AI LAB 2 REPORT

Implementation of Gibbs Sampler, Bayesian Networks, Markov Decision Process and Hidden Markov Models

Abhinav Khanna(2017CSB1061)

CS512: Artificial Intelligence

Part I

Gibbs Sampler

INTRODUCTION

In this section, I have first derived the conditional distributions on a bivariate Gaussian Normal Distribution, and using that a Gibbs Sampler has been implemented. It works on two values of a: a=0 and a=0.99

A: Conditional Probability Distribution

In the following images, I have derived the conditional probability distributions P(x1 | x2) and P(x2 | x1)

Part 1 A) For a general bivariate Normal distribution

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix} \Rightarrow \Sigma^{-1} = \frac{1}{|\Sigma|} \begin{pmatrix} \sigma_2^2 & -\sigma_{12} \\ -\sigma_{12} & \sigma_1^2 \end{pmatrix}$$

where $|\Sigma| = \sigma_1^2 \sigma_2^2 - (\sigma_{12})^2$

$$P(x_1, x_2) = \frac{1}{(2\pi)^{d/2} (|\Sigma|)^{1/2}} \exp\left\{ -\frac{1}{2} (x-\mu)^T (\Sigma^{-1}) (x-\mu) \right\}$$

Let $Q(x_1, x_2) = (x-\mu)^T \Sigma^{-1} (x-\mu)$

$$\Rightarrow \frac{1}{|\Sigma|} \left(\begin{bmatrix} x_1 - \mu_1 , & x_2 - \mu_2 \end{bmatrix} \begin{bmatrix} \sigma_2^2 & -\sigma_{12} \\ -\sigma_{12} & \sigma_1^2 \end{bmatrix} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix} \right) = Q(x_1, x_2)$$

$\therefore$ ~~$P(x_1, x_2) = \frac{1}{2\pi}$~~

$|\Sigma| = \sigma_1^2 \sigma_2^2 - (\sigma_{12})^2$

$= \sigma_1^2 \sigma_2^2 (1 - \rho^2)$

$\therefore P(x_1, x_2) = \frac{1}{2\pi \sigma_1 \sigma_2 \sqrt{(1-\rho^2)}} \left[-\frac{1}{2} Q(x_1, x_2) \right]$

