,0,1],
                  [1,0,0,1,1,0]])

    H_hat,G = Generator(H)
    print("H = \n", H)
    print("H_hat = \n", H_hat)
    print("G = \n", G)

    check = np.matmul(H_hat, G) % 2 #should be equal 0

    H1 = np.loadtxt("H1.txt", delimiter = " ")
    y1 = np.loadtxt("y1.txt", delimiter = " ")
    sys_G = np.loadtxt("sys_G.txt", delimiter = " ")

    message, sucess, iterations = LDPC(H1, y1, p=0.1)

    print(message)
    print("Sucess: ",sucess)
    print("Iterations: ", iterations)

    english_message = to_english(message)

    print(english_message)
```

3.1 Code for exact inference

```
import itertools


def pass_message(x_i: np.array, x_j:np.array, probs, beta):

    assert x_i.shape[0] == 2**x_i.shape[1]
    assert x_j.shape[0] == 2**x_j.shape[1]
    assert probs.shape[0] == x_i.shape[0]

    passed_message = np.zeros(x_j.shape[0])
    for j in range(x_i.shape[0]):
        col = x_i[j,:]
        #Within vector
        next_node = np.roll(col, -1)
        #Only take N-1 first rows because of how this is a bit hacky (Nth
row makes no sense)
        is_same_within = ((col * next_node + np.ones_like(col))/2)[:-1]

        pot_within = np.exp(beta* np.sum(is_same_within))

        #Across vectors
        is_same_across = (col.flatten() * x_j + np.ones_like(x_j))/2
        pot_across = np.exp(beta * np.sum(is_same_across, axis = 1))

        passed_message = passed_message + pot_within*pot_across*probs[j]

    return passed_message/np.sum(passed_message)


def exact_inference(X, beta, N):
    #Assume unifrom prior (result will depend on prior)
    prior = np.ones(X.shape[0])/X.shape[0]

    probs_xi = prior
    for i in range(N-1):
        probs_xi = pass_message(np.copy(X), np.copy(X), probs_xi, beta)

    prob_x_00 = np.sum(probs_xi[(X[:,0] == -1) & (X[:,9] == -1)])
    prob_x_01 = np.sum(probs_xi[(X[:,0] == -1) & (X[:,9] == 1)])
    prob_x_10 = np.sum(probs_xi[(X[:,0] == 1) & (X[:,9] == -1)])
    prob_x_11 = np.sum(probs_xi[(X[:,0] == 1) & (X[:,9] == 1)])

    return [prob_x_00, prob_x_01, prob_x_10, prob_x_11]


def solve_question3(N):
    betas = np.array([4.0, 1.0, 0.01])
    X = np.array(list(itertools.product([-1,1], repeat=10)))

    results = {}
    for beta in betas:
        result = exact_inference(X, beta, N)
```

```
        results[beta] = {'-1,-1' : result[0], '-1,1' : result[1],
                         '1,-1' : result[2], '1,1': result[3]}

    return results

solve_question3(10)
```

3.2 Code for Mean Field Approximation

```
import numpy as np

N = 10
iterations = 10
beta = 4

def find_neighbours(matrix, i, j):

    #Neighbours in this case refers to \sum_neighbours 2(q_ij) - 1
    try:
        neighbour1 = 2*matrix[i-1][j] - 1
    except:
        neighbour1 = 0

    try:
        neighbour2 = 2*matrix[i][j-1] - 1
    except:
        neighbour2 = 0

    return(neighbour1, neighbour2)


def mf_approx_q(iterations, q0, beta):
    q = q0
    for iter in range(iterations):
        for i, q_i in enumerate(q):
```

```
            for j, q_ij in enumerate(q_i):
                neighbours = find_neighbours(q, i, j)
                a_n = beta * np.sum(neighbours)
                q[i,j] = np.divide(np.exp(a_n), 1+ np.exp(a_n))


    return q

def mf_approx_marginal(indices: list, iterations, q0, beta):
    q = mf_approx_q(iterations, q0, beta)

    return [q[i, j] for i,j in indices]


def solve_question2(iterations, N):
    betas = np.array([4.0, 1.0, 0.01])
    q0 = np.random.uniform(0,1, size = (N,N))
    results = {}
    for beta in betas:
        result = mf_approx_marginal([(0,9), (9,9)], iterations, q0, beta)

        results[beta] = {'-1,-1' : np.prod(result), '-1,1' :
np.prod([result[0], 1 - result[1]]),
                         '1,-1' : np.prod([1 - result[0], result[1]]),
'1,1': np.prod([np.ones(2) - result])}

    return results

solve_question2(20, 10)
```

3.3 Code for Gibbs sampling

```
import numpy as np
```

```
lattice_init = np.random.choice([-1, 1], size = (10,10))
print('Visualization of a randomly initialized lattice')
print(lattice_init)

def find_neighbors(lattice, i, j):
    """
    Function that returns te neighbors of a node in a lattice
    Input:
    lattice -- 10 by 10 array of binary values (1 or -1)
    i -- scalar, the horizontal coordinate of a point
    j -- scalar, the vertical coordinate of a point

