MATHEMATICS IN INTELLIGENT SYSTEMS – 5 19MAT301

SAMPLING

END SEMESTER PROJECT

TEAM - 17

TEAM MEMBERS

HARISH K	CB.EN.U4AIE19029
SANJAY.B	CB.EN.U4AIE19056
SABARISHWARAN G	CB.EN.U4AIE19053
SRI GOKUL PRAZATH	CB.EN.U4AIE19062

श्रद्धावान् लभते ज्ञानम्

AMRITA VISHWA VIDYAPEETHAM UNIVERSITY

ACKNOWLEDGEMENT

I would like to thank all those who have helped me in completing the project under the subject “Mathematics in Intelligent Systems” in our branch computer science engineering -Artificial Intelligence.

First and fore most I would like to thank the faculties in charge – Dr. Soman K.P who have helped us with all the necessary knowledge needed to carry out our project and who was available at every working hour, ready to solve our doubts and improve our project to yield the best possible result.

Table of Contents

Sample

Given a set of variables $X = \{X_1, \ldots., X_n\}$ a sample, denoted by S is an instantiation of all variables:

$$S = (x_1, x_2, \ldots \ldots, x_n)$$

A sample is an **outcome of a random experiment**. When we sample a random variable, we obtain one specific value out of the set of its possible values. That particular value is called a sample. The possible values and the likelihood of each is determined by the random variable's probability distribution.

A sample is **just a part of a population**.

For example, let's say our population was every American, and we wanted to find out how much the average person earns. Time and finances stop you from knocking on every door in America, so you choose to ask 1,000 random people. This one thousand people are our sample.

How to draw a Sample?

Univariate Distribution

A univariate distribution is a probability distribution of only one random variable. This is in contrast to a multivariate distribution, the probability distribution of a random vector (consisting of multiple random variables).

One of the simplest examples of a discrete univariate distribution is the discrete uniform distribution, where all elements of a finite set are equally likely.
It is the probability model for the outcomes of tossing a fair coin, rolling a fair die, etc. The univariate continuous uniform distribution on an interval [a, b] has the property that all sub-intervals of the same length are equally likely.

Example:
Given random variable X having domain {0, 1} and a distribution
P(X) = (0.3, 0.7)

Task:
Generate samples of X from P.

Process:

- First, draw a random number r $\in$ [0, 1]
- If (r < 0.3) then set X=0
- Else set X=1

Multivariate Distribution

The Multivariate normal distribution is a generalization of the one-dimensional (univariate) normal distribution to higher dimensions.

One definition is that a random vector is said to be k-variate normally distributed if every linear combination of its k components has a univariate normal distribution. Its importance derives mainly from the multivariate central limit theorem. The multivariate normal distribution is often used to describe, at least approximately, any set of possible real-valued random variables each of which clusters around a mean value.

- The multivariate normal distribution of a *n*-dimensional random vector $X = \{X_1, \ldots., X_n\}$

- Express the distribution in product form,
  $P(X) = P(X1) \times P(X2|X1) \times \ldots. \times P(X_n|X_1, \ldots, X_{n-1})$

- Sample variables one by one from left to right, along the ordering dictated by the product form.

Bivariate Distribution

A **bivariate distribution** is the probability that a certain event will occur when there are two independent random variables in your scenario.

For example, having two bowls, each filled with two different types of candies, and pulling one candy from each bowl gives you two independent random variables: the two different candies.
Since you're pulling one candy from each bowl at the same time, you have a bivariate distribution when calculating your probability of ending up with particular kinds of candies.

Some of the more common ways to characterize it include:

- Random variables X & Y are bivariate normal if aX + bY has a normal distribution for all a, b ∈ R.
- If a and b are non-zero constants, aX + bY has a normal distribution.
- If X – aY and Y are independent and if Y – bx and X are independent for all a, b (such that ab ≠ 0 or 1), then (X, Y) has a normal distribution.

Categorical Distribution

A categorical distribution is a discrete probability distribution that describes the possible results of a random variable that can take on one of K possible categories, with the probability of each category separately specified.

The K-dimensional categorical distribution is the most general distribution over a K-way event; any other discrete distribution over a size-K sample space is a special case.
The parameters specifying the probabilities of each possible outcome are constrained only by the fact that each must be in the range 0 to 1, and all must sum to 1.

A **categorical distribution** is a discrete probability distribution whose sample space is the set of k individually identified items. It is the generalization of the **Bernoulli distribution** for a categorical random variable.

In one formulation of the distribution, the sample space is taken to be a finite sequence of integers. The exact integers used as labels are unimportant;
they might be {0, 1, ..., k − 1} or {1, 2, ..., k} or any other arbitrary set of values.

Bernoulli Distribution

The Bernoulli distribution, named after Swiss mathematician Jacob Bernoulli, is the discrete probability distribution of a random variable which takes the value 1 with probability **p** and the value 0 with probability **q = 1-p**.

Generally, it can be thought of as a model for the set of possible outcomes of any single experiment that asks a yes–no question. Such questions lead to outcomes that are Boolean-valued:
A single bit whose value is **success/yes/true/one** with probability **p** and **failure/no/false/zero** with probability **q**.