$\therefore P(x_1 | x_2) = \frac{P(x_1, x_2)}{P(x_2)}$

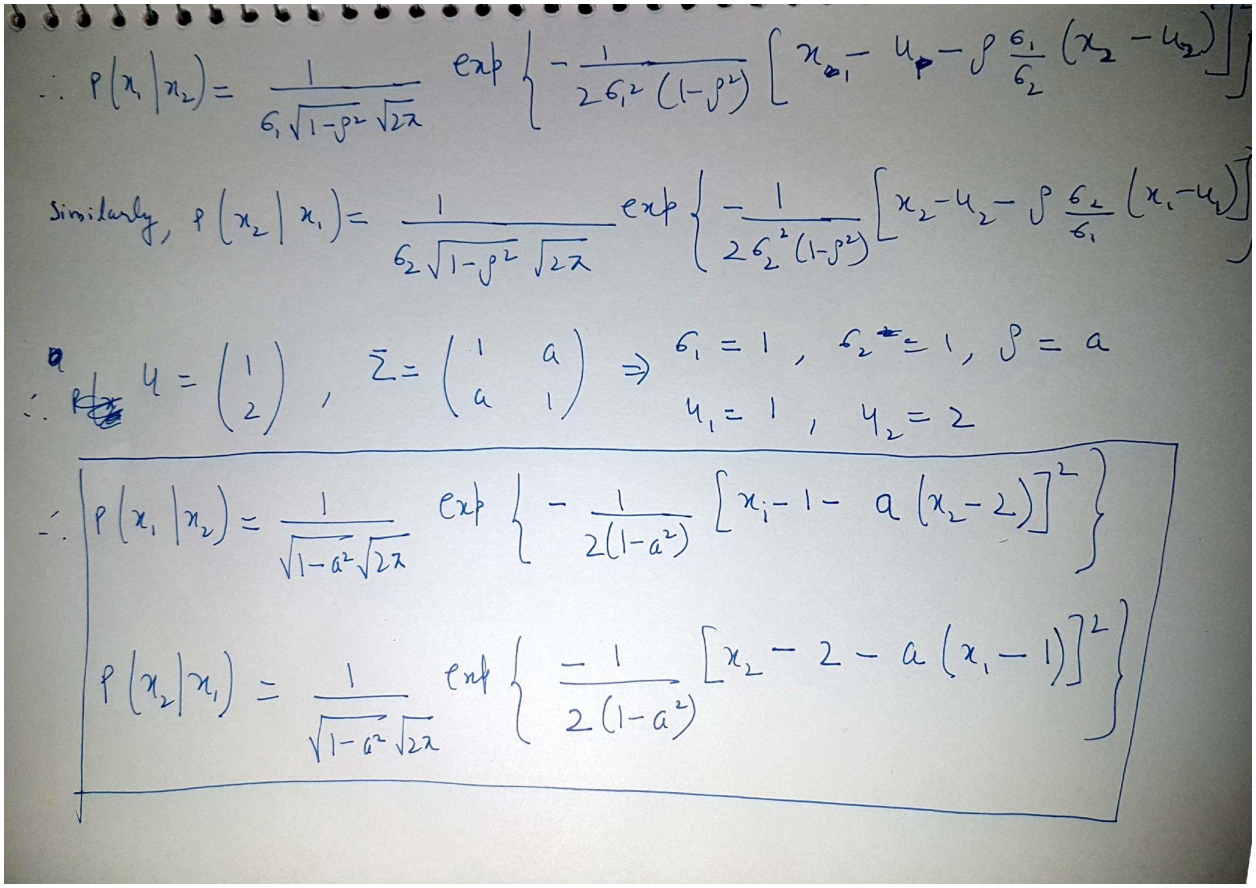
$\Rightarrow \exp\left\{ \frac{-1}{(1-\rho^2)} \left[\left(\frac{x_1 - \mu_1}{\sigma_1}\right)^2 + \left(\frac{x_2 - \mu_2}{\sigma_2}\right)^2 - 2\left(\frac{x_1 - \mu_1}{\sigma_1}\right)\left(\frac{x_2 - \mu_2}{\sigma_2}\right) \right] \right\} \Big/ 2\pi \sigma_1 \sigma_2 \sqrt{1-\rho}$

$\exp\left\{ -\frac{1}{2} \left(\frac{x_2 - \mu_2}{\sigma_2}\right)^2 \right\} \Big/ \sigma_2 \sqrt{2\pi}$

$$\Rightarrow \exp\left\{ \frac{-1}{2(1-\rho^2)} \left[\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2 - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \right] + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2 \right\}}{\sigma_1\sqrt{1-\rho^2}\ \sqrt{2\pi}}$$

$$\Rightarrow \exp\left\{ \frac{-1}{2\sigma_1^2(1-\rho^2)} \left[(x_1-\mu_1)^2 + \rho^2\frac{\sigma_1^2}{\sigma_2^2}(x_2-\mu_2)^2 - 2\rho\frac{\sigma_1}{\sigma_2}(x_1-\mu_1)(x_2-\mu_2) \right] \right\}}{\sqrt{2\pi}\ \sigma_1\sqrt{1-\rho^2}}$$

$$\Rightarrow \frac{1}{\sigma_1\sqrt{1-\rho^2}\ \sqrt{2\pi}} \exp\left\{ \frac{-1}{2\sigma_1^2(1-\rho^2)} \left[x_1-\mu_1 - \rho\frac{\sigma_1}{\sigma_2}(x_2-\mu_2) \right]^2 \right\}$$



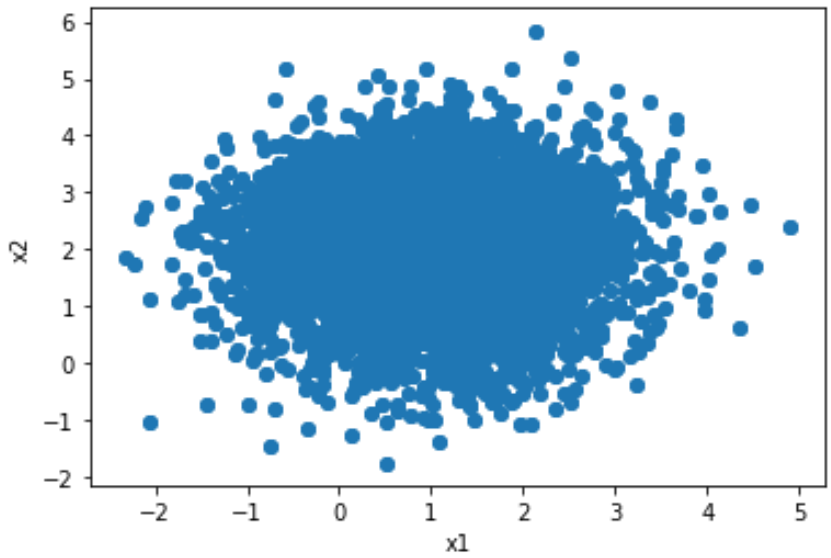
B, C and D:

