## Contents

1 Sampling from Random Variables 4  
1.1 Standard distributions 4  
1.2 Sampling from non-standard distributions 7  
1.2.1 Inverse transform sampling with discrete variables 8  
1.2.2 Inverse transform sampling with continuous variables 11  
1.2.3 Rejection sampling 12  

2 Markov Chain Monte Carlo 16  
2.1 Monte Carlo integration 16  
2.2 Markov Chains 17  
2.3 Putting it together: Markov chain Monte Carlo 19  
2.4 Metropolis Sampling 19  
2.5 Metropolis-Hastings Sampling 23  
2.6 Metropolis-Hastings for Multivariate Distributions 26  
2.6.1 Blockwise updating 27  
2.6.2 Componentwise updating 29  
2.7 Gibbs Sampling 36  

3 Graphical Models 39  
3.1 A Short Review of Probability Theory 39  
3.2 The Burglar Alarm Example 41  
3.2.1 Conditional probability tables 42  
3.2.2 Explaining away 44  
3.2.3 Joint distributions and independence relationships 45  
3.3 Graphical Model Notation 47  
3.3.1 Example: Consensus Modeling with Gaussian variables 47  

4 Approximate Inference in Graphical Models 50  
4.1 Prior predictive distributions 50  
4.2 Posterior distributions 53  
4.2.1 Rejection Sampling 53  
4.2.2 MCMC Sampling 54
4.2.3 Example: Posterior Inference for the consensus model with normally distributed variables ........................................... 56
Note to Students

Exercises

This course book contains a number of exercises in which you are asked to simulate Matlab code, produce new code, as well as produce graphical illustrations and answers to questions. The exercises marked with ** are optional exercises that can be skipped when time is limited.

Organizing answers to exercises

It is helpful to maintain a document that organizes all the material related to the exercises. Matlab can facilitate part of this organization using the “publish” option. For example, if you have a Matlab script that produces a figure, you can publish the code as well as the figure produced by the code to a single external document. You can find the publishing option in the Matlab editor under the file menu. You can also use the publish function directly in the command window. You can change the publish configuration (look under the file menu of the editor window) to produce pdfs, Word documents and a number of file formats.

Matlab documentation

It will probably happen many times that you will need to find the name of a Matlab function or a description of the input and output variables for a given Matlab function. It is strongly recommended to have always have the Matlab documentation running in a separate window for quick consultation. You can access the Matlab documentation by typing doc in the command window. For specific help on a given matlab function, for example the function fprintf, you can type doc fprintf to get a help screen in the matlab documentation window or help fprintf to get a description in the matlab command window.
Chapter 1

Sampling from Random Variables

Probabilistic models proposed by researchers are often too complicated for analytic approaches. Increasingly, researchers rely on computational, numerical-based methods when dealing with complex probabilistic models. By using a computational approach, the researcher is freed from making unrealistic assumptions required for some analytic techniques (e.g., such as normality and independence).

The key to most approximation approaches is the ability to sample from distributions. Sampling is needed to predict how a particular model will behave under some particular set of circumstances, and to find appropriate values for the latent variables ("parameters") when applying models to experimental data. Most computational sampling approaches turn the problem of sampling from complex distributions into subproblems involving simple sampling distributions. In this chapter, we will illustrate two sampling approaches: the inverse transformation method and rejection sampling. These approaches are appropriate mostly for the univariate case where we are dealing with single-valued outcomes. In the next chapter, we discuss Markov chain Monte Carlo approaches that operate efficiently with multivariate distributions.

1.1 Standard distributions

Some distributions are used so often, they are becoming part of a standard set of distributions supported software packages. The Matlab statistics toolbox supports a large number of probability distributions. Using Matlab, it becomes quite easy to calculate the probability density, cumulative density of these distributions, as well as generate random values from these distributions. Table 1.1 lists some of the standard distributions supported by Matlab. The Matlab documentation lists many more distributions that can be simulated with Matlab. Using online resources, it is often easy to find support for a number of other common distributions.