Example:

It can be used to represent a (possibly biased) coin toss where 1 and 0 would represent "heads" and "tails" (or vice versa), respectively, and p would be the probability of the coin landing on heads or tails, respectively.
In particular, unfair coins would have $p \neq \frac{1}{2}$
If X is a random variable with this distribution, then:

$$\Pr(X = 1) = p = 1 - \Pr(X = 0) = 1 - q$$

The probability mass function f of this distribution, over possible outcomes k, is

$$f(k;p) = \begin{cases} p & if\ k = 1, \\ q = 1 - p & if\ k = 0 \end{cases}$$

Sampling

A sample is a subset of individuals from a larger population. Sampling means selecting the group that you will actually collect data from in your research. For example, if you are researching the opinions of students in your university, you could survey a sample of 100 students.

In statistics, sampling allows you to test a hypothesis about the characteristics of a population.

Sampling is a process used in statistical analysis in which a predetermined number of observations are taken from a larger population.
The methodology used to sample from a larger population depends on the type of analysis being performed, but it may include simple random sampling or systematic sampling.

To draw valid conclusions from your results, you have to carefully decide how you will select a sample that is representative of the group as a whole. There are two types of sampling methods:

- **Probability sampling** involves random selection, allowing you to make strong statistical inferences about the whole group.
- **Non-probability sampling** involves non-random selection based on convenience or other criteria, allowing you to easily collect data.

Probability Sampling methods

Probability sampling means that every member of the population has a chance of being selected. It is mainly used in quantitative research. If you want to produce results that are representative of the whole population, probability sampling techniques are the most valid choice.

There are four main types of probability sample.

- Simple random sampling
- Systematic sampling
- Stratified sampling
- Cluster sampling

Simple random sampling:

In a simple random sample, every member of the population has an equal chance of being selected. Your sampling frame should include the whole population.

To conduct this type of sampling, you can use tools like random number generators or other techniques that are based entirely on chance.

Example:
You want to select a simple random sample of 100 employees of Company X. You assign a number to every employee in the company database from 1 to 1000, and use a random number generator to select 100 numbers.

Systematic sampling

Systematic sampling is similar to simple random sampling, but it is usually slightly easier to conduct. Every member of the population is listed with a number, but instead of randomly generating numbers, individuals are chosen at regular intervals.

Stratified sampling

Stratified sampling involves dividing the population into subpopulations that may differ in important ways. It allows you draw more precise conclusions by ensuring that every subgroup is properly represented in the sample.

To use this sampling method, you divide the population into subgroups (called strata) based on the relevant characteristic (e.g. gender, age range, income bracket, job role).

Cluster sampling

Cluster sampling also involves dividing the population into subgroups, but each subgroup should have similar characteristics to the whole sample. Instead of sampling individuals from each subgroup, you randomly select entire subgroups.

If it is practically possible, you might include every individual from each sampled cluster. If the clusters themselves are large, you can also sample individuals from within each cluster using one of the techniques above. This is called multistage sampling.

Non-Probability Sampling methods

In a non-probability sample, individuals are selected based on non-random criteria, and not every individual has a chance of being included.

This type of sample is easier and cheaper to access, but it has a higher risk of sampling bias. That means the inferences you can make about the population are weaker than with probability samples, and your conclusions may be more limited. If you use a non-probability sample, you should still aim to make it as representative of the population as possible.

There are four main types of non-probability sample.

- Convenience sampling
- Voluntary sampling
- Purposive sampling
- Snowball sampling

Convenience sampling

A convenience sample simply includes the individuals who happen to be most accessible to the researcher.

This is an easy and inexpensive way to gather initial data, but there is no way to tell if the sample is representative of the population, so it can't produce generalizable results.

Example:
You are researching opinions about student support services in your university, so after each of your classes, you ask your fellow students to complete a survey on the topic. This is a convenient way to gather data, but as you only surveyed students taking the same classes as you at the same level, the sample is not representative of all the students at your university.

Voluntary response sampling

Similar to a convenience sample, a voluntary response sample is mainly based on ease of access. Instead of the researcher choosing participants and directly contacting them, people volunteer themselves (for example: by responding to a public online survey).

Voluntary response samples are always at least somewhat biased, as some people will inherently be more likely to volunteer than others.

Purposive sampling

This type of sampling, also known as judgement sampling, involves the researcher using their expertise to select a sample that is most useful to the purposes of the research.

It is often used in qualitative research, where the researcher wants to gain detailed knowledge about a specific phenomenon rather than make statistical inferences, or where the population is very small and specific.

Snowball sampling

If the population is hard to access, snowball sampling can be used to recruit participants via other participants. The number of people you have access to “snowballs” as you get in contact with more people.

Monte Carlo Method

Monte Carlo Method is a mathematical technique, which is used to estimate the possible outcomes of an uncertain event.

History

The Monte Carlo Method was invented by John von Neumann and Stanislaw Ulam during World War II to improve decision making under uncertain conditions. It was named after a well-known casino town, called Monaco, since the element of chance is core to the modeling approach, similar to a game of roulette.

Process

Regardless of what tool you use, Monte Carlo techniques involves three basic steps:

1. Set up the **predictive model**, identifying both the dependent variable to be predicted and the independent variables (also known as the input, risk or predictor variables) that will drive the prediction.

2. Specify probability distributions of the independent variables. Use historical data and/or the analyst's subjective judgment to define a range of likely values and assign probability weights for each.

3. Run simulations repeatedly, generating random values of the independent variables. Do this until enough results are gathered to make up a representative sample of the near infinite number of possible combinations.