    Output:
    neighbors -- list of tuples, containing the coordinates of the
                 node with coordinates i and j
    """
    #Initialize the list of neighbors
    neighbors = []

    #Append the neighbors without thinking of the border cases
    neighbors.append((i+1, j))
    neighbors.append((i-1, j))
    neighbors.append((i, j+1))
    neighbors.append((i, j-1))

    #Initialize the list with the true neighbors
    neighbors_ = []

    #Isolate the neighbors that are within the bounds of the 10 by 10
lattice
    for coordinate in neighbors:
        if coordinate[0] != -1 and coordinate[1] != -1 and coordinate[0]
!= 10 and coordinate[1] != 10:
            neighbors_.append(coordinate)

    return neighbors_

def gibbs_sampling(beta, iterations):
    """
    Function that performs Gibbs sampling
    Input:
    beta -- scalar
    iterations -- scalar

    Output:
    prob_1_1 -- scalar, probability that x_1,10 and x_10,10 are both 1
    prob_0_0 -- scalar, probability that x_1,10 and x_10,10 are both -1
    prob_1_0 -- scalar, probability that x_1,10 and x_10,10 are 1 and -1,
respectively
    prob_0_1 -- scalar, probability that x_1,10 and x_10,10 are -1 and 1,
respectively
    """
    #Initialize the lattice randomly
    lattice_init = np.random.choice([-1, 1], size = (10,10))
```

```
    samples = lattice_init.flatten()
    lattice = samples.reshape((10, 10))
    #Initialize the probabilities to zero
    prob_1_1 = 0
    prob_0_0 = 0
    prob_1_0 = 0
    prob_0_1 = 0

    for iteration in range(iterations):
        for i in range(10):
            for j in range(10):

                #Find the index in the flattened array corresponding
                #to a coordinate in the lattice
                index = 10*i+j

                #Denominator calculation for the probability of a sample
being 1
                ps = find_neighbors(lattice, i, j)
                product_1 = 1
                product_2 = 1
                for p in ps:
                    if lattice[p[0]][p[1]] != 1:
                        I_ = 0
                    elif lattice[p[0]][p[1]] == 1:
                        I_ = 1
                    phi_1 = np.e ** (beta * I_)
                    product_1 = product_1 * phi_1
                for p in ps:
                    if lattice[p[0]][p[1]] != -1:
                        I_ = 0
                    elif lattice[p[0]][p[1]] == -1:
                        I_ = 1
                    phi_1 = np.e ** (beta * I_)
                    product_2 = product_2 * phi_1
                prob_denominator = product_1 + product_2

                #Numerator calculation for the probability of a sample
being one
                prob_numerator = 1
                points = find_neighbors(lattice, i, j)
                for point in points:

                    if lattice[point[0]][point[1]] != 1:
                        I = 0
                    elif lattice[point[0]][point[1]] == 1:
                        I = 1

                    phi_1 = np.e ** (beta * I)
                    prob_numerator = prob_numerator * phi_1

                #Divide the numerator and denominator found above to find
the
                #probability of the sample to be 1
```

```
                p1 = prob_numerator/prob_denominator
                #Draw a sample from this distribution
                sample = np.random.choice([-1, 1], p = [1-p1, p1])
                samples[index] = sample
                lattice = samples.reshape((10, 10))

        #For each of the lattices, check the values of x_1, 10 and x_10,10
        #and update the probabilities accordingly
        if lattice[0][9] == 1 and lattice[9][9] == 1:
            prob_1_1 += 1
        elif lattice[0][9] == -1 and lattice[9][9] == -1:
            prob_0_0 += 1
        elif lattice[0][9] == 1 and lattice[9][9] == -1:
            prob_1_0 += 1
        elif lattice[0][9] == -1 and lattice[9][9] == 1:
            prob_0_1 += 1
    #Divide the count by how many iterations were done
    prob_1_1, prob_1_0, prob_0_1, prob_0_0 = prob_1_1/iterations,
prob_1_0/iterations, prob_0_1/iterations, prob_0_0/iterations


    return prob_1_1, prob_1_0, prob_0_1, prob_0_0
```