To illustrate how we can use some of these functions, Listing 1.1 shows Matlab code that visualizes the Normal($\mu, \sigma$) distribution where $\mu = 100$ and $\sigma = 15$. To make things concrete, imagine that this distribution represents the observed variability of IQ coefficients in some
Table 1.1: Examples of Matlab functions for evaluating probability density, cumulative density and drawing random numbers

<table>
<thead>
<tr>
<th>Distribution</th>
<th>PDF</th>
<th>CDF</th>
<th>Random Number Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>normpdf</td>
<td>normcdf</td>
<td>norm</td>
</tr>
<tr>
<td>Uniform (continuous)</td>
<td>unipdf</td>
<td>unifcdf</td>
<td>unifrnd</td>
</tr>
<tr>
<td>Beta</td>
<td>betapdf</td>
<td>betacdf</td>
<td>betarnd</td>
</tr>
<tr>
<td>Exponential</td>
<td>exppdf</td>
<td>expcdf</td>
<td>exprnd</td>
</tr>
<tr>
<td>Uniform (discrete)</td>
<td>unipdf</td>
<td>unidcdf</td>
<td>unidrnd</td>
</tr>
<tr>
<td>Binomial</td>
<td>binopdf</td>
<td>binocdf</td>
<td>binornd</td>
</tr>
<tr>
<td>Multinomial</td>
<td>mnpdf</td>
<td>mnrnd</td>
<td></td>
</tr>
<tr>
<td>Poisson</td>
<td>poisspdf</td>
<td>poisscdf</td>
<td>poissrnd</td>
</tr>
</tbody>
</table>

Figure 1.1: Illustration of the Normal($\mu$, $\sigma$) distribution where $\mu = 100$ and $\sigma = 15$.

population. The code shows how to display the probability density and the cumulative density. It also shows how to draw random values from this distribution and how to visualize the distribution of these random samples using the `hist` function. The code produces the output as shown in Figure 1.1. Similarly, Figure 1.2 visualizes the discrete distribution Binomial($N, \theta$) distribution where $N = 10$ and $\theta = 0.7$. The binomial arises in situations where a researcher counts the number of successes out of a given number of trials. For example, the Binomial(10, 0.7) distribution represents a situation where we have 10 total trials and the probability of success at each trial, $\theta$, equals 0.7.

Exercises

1. Adapt the Matlab program in Listing 1.1 to illustrate the Beta($\alpha, \beta$) distribution where $\alpha = 2$ and $\beta = 3$. Similarly, show the Exponential($\lambda$) distribution where $\lambda = 2$. As discussed in the note at the beginning of this book, you can use the publish option to organize the source code as well as the resulting figures into a single document.

2. Adapt the matlab program above to illustrate the Binomial($N, \theta$) distribution where
Listing 1.1: Matlab code to visualize Normal distribution.

```matlab
%% Explore the Normal distribution N( mu , sigma )
mu = 100;  % the mean
sigma = 15;  % the standard deviation
xmin = 70;  % minimum x value for pdf and cdf plot
xmax = 130;  % maximum x value for pdf and cdf plot
n = 100;  % number of points on pdf and cdf plot
k = 10000;  % number of random draws for histogram

% create a set of values ranging from xmin to xmax
x = linspace( xmin , xmax , n );
p = normpdf( x , mu , sigma );  % calculate the pdf
c = normcdf( x , mu , sigma );  % calculate the cdf

figure( 1 ); clf;  % create a new figure and clear the contents
subplot( 1,3,1 );
plot( x , p , 'k' );
xlabel( 'x' ); ylabel( 'pdf' );
title( 'Probability Density Function' );

subplot( 1,3,2 );
plot( x , c , 'k' );
xlabel( 'x' ); ylabel( 'cdf' );
title( 'Cumulative Density Function' );

% draw k random numbers from a N( mu , sigma ) distribution
y = normrnd( mu , sigma , k , 1 );

subplot( 1,3,3 );
hist( y , 20 );
xlabel( 'x' ); ylabel( 'frequency' );
title( 'Histogram of random values' );
```

Figure 1.2: Illustration of the Binomial($N, \theta$) distribution where $N = 10$ and $\theta = 0.7$. 
N = 10 and θ = 0.7. Produce a illustration that looks similar to Figure 1.2.

3. Write a simple demonstration program to sample 10 values from a Bernoulli(θ) distribution with θ = 0.3. Note that the Bernoulli distribution is one of the simplest discrete distributions to simulate. There are only two possible outcomes, 0 and 1. With probability θ, the outcome is 1, and with probability 1 − θ, the outcome is 0. In other words, \( p(X = 1) = θ \), and \( p(X = 0) = 1 − θ \). This distribution can be used to simulate outcomes in a number of situations, such as head or tail outcomes from a weighted coin, correct/incorrect outcomes from a true/false question, etc. In Matlab, you can simulate the Bernoulli distribution using the binomial distribution with \( N = 1 \). However, for the purpose of this exercise, please write the code needed to sample Bernoulli distributed values that does not make use of the built-in binomial distribution.