Monte Carlo Estimate

An estimator is a function of the samples. It produces an estimate of the unknown parameter of the sampling distribution.

Given IID samples (Independent and identically distributed random variables) $S^1,\ S^2,\ \ldots\ S^T$ drawn from P, the Monte Carlo estimate of $E_P[g(x)]$ is given by:

$$g = \frac{1}{T}\sum_{t=1}^{T} g(S^t)$$

Example:

Given:
A distribution P(X) = (0.3, 0.7).
g(X) = 40 if X equals 0
= 50 if X equals 1
Estimate $E_P[g(x)] = (40\ \times\ 0.3 + 50\ \times 0.7\ ) = 47$
Generate k samples from P: 0,1,1,1,0,1,1,0,1,0

$$g = \frac{40\ \times \#samples(X = 0) + 50\ \times \#samples(X = 1)}{\#samples}$$

$$= \frac{40\ \times 4 + 50\ \times 6}{10} = 46$$

Markov chain Monte Carlo (MCMC)

In statistics, Markov chain Monte Carlo (MCMC) methods comprise a class of algorithms for sampling from a probability distribution. By constructing a Markov chain that has the desired distribution as its equilibrium distribution, one can obtain a sample of the desired distribution by recording states from the chain. The more steps are included, the more closely the distribution of the sample matches the actual desired distribution. Various algorithms exist for constructing chains, including the Metropolis–Hastings algorithm.

MCMC is a class of techniques for sampling from a probability distribution and can be used to estimate the distribution of parameters given a set of observations.

Bayesian Network

A Bayesian network (also known as a Bayes network, Bayes net, belief network, or decision network) is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). Bayesian networks are ideal for taking an event that occurred and predicting the likelihood that any one of several possible known causes was the contributing factor.

For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

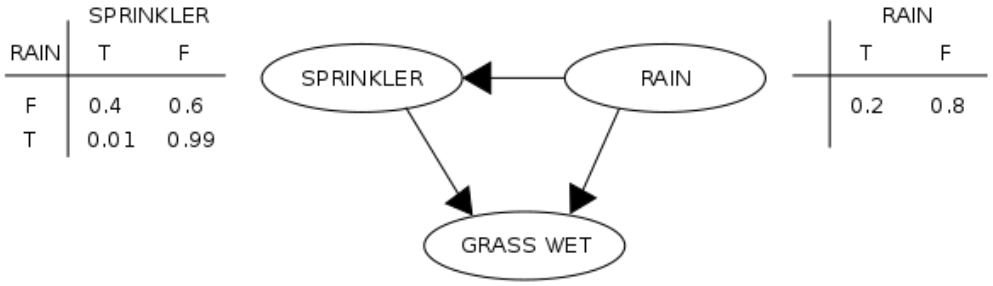
Graphical model

- Formally, Bayesian networks are directed acyclic graphs (DAGs) whose nodes represent variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses.
- Edges represent conditional dependencies; nodes that are not connected represent variables that are conditionally independent of each other.

- Each node is associated with a probability function that takes, as input, a particular set of values for the node's parent variables, and gives (as output) the probability of the variable represented by the node.

Example

Two events can cause grass to be wet: an active sprinkler or rain. Rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler usually is not active). This situation can be modeled with a Bayesian network (given below). Each variable has two possible values, T (for true) and F (for false).



RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99

RAIN T	RAIN F
0.2	0.8

SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

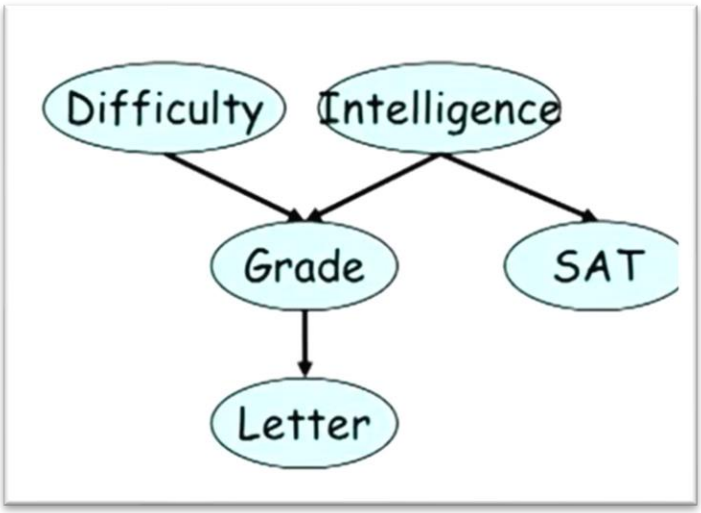
The joint probability function is, by the chain rule of probability,

$$\Pr(G, S, R) = \Pr(G \mid S, R) \Pr(S|R) \, Pr(R)$$

where G = "Grass wet (true/false)", S = "Sprinkler turned on (true/false)", and R = "Raining (true/false)".

Example – Bayesian Network

For the implementation of the sampling methods using Bayesian Network, we have used the following Bayesian Network,



D1	0.6
D2	0.4

I1	0.7
I2	0.3

	G1	**G2**	**G3**
I1D1	0.3	0.4	0.3
I2D1	0.05	0.25	0.7
I1D2	0.9	0.08	0.02
I2D2	0.5	0.3	0.2

	L1	**L2**
G1	0.1	0.9
G2	0.4	0.6
G3	0.99	0.01

	S1	**S2**
I1	0.95	0.05
I2	0.2	0.8

Prior Sampling

Prior Sampling is a sampling technique that uses a Bayes' Net to generate samples.