Having derived the conditional probability distributions, I have implemented a gibbs sampler for a = 0 and a = 0.99. Also, mean and covariance matrix are estimated from the 10,000 samples generated.

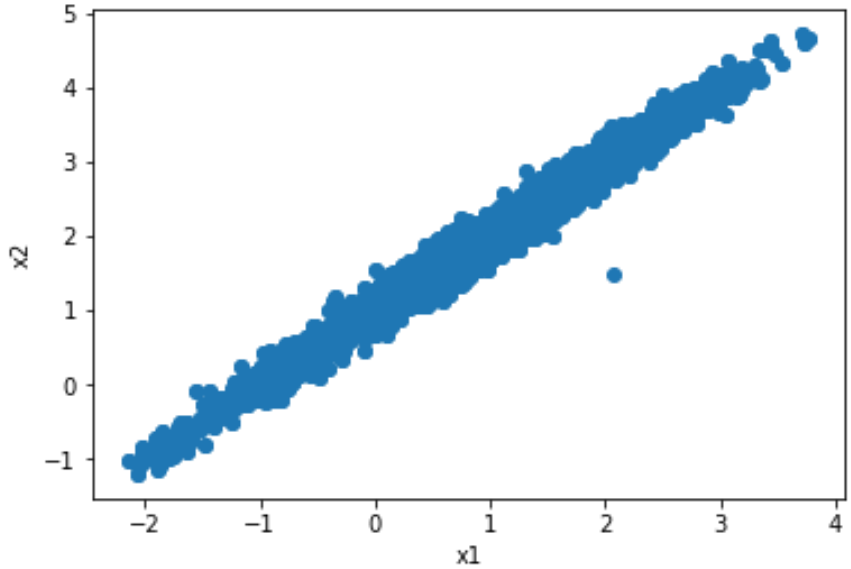
To run the code for a= 0, go to directory “Code/Part 1” and run the jupyter notebook “Gibbs_Sampler(a=0).ipynb”. Similarly, for a = 0.99, run the notebook “Gibbs_Sampler(a=0.99).ipynb”.