4. It is often useful in simulations to ensure that each replication of the simulation gives the exact same result. In Matlab, when drawing random values from distributions, the values are different every time you restart the code. There is a simple way to “seed” the random number generators to insure that they produce the same sequence. Write a Matlab script that samples two sets of 10 random values drawn from a uniform distribution between \([0,1]\). Use the seeding function between the two sampling steps to demonstrate that the two sets of random values are identical. Your Matlab code could use the following line:

```matlab
seed=1; rand('state',seed); randn('state',seed);
```

5. Suppose we know from previous research that in a given population, IQ coefficients are Normally distributed with a mean of 100 and a standard deviation of 15. Calculate the probability that a randomly drawn person from this population has an IQ greater than 110 but smaller than 130. You can achieve this using one line of Matlab code. What does this look like?

** 6 The Dirichlet distribution is currently not supported by Matlab. Can you find a Matlab function, using online resources, that implements the sampling from a Dirichlet distribution?

1.2 Sampling from non-standard distributions

Suppose we wish to sample from a distribution that is not one of the simple standard distributions that is supported by Matlab. In modeling situations, this situation frequently arises, because a researcher can propose new noise processes or combinations of existing distributions. Computational methods for solving complex sampling problems often rely of sampling distributions that we do know how to sample from efficiently. The random values from these simple distributions can then be transformed or compared to the target distribution. In fact, some of the techniques discussed in this section are used by Matlab internally to sample from distributions such as the Normal and Exponential distributions.
1.2.1 Inverse transform sampling with discrete variables

Inverse transform sampling (also known as the inverse transform method) is a method for generating random numbers from any probability distribution given the inverse of its cumulative distribution function. The idea is to sample uniformly distributed random numbers (between 0 and 1) and then transform these values using the inverse cumulative distribution function. The simplicity of this procedure lies in the fact that the underlying sampling is just based on transformed uniform deviates. This procedure can be used to sample many different kinds of distributions. In fact, this is how many Matlab implements many of its random number generators.

It is easiest to illustrate this approach on a discrete distribution where we know the probability of each individual outcome. In this case, the inverse transform method just requires a simple table lookup. To give an example of some non-standard discrete distribution, we use some data from experiments that have looked at how well humans can produce uniform random numbers (e.g. Treisman and Faulkner, 1987). In these experiments, subjects produce a large number of random digits (0,...,9) and investigators tabulate the relative frequencies of each random digit produced. As you might suspect, subjects do not always produce uniform distributions. Table 1.2.1 shows some typical data. Some of the low and the high numbers are underrepresented while some specific digits (e.g. 4) are overrepresented. For some reason, the digits 0 and 9 were never generated by the subject (perhaps because the subject misinterpreted the instructions). In any case, this data is fairly typical and demonstrates that humans are not very good at producing uniformly distributed random numbers.

Table 1.2: Probability of digits observed in human random digit generation experiment. The generated digit is represented by \( X \); \( p(X) \) and \( F(X) \) are the probability mass and cumulative probabilities respectively. The data was estimated from subject 6, session 1, in experiment by Treisman and Faulkner (1987).

<table>
<thead>
<tr>
<th>( X )</th>
<th>( p(X) )</th>
<th>( F(X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>0.100</td>
<td>0.100</td>
</tr>
<tr>
<td>2</td>
<td>0.090</td>
<td>0.190</td>
</tr>
<tr>
<td>3</td>
<td>0.095</td>
<td>0.285</td>
</tr>
<tr>
<td>4</td>
<td>0.200</td>
<td>0.485</td>
</tr>
<tr>
<td>5</td>
<td>0.175</td>
<td>0.660</td>
</tr>
<tr>
<td>6</td>
<td>0.190</td>
<td>0.850</td>
</tr>
<tr>
<td>7</td>
<td>0.050</td>
<td>0.900</td>
</tr>
<tr>
<td>8</td>
<td>0.100</td>
<td>1.000</td>
</tr>
<tr>
<td>9</td>
<td>0.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Suppose we now want to mimic this process and write an algorithm that samples digits according to the probabilities shown in Table 1.2.1. Therefore, the program should produce
a 4 with probability .2, a 5 with probability .175, etc. For example, the code in Listing 1.2 implements this process using the built-in matlab function \texttt{randsample}. The code produces the illustration shown in Figure 1.2.1.