Process

- For I = 1, 2, …., n, sample xi from P (Xi | Parents (Xi))
- Return (x1, x2, ……, xn)

- This process generates samples with probability:

$$S_{PS}(x1, \ldots., xn) = \prod_i P(xi \mid parents(Xi)) = P(x1, x2, \ldots. xn)$$

- Let the number of samples of an event be $N_{PS}(x1, x2, \ldots.., xn)$
- Estimate from N samples is $Q_N(x1, \ldots, xn) = N_{PS}(x1, \ldots, xn)/N$
- Then,

$$lim_{N \to \infty} Q_N(x1, \ldots, xn) = lim_{N \to \infty} \frac{N_{PS}(x1, \ldots, xn)}{N}$$
$$= S_{PS}(x1, \ldots, xn)$$
$$= P(x1, \ldots, xn)$$

I.e., the sampling procedure is consistent

Implementation

MATLAB Code

getVal function

getVal function is generally used to get the value of the node used in the Bayesian Network from the generated random sample.

We will be having 2 input arguments, where,

- The **P** is used to decide the row with the help of its parent node
- **r** is the randomly generated value between 0 and 1

Output: It gives the value of the node

```
function V=getVal(P,r)
   V=-1;
   %disp(P)
   if(sum(P)==1)
       C=zeros(length(P));
       C(1)=P(1);
       for i=2:length(C)
           C(i)=P(i)+C(i-1);
       end
       for i=1:length(C)
           if(r<C(i))
               V=i;
               break
           end
       end
   end
end
```

Prior Sampling

```
clc;
clear all;
close all;
%Probablity Arrays
D=[0.6;0.4];
I=[0.7;0.3 ];
P=zeros(48,1);
%Probablity Matrices
cpdG=[0.3 0.4 0.3; 0.05 0.25 0.7; 0.9 0.08 0.02; 0.5 0.3
0.2];
cpdL=[0.1 0.9;0.4 0.6;0.99 0.01];
cpdS=[0.95 0.05; 0.2 0.8];
nS=5*10000;%No.of.nodes*no.of.samples
R=rand(1,nS);%generating random values
Sample=[];
for i=1:5:nS
    tmp(1)=getVal(D,R(i));%finding the value with the help
of random numbers generated
    tmp(2)=getVal(I,R(i+1));
    tmp(3)=getVal(cpdG((tmp(2)-1)*2+tmp(1),:),R(i+2));
    tmp(4)=getVal(cpdS(tmp(2),:),R(i+3));
    tmp(5)=getVal(cpdL(tmp(3),:),R(i+4));
    T(1)="D"+tmp(1);
    T(2)="I"+tmp(2);
    T(3)="G"+tmp(3);
    T(4)="S"+tmp(4);
    T(5)="L"+tmp(5);
    Sample=[Sample;T];%insert the samples
end
Sample(1:10,:)
sz = size(Sample)
I1 = length(find(Sample=='I1'))/length(Sample)
```

From this we are finding the probability of I1 from the Bayesian Network provided.

Output

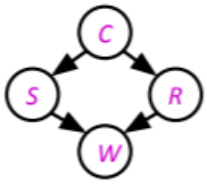
```
ans =

  10×5 string array

    "D1"    "I2"    "G1"    "S2"    "L2"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I2"    "G3"    "S2"    "L1"
    "D1"    "I1"    "G1"    "S1"    "L2"
    "D1"    "I1"    "G3"    "S1"    "L1"
    "D1"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I2"    "G1"    "S2"    "L2"
    "D1"    "I1"    "G1"    "S1"    "L2"
    "D2"    "I2"    "G2"    "S2"    "L1"
    "D1"    "I1"    "G1"    "S1"    "L2"


sz =

       10000           5


I1 =

    0.7025
```

Rejection Sampling

Rejection sampling is a Monte Carlo algorithm to sample data from a sophisticated ("difficult to sample from") distribution with the help of a proxy distribution.

Rejection sampling is a computational technique whose aim is generating random numbers from a target probability distribution f(x). It is related to the general field of Monte Carlo methods, whose core is generating repeated random sampling to make numerical estimation of unknown parameters.

A simple application of prior sampling for estimating conditional probabilities.



- Let's say we want $P(C \mid r, w) = \propto P(C, r, w)$
- For these counts, samples with '-r' or '–w' **are not relevant**
- So, we count the **C** outcomes for samples with **r, w** and reject all other samples
- This is called **rejection sampling**