OBSERVATION

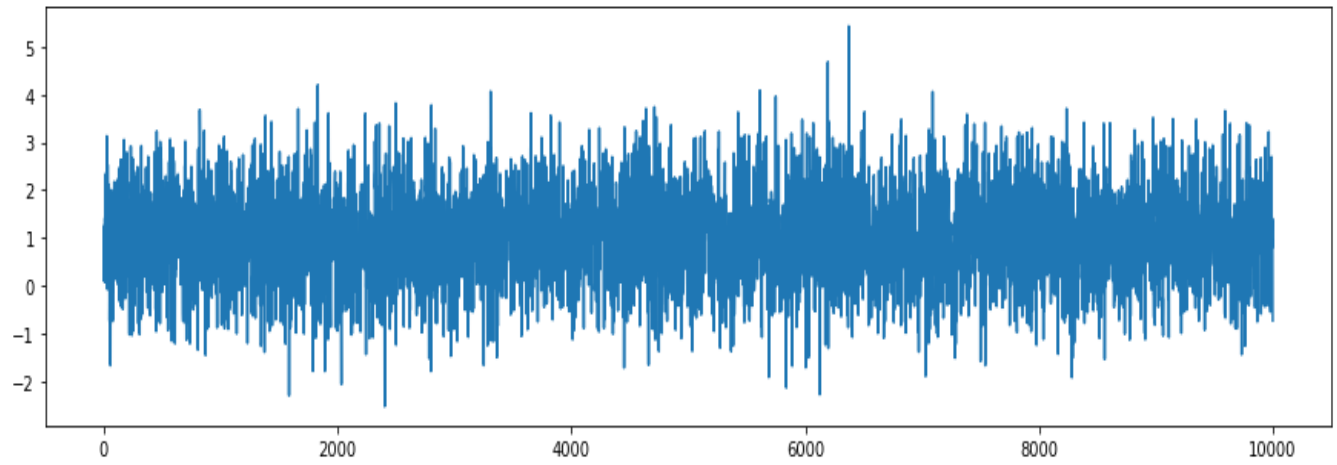
Simulation Plot for **a** = **0**



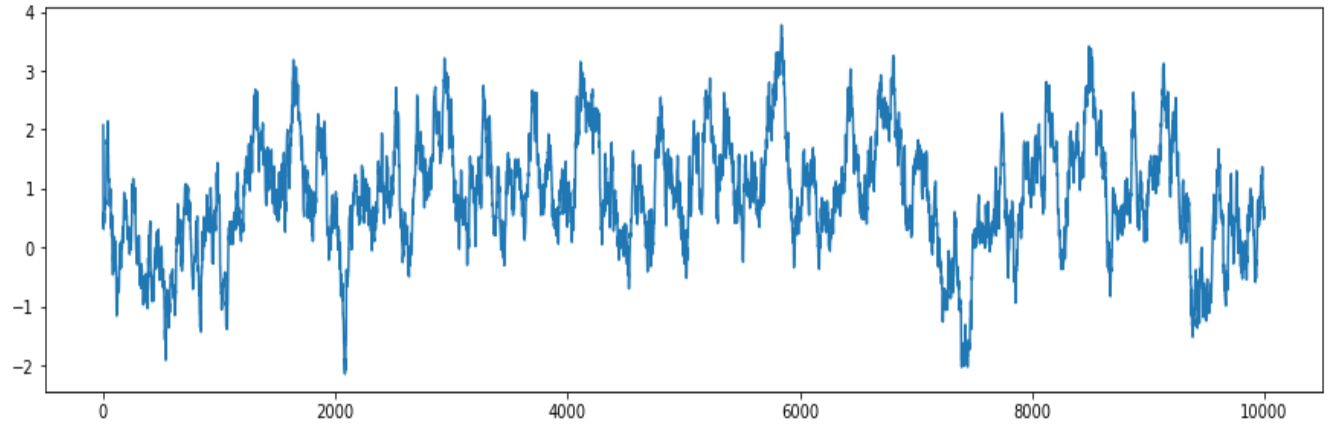
Simulation Plot for **a** = **0.99**



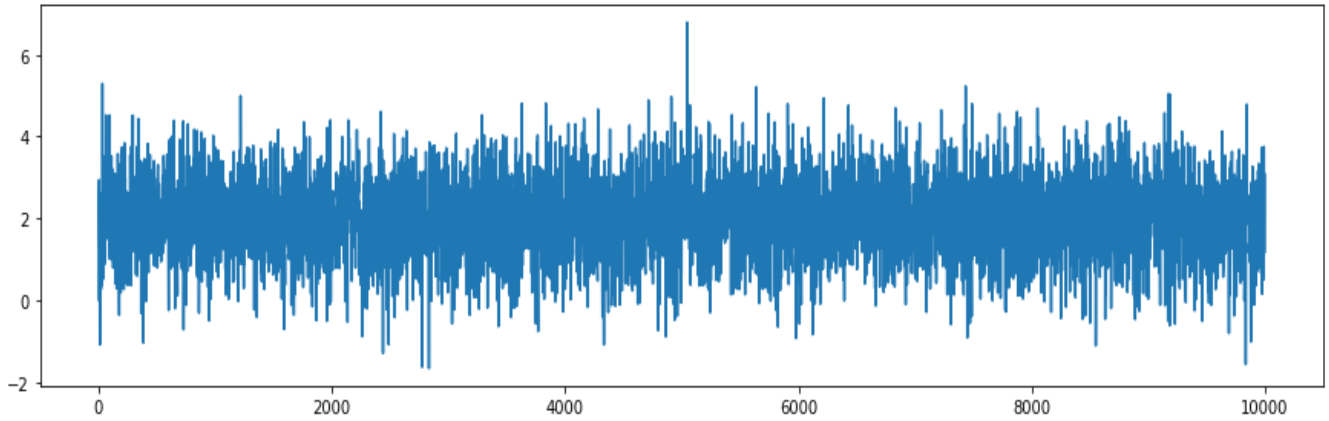
Traceplot for **x1** for **a** = **0**



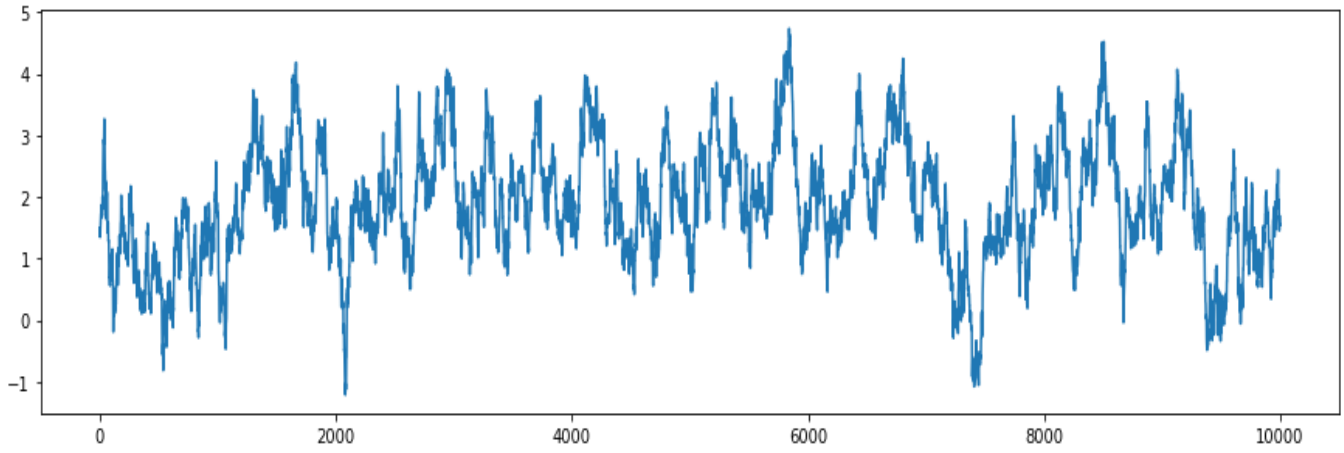
Traceplot for **x1** for **a** = **0.99**



Traceplot for **x2** for **a** = **0**



Traceplot for **x2** for **a** = **0.99**



CONCLUSIONS

Looking at the simulation plots, it is clear that for a = 0, it is a bivariate Gaussian distribution with mean at (1,2). All the points are normally distributed around (1,2). There is no correlation between x1 and x2.

However, for a = 0.99, the graph becomes highly skewed, almost giving a linear relationship between x1 and x2. It shows a very high correlation between x1 and x2.

Looking at the trace plots, they give the expected results(as discussed in this link, provided in the assignment pdf) for a = 0 and a = 0.99, with plots for x1 and x2 for a = 0

showing very low correlation, while for a = 0.99, it shows very high correlation.

PART II

Prey and Predator model using Bayesian Network

PART 1

A simple game of a sheep and two cavemen is simulated. The cavemen follow the algorithm of going to the cell which has the lowest distance from the current position of the sheep. The sheep follows the algorithm that if any of the cavemen are within radius 2 of the sheep, then it moves to the cell where the sum of the nearest corner, the distance of caveman 1 and the distance of caveman 2 is maximized.

To run this code, go into the directory "Code/Part 2", and run the jupyter notebook "Simple Sheep Catching.ipynb".

OBSERVATION AND CONCLUSION

The simulation is run twice, each time a maximum of 100 iterations are allowed. For each simulation, the positions of the sheep and the two cavemen is randomized. It is observed that the **sheep wins** almost all of the time. The results are in accordance with what is expected. The cavemen can only go in 4 directions, and that too only one cell at a time. On the other hand, the sheep can go diagonally, and can choose to move one or two steps. Thus, it is able to escape the cavemen most of the time.

PART 2

The same game is simulated, with the same rules, but caveman 1 is trained first. To train caveman 1, a dictionary of all the possible states of the board, consisting of the positions of both the cavemen and the sheep is created. For each state, the eight possible movements of the board are assigned the number 0 initially.

Then, the simulation is run for 100000 times, allowing 200 moves each time. For each move, the caveman 1 makes a prediction, based on which movement of the sheep has the

highest number attached to it, and moves in the direction that will minimize the distance between itself and the predicted position of the sheep.