Implementation

MATLAB Code

Rejection Sampling

```
clc;
clear all;
close all;
%Probablity Arrays
D=[0.6;0.4];
I=[0.7;0.3 ];
P=zeros(48,1);
%Probablity Matrices
cpdG=[0.3 0.4 0.3; 0.05 0.25 0.7; 0.9 0.08 0.02; 0.5 0.3
0.2];
cpdL=[0.1 0.9;0.4 0.6;0.99 0.01];
cpdS=[0.95 0.05; 0.2 0.8];
nS=5*10000;%No.of.nodes*no.of.samples
R=rand(1,nS);%generating random values
Sample=[];
for i=1:5:nS
    tmp(1)=getVal(D,R(i));%finding the value with the help
of random numbers generated
    tmp(2)=getVal(I,R(i+1));
    tmp(3)=getVal(cpdG((tmp(2)-1)*2+tmp(1),:),R(i+2));
    tmp(4)=getVal(cpdS(tmp(2),:),R(i+3));
    tmp(5)=getVal(cpdL(tmp(3),:),R(i+4));
    T(1)="D"+tmp(1);
    T(2)="I"+tmp(2);
    T(3)="G"+tmp(3);
    T(4)="S"+tmp(4);
    T(5)="L"+tmp(5);
    if(T(3)=="G2")%Rejecting the unwanted values
    Sample=[Sample;T];%insert the samples
    end
end
Sample(1:10,:)
sz = size(Sample)
S1=length(find(Sample=='S1'))/length(Sample)
```

From this we are finding the P (S1 | G2) [probability of S1 given G2] from the Bayesian Network provided.

Output

```
ans =

  10×5 string array

    "D1"    "I1"    "G2"    "S1"    "L1"
    "D1"    "I1"    "G2"    "S1"    "L1"
    "D2"    "I1"    "G2"    "S1"    "L2"
    "D1"    "I1"    "G2"    "S1"    "L2"
    "D1"    "I1"    "G2"    "S1"    "L1"
    "D1"    "I1"    "G2"    "S1"    "L2"
    "D1"    "I1"    "G2"    "S1"    "L1"
    "D1"    "I2"    "G2"    "S1"    "L2"
    "D2"    "I1"    "G2"    "S1"    "L1"
    "D2"    "I2"    "G2"    "S2"    "L1"


sz =

        2945           5


S1 =

    0.8255
```

Likelihood Weighting

- Likelihood weighting is a form of importance sampling where the variables are sampled in the order defined by a belief network, and evidence is used to update the weights.

- The weights reflect the probability that a sample would not be rejected.

Why Likelihood Weighting

Problem with rejection sampling:

- If evidence is unlikely, rejects lots of samples
- Evidence not exploited as you sample
- Consider P (Shape | Colour = blue)

Solution:

Idea: **fix evidence variables, sample the rest**

- **Problem:** Sample distribution not consistent!
- **Solution:** Weight each sample by probability of evidence variables given parents

Algorithm

- Input: evidence $e_1, \ldots., e_k$
- w = 1.0
- for I = 1, 2, …., n
  - If Xi is an evidence variable
    - xi = observed value for Xi
    - Set w = w * P (xi | parents (Xi))
  - else
    - Sample xi from P (Xi | parents (Xi))
- Return (x1, x2, ……, xn), w

Pros & Cons of Likelihood Weighting

Likelihood weighting is **good**

- All samples are used
- The values of downstream variables are influenced by upstream evidence

Likelihood weighting still has **weaknesses**,

- The values of upstream variables are unaffected by downstream evidence.
- With high probability, one lucky sample will have much larger weight than the others, dominating the result.

Implementation

MATLAB Code

Likelihood Weighting

```
clc;
clear all;
close all;
%Probablity Arrays
D=[0.6;0.4];
I=[0.7;0.3 ];
P=zeros(48,1);
%Probablity Matrices
cpdG=[0.3 0.4 0.3; 0.05 0.25 0.7; 0.9 0.08 0.02; 0.5 0.3 0.2];
cpdL=[0.1 0.9;0.4 0.6;0.99 0.01];
cpdS=[0.95 0.05; 0.2 0.8];
nS=5*10000;%No.of.nodes*no.of.samples
R=rand(1,nS);%generating random values
Sample=[];
weight=[];
for i=1:5:nS
    tmp(1)=getVal(D,R(i));%finding the value with the help of
random numbers generated
    tmp(2)=1;%defining the values manually
    tmp(3)=getVal(cpdG((tmp(2)-1)*2+tmp(1),:),R(i+2));
    tmp(4)=1;
    tmp(5)=getVal(cpdL(tmp(3),:),R(i+4));
    T(1)="D"+tmp(1);
```

```
    T(2)="I"+tmp(2);
    T(3)="G"+tmp(3);
    T(4)="S"+tmp(4);
    T(5)="L"+tmp(5);
    w=D(tmp(1))*cpdG(tmp(1),tmp(2))*cpdL(tmp(3));%assigniing
weight for each sample
    Sample=[Sample;T];%insert the samples
end
Sample(1:10,:)
sz = size(Sample)
%P(L=L1|I=I1,S=S1
L1=length(find(Sample=='L1'))/length(Sample)
```

From this we are finding the P(L1 | I1 and S1) [probability of L1 given I1 and S1] from the Bayesian Network provided.

Output

```
ans =

  10×5 string array

    "D1"    "I1"    "G3"    "S1"    "L1"
    "D1"    "I1"    "G2"    "S1"    "L1"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D1"    "I1"    "G1"    "S1"    "L2"
    "D1"    "I1"    "G2"    "S1"    "L1"
    "D1"    "I1"    "G2"    "S1"    "L2"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D1"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D1"    "I1"    "G1"    "S1"    "L2"


sz =

       10000           5


L1 =

    0.6197
```

Gibbs Sampling

Gibbs sampling is a Markov Chain Monte Carlo algorithm for high-dimensional data such as image processing and micro arrays.
It is called Monte Carlo because it draws samples from specified probability distributions; the Markov chain comes from the fact that each sample is dependent on the previous sample.