Once everyone has made their move, the number assigned to the new position of the sheep, corresponding to the original state, is incremented.

Once caveman 1 has been trained, the same simulation is run as before, this time 5 iterations allowing 200 moves each.

To run this code, go into the directory “Code/Part 2”, and run the jupyter notebook “Bayesian Network Sheep Catching.ipynb”

OBSERVATIONS

It is observed that this time, **caveman 1 wins** almost every time. This is what we expected as the caveman 1 has now learnt the strategy of the sheep.

NOTE

Sometimes the output can be that “Caveman 1/Caveman 2 wins without moving.”. This is to show that when the grid is initialized randomly, it might generate an already winning position for either Cavemen.

PART III

Markov Decision Process

VALUE ITERATION

To perform value iteration for both cases where Magneto is lazy, and the case where Magneto is smart, first all the possible states are generated. For each state, the utility vector is initialized as zero. Then, following the Bellman update, a new utility vector is generated. The only difference in the MDPs of lazy Magneto vs smart Magneto is the probability distribution.

Once the utility vector is generated, simulations are run for both the cases. The simulation ends if Wolverine captures Jean, Magneto captures Jean, or Magneto captures wolverine. 10000 such simulations are run. Wolverine follows the principle that it chooses the neighbor which has the maximum utility.

To run lazy value iteration, go into directory “Code/Part 3”, and run the jupyter notebook “Value Iteration Dumb.ipynb”. To run smart value iteration, go into the directory “Code/Part 3”, and run the jupyter notebook “Value Iteration Smart.ipynb”.

POLICY ITERATION

Initially, a random policy is generated. To make implementation a little easier, for a given state the policy outputs the new position of the wolverine instead of giving directions. Then, this policy is updated in each iteration. All other steps are the same as followed in the implementation of Value Iteration.

Once policy is generated, similar simulation as mentioned before is implemented, the only difference being that a policy is predefined for each position.

To run lazy policy iteration, go into directory “Code/Part 3”, and run the jupyter notebook “Policy Iteration Dumb.ipynb”. To run smart value iteration, go into the directory “Code/Part 3”, and run the jupyter notebook “Policy Iteration Smart.ipynb”.

OBSERVATIONS

Error (for value iteration) = 0.001

Magneto Algorithm	Wolverine captures Jean	Magneto captures Wolverine	Magneto captures Jean
Value Iteration Dumb	72.16%	12.60%	15.24%
Value Iteration Smart	66.66%	24.42%	8.92%
Policy Iteration Dumb	67.33%	14.97%	17.70%
Policy Iteration Smart	59.70%	28.60%	11.70%

CONCLUSION

- We can see that the percentage of Wolverine capturing Jean is better in both Value and Policy Iteration when Magneto is dumb. This is in accordance with our expectations.
- For both dumb Magneto and smart Magneto, Value Iteration performs slightly better than Policy Iteration.
- Magneto captures Wolverine with much better percentage when it plays smart as compared to when it plays dumb.

PART IV

Robot localization using Hidden Markov Model

IMPLEMENTATION

The theory for this part is taken from section 15.3.1(simplified matrix algorithm) and 15.3.2(HMM example: Localization) of Artificial Intelligence: A Modern Approach(Third Edition) by Russell, Norvig.

For 6 possible states, we consider a 6 x 6 matrix T, where:

$$\mathbf{T}_{ij} = P(X_t = j \mid X_{t-1} = i)$$

We also consider a diagonal matrix O, a 6 x 6 matrix, such that:

$$\mathbf{O}_{ii} = P(e_t \mid X_t = i)$$

The forwarding equation then simplifies to:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^T \mathbf{f}_{1:t}$$

This forwarding equation has been implemented.

The matrix **O** is updated as follows:

$$P(\mathbf{E}_t = e_t \mid \mathbf{X}_t = i) = \mathbf{O}_{tii} = (1 - e)^{(4-dit)}(e)^{(dit)}$$

Where e is the error rate, and dit is the discrepancy.

To run this code, go into the directory “Code/Part 4”, and run the notebook “HMM.ipynb”.

It outputs the probabilities of each state after every 10 iterations.

References:

- Artificial Intelligence: A Modern Approach(Third Edition) by Russell, Norvig.
- https://www.statlect.com/fundamentals-of-statistics/Markov-Chain-Monte-Carlo-diagnostics
- Class Lectures