Gibbs sampling is an efficient way of reducing a multi-dimensional problem to a lower-dimensional problem. The entire parameter vector is subdivided into smaller sub-vectors.

Process

$initialize\ Y^0, X^0$

$for\ j = 1, 2, 3, \dots do$
$\quad sample\ X^j \sim p(X \mid Y^{j-1})$
$\quad sample\ Y^j \sim p(Y \mid X^j)$
$end\ for$

Algorithm

- ❑ States are complete assignments to all variables.
- ❑ Evidence variables remain fixed, other variables change.
- ❑ To generate the next state, pick a variable and sample a value for it conditioned on all the other variables:

$$X_i' \sim P(Xi \mid x1, \dots, x_{i-1}, x_{i+1}, \dots., xn)$$

Will tend to move towards states of higher probability, but can go down too.

- In a Bayes net,

$$P(Xi \mid x1, \dots, x_{i-1}, x_{i+1}, \dots., xn) = P(Xi \mid markov\_blanket(Xi))$$

Gibbs sampling is consistent

Implementation

MATLAB Code
Gibbs sampling

```
clc;
clear all;
close all;
%Probablity Arrays
D=[0.6;0.4];
I=[0.7;0.3 ];
P=zeros(48,1);
%Probablity Matrices
cpdG=[0.3 0.4 0.3; 0.05 0.25 0.7; 0.9 0.08 0.02; 0.5 0.3 0.2];
cpdL=[0.1 0.9;0.4 0.6;0.99 0.01];
cpdS=[0.95 0.05; 0.2 0.8];
nS=5*10000;%No.of.nodes*no.of.samples
R=rand(1,nS);%generating random values
Sample=[];
tmp(1)=getVal(D,R(1));%finding the value with the help of random
numbers generated
tmp(2)=getVal(I,R(1+1));
tmp(3)=getVal(cpdG((tmp(2)-1)*2+tmp(1),:),R(1+2));
tmp(4)=getVal(cpdS(tmp(2),:),R(1+3));
tmp(5)=getVal(cpdL(tmp(3),:),R(1+4));
T(1)="D"+tmp(1);
T(2)="I"+tmp(2);
T(3)="G"+tmp(3);
T(4)="S"+tmp(4);
T(5)="L"+tmp(5);
Sample=[Sample;T];
for i=6:5:nS
    tmp(3)=getVal(cpdG((tmp(2)-1)*2+tmp(1),:),R(i+2));
    tmp(5)=getVal(cpdL(tmp(3),:),R(i+4));
    T(1)="D"+tmp(1);
    T(2)="I"+tmp(2);
    T(3)="G"+tmp(3);
    T(4)="S"+tmp(4);
    T(5)="L"+tmp(5);
    Sample=[Sample;T];%insert the samples
end
Sample(1:10,:)
sz = size(Sample)
L2=length(find(Sample=='L2'))/length(Sample)
```

From this we are finding the P(L2, D, I, S) [probability of L2 by fixing “D”, “I”, “S” values] from the Bayesian Network provided.

Output

```
ans =

  10×5 string array

    "D2"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I1"    "G2"    "S1"    "L2"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I1"    "G2"    "S1"    "L2"
    "D2"    "I1"    "G3"    "S1"    "L1"
    "D2"    "I1"    "G2"    "S1"    "L1"
    "D2"    "I1"    "G3"    "S1"    "L1"


sz =

       10000           5


L2 =

    0.1936
```

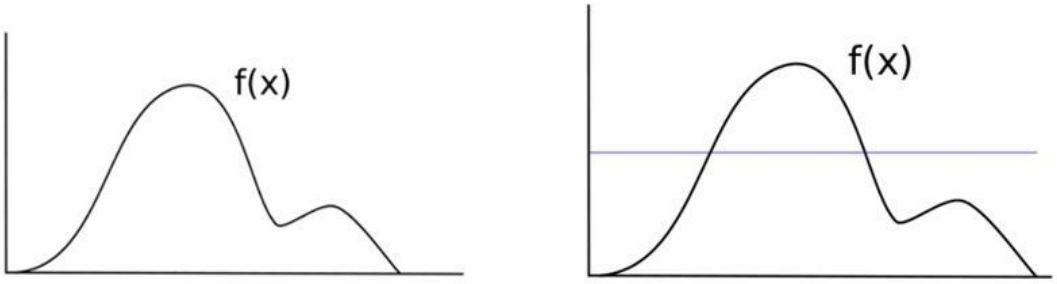
Slice Sampling

Slice sampling is a Markov Chain Monte Carlo (MCMC) method for sampling from a probability distribution.

The basic idea is that any distribution can be sampled from by selecting uniformly spaced points under a probability distribution curve using a MCMC algorithm; multivariate distributions can be sampled by treating each variable separately.

Basic Steps of Slice Sampling:

- Choose a starting value x0 for which f(x0) > 0.
- Sample a y value uniformly between 0 and f(x0).
- Draw a horizontal line across the curve at this y position.
- Sample a point (x, y) from the line segments within the curve.
- Repeat from step 2 using the new x value.



Advantages:

- It can easily draw the posterior samples from any prior distribution as long as these distributions have a reasonable value range of parameters.
- It's easier to implement than other, similar MCMC methods like Gibbs sampling
- It needs less tuning than the Metropolis–Hastings algorithm
- It adaptively chooses the magnitude of changes made in prior steps, It is more efficient than basic Metropolis algorithms.

Disadvantages:

- It can be a complicated procedure to employ.
- It is not efficient in case of multivariate distribution

Implementation

MATLAB Code

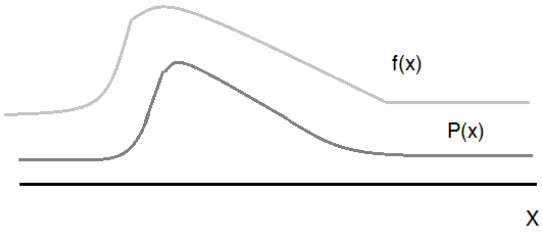
```
clc;
close all;

smpl = slicesamples(5,10); % No of Samples in each slices &
Defining No of slices

x = -3:0.1:3;
y = normpdf(x,0,1);
figure;
plot(x,y)

function p = func(x) % Defining the function
    p = normpdf(x,0,1);
end

function smpl = slicesamples(size,number)

smpl = []; % Initialising an
w = 0.01*rand(1,size); % Defining the width parameter
u = 0.1; % Choosing the initial value
x = zeros(1,size);

for i = 1:number
smpl(i,:) = x; % first we are assigning all values as 0
p = func(x);
up = rand(1,1)*p;
r = rand(1,1); % generating random number between 0 and 1
xl = x - r*w; % finding left extreme of slice
xr = x + (1-r)*w; % finding right extreme of slice

while (func(xl) > u) % finding the correct left extreme
    xl = xl - w;
end
while (func( xr) > u) % finding the correct right extreme
```

```
    xr = xr + w;
end

xp = rand(1,1)*(xr-xl) + xl;
while(func(xp) < up)
    xp = rand(1,1)*(xr-xl) + xl;
    if func(xp) > up
        break;
    else
        if xp>x
            xr = xp;
        else
            xl = xp;
        end
    end
end
x = xp;
u = up;
end
end
```

Metropolis Hastings

The Metropolis–Hastings algorithm is a **Markov chain Monte Carlo** (MCMC) method for obtaining a sequence of random samples from a probability distribution from which direct sampling is difficult. This sequence can be used to approximate the distribution (Example: to generate a histogram) or to compute an integral (Example: an expected value).

Metropolis–Hastings and other MCMC algorithms are generally used for sampling from multi-dimensional distributions, especially when the number of dimensions is high. For single-dimensional distributions, there are usually other methods (Example: adaptive rejection sampling) that can directly return independent samples from the distribution, and these are free from the problem of autocorrelated samples that is inherent in MCMC methods.



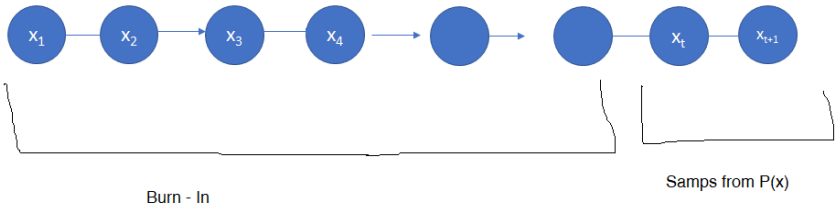
We have to sample from the distribution P(x) which we don't have any idea about. (we don't know the exact form of P(x)

We want to generate Random variate from p(x),
What we are given is f(x).

But $$P(x) = \frac{f(x)}{NC}$$

where, NC – Normalising Constant
Random variate values are assumed to be states in a Markov chain
There are infinite states. We want to generate as many as we want.
But its histogram of state values should follow distribution P(x)

MCMC Approach



The Objective here is to use only f(x) to get sample from P(x), so we have to consider a Markov chain.
The initial samples which we get from the Markov chain may not follow the target distribution P(x) known as **Burn – In Period** and hence we they are not required.
But at some point we get a state X_t such that everything from there on is going to be treated as a sample from P(x).
Sample the next candidate based on its previous value. (X_{t+1} from X_t)

- **Sample :-**

$$g(X_{t+1}|X_t)$$
$$g(X_{t+1}|X_t) = N(X_t\ ,\ \sigma^2)$$

- **Acceptance Rule :-** $A(X_t \rightarrow X_{t+1})$

$$p(a)T(a \rightarrow b) = p(b)T(b \rightarrow a)$$

$$\frac{f(a)}{NC}g(b|a)A(a \rightarrow b) = \frac{f(b)}{NC}\ g(a|b)A(b \rightarrow a)$$

$$\frac{A(a \rightarrow b)}{A(b \rightarrow a)} = \frac{f(b)}{f(a)} \times \frac{g(a|b)}{g(b|a)}$$

rf rg

Let ‘a’ be the current point (x=a)
Let ‘b’ be the proposed point(x=b)

$rfrg < 1$	$r_f r_g \geq 1$
$A(a \to b) = r_f r_g$	$A(a \to b) = 1$
$A(b \to a) = 1$	$A(b \to a) = \frac{1}{r_f r_g}$

$$So,\ A(a \to b) = Min(1,\ r_f r_g)$$

Metropolis - Hastings Algorithm

Algorithm:

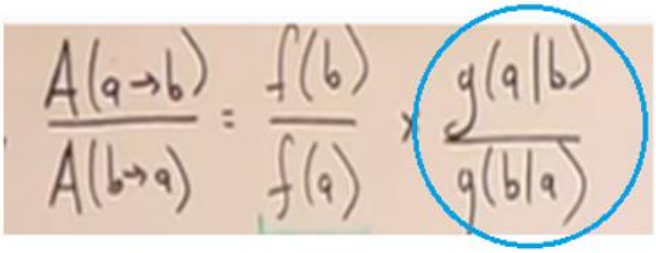
- $Intitialize\ x^0, sigma$
- $Re\ p\ eat$

$Generate\ r\ from\ U(0,1)$
$Assume\ current\ state\ x = x^i$
$Generate\ sample\ x^* from\ \ g(x^*|x^i)$
$x^* = normrnd(x = x^i, sigma)$
$Compute\ \ g(x^*|x) \leftarrow normpdf(x^*, mu = x^i, \sigma = sigma)$
$Compute\ \ g(x|x^*) \leftarrow normpdf(x^i, mu = x^*, \sigma = sigma)$
$Compute\ f(x^*), f(x^i)$

$$Compute\ P(x, x^*) = min\left[1, \frac{f(x^*)g(x|x^*)}{f(x)g(x^*|x)}\right]$$

$If\ \ r \leq P(x, x^*)$
$Accept\ \ x^*,\ \ x^{i+1} = x^*,\ i = i + 1$
$else\ \ Reject\ \ x^*$

Metropolis Algorithm :-

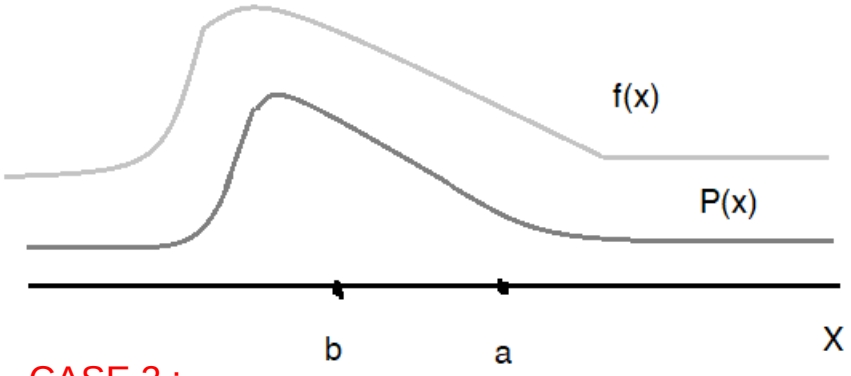


$$= 1$$

$$A(a \to b) = Min\left(1, \frac{f(a)}{f(b)}\right) = Min\left(1, \frac{p(b)}{p(a)}\right)$$

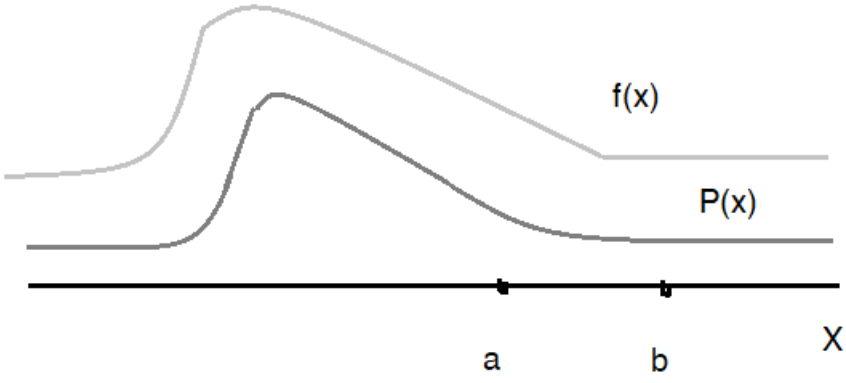
CASE 1 :-

$p(b) > p(a) => A(a \to b) = 1$



CASE 2 :-

$$p(b) < p(a) => A(a \to b) = \frac{p(b)}{p(a)}$$



Implementation

Python Code

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

def normal_dt(x,mu,sigma):
    numerator = np.exp((-(x-mu)**2)/(2*sigma**2))
    denominator = sigma * np.sqrt(2*np.pi)
    return numerator/denominator

def random_coin(p):
    unif = random.uniform(0,1)
    if unif>=p:
        return False
    else:
        return True

def gauss_mcmc(hops,mu,sigma):
    states = []
    burn_in = int(hops*0.2)
    current = random.uniform(-5*sigma+mu,5*sigma+mu)
    for i in range(hops):
        states.append(current)
        movement = random.uniform(-5*sigma+mu,5*sigma+mu)

        curr_prob = normal_dt(x=current,mu=mu,sigma=sigma)
        move_prob = normal_dt(x=movement,mu=mu,sigma=sigma)

        acceptance = min(move_prob/curr_prob,1)
        if random_coin(acceptance):
            current = movement
    return states[burn_in:]

lines = np.linspace(-3,3,1000)
normal_curve = [normal_dt(l,mu=0,sigma=1) for l in lines]
dist = gauss_mcmc(100_000,mu=0,sigma=1)
plt.hist(dist,density = True,bins=20)
plt.plot(lines,normal_curve)
```

Output

```
Out[13]: [<matplotlib.lines.Line2D at 0x204fbde6490>]
```