Instead of using the built-in functions such as \texttt{randsample} or \texttt{mnrnd}, it is helpful to consider how to implement the underlying sampling algorithm using the inverse transform method. We first need to calculate the cumulative probability distribution. In other words, we need to know the probability that we observe an outcome equal to or smaller than some particular value. If \( F(X) \) represents the cumulative function, we need to calculate \( F(X = x) = p(X \leq x) \). For discrete distribution, this can be done using simple summing. The cumulative probabilities of our example are shown in the right column of Table 1.2.1. In the algorithm, the idea is to sample uniform random deviates (i.e., random numbers between 0 and 1) and to compare each random number against the table of cumulative probabilities. The first outcome for which the random deviate \( \text{exceeds and does not equal} \) the associated cumulative probability corresponds to the sampled outcome. Figure 1.2.1 illustrates this when the uniform random number, \( U = 0.8 \) leading to a sampled outcome \( X = 6 \). This process of repeated sampling of uniform deviates and comparing these to the cumulative distribution forms the basis for the inverse transform method for discrete variables. Note that we are applying an inverse function, because we are doing an inverse table lookup.

Listing 1.2: Matlab code to simulate sampling of random digits.

```matlab
1 \% Simulate the distribution observed in the
2 \% human random digit generation task
3
4 \% probabilities for each digit
5 theta = [0.000; ... \% digit 0
6 0.100; ... \% digit 1
7 0.090; ... \% digit 2
8 0.095; ... \% digit 3
9 0.200; ... \% digit 4
10 0.175; ... \% digit 5
11 0.190; ... \% digit 6
12 0.050; ... \% digit 7
13 0.100; ... \% digit 8
14 0.000 ]; ... \% digit 9
15
16 \% fix the random number generator
17 seed = 1; \texttt{rand('state', seed)};
18
19 \% let's say we draw \( K \) random values
20 K = 10000;
21 digitset = 0:9;
22 Y = \texttt{randsample(digitset,K,true,theta)};
23
24 \% create a new figure
25 \texttt{figure( 1 ); \texttt{clf};}
26```
CHAPTER 1. SAMPLING FROM RANDOM VARIABLES

```matlab
% Show the histogram of the simulated draws
counts = hist(Y, digitset);
bar(digitset, counts, 'k');
xlim([-0.5 9.5]);
xlabel('Digit');
ylabel('Frequency');
title('Distribution of simulated draws of human digit generator');
```

Figure 1.3: Illustration of the BINOMIAL($N, \theta$) distribution where $N = 10$ and $\theta = 0.7$.

Exercises

1. Create the Matlab program that implements the inverse transform method for discrete variables. Use it to sample random digits with probabilities as shown in Table 1.2.1. In order to show that the algorithm is working, sample a large number of random digits and create a histogram similar to Figure 1.2.1. Your program should never sample digits 0 and 9 as they are given zero probability in the table.

** 2 One solution to the previous exercise that does not require any loops is by using the multinomial random number generator `mnrnd`. Can you show how to use this function to sample digits according to the probabilities shown in Table 1.2.1

** 3 Can you explain why the algorithm as described above might be inefficient when dealing with skewed probability distributions? [hint: imagine a situation where the first N-1 outcomes have zero probability and the last outcome has probability one]. Can you think of a simple change to the algorithm to improve its efficiency?
1.2.2 Inverse transform sampling with continuous variables

The inverse transform sampling approach can also be applied to continuous distributions. Generally, the idea is to draw uniform random deviates and to apply the inverse function of the cumulative distribution applied to the random deviate. In the following, let \( F(X) \) be the cumulative density function (CDF) of our target variable \( X \) and \( F^{-1}(X) \) be the inverse of this function, assuming that we can actually calculate this inverse. We wish to draw random values for \( X \). This can be done with the following procedure:

1. Draw \( U \sim \text{Uniform}(0, 1) \)
2. Set \( X = F^{-1}(U) \)
3. Repeat

Let’s illustrate this approach with a simple example. Suppose we want to sample random numbers from the exponential distribution. When \( \lambda > 0 \), the cumulative density function is \( F(x|\lambda) = 1 - \exp(-\lambda x) \). Using some simple algebra, one can find the inverse of this function, which is \( F^{-1}(u|\lambda) = -\log(1 - u)/\lambda \). This leads to the following sampling procedure to sample random numbers from a Exponential(\( \lambda \)) distribution:

1. Draw \( U \sim \text{Uniform}(0, 1) \)
2. Set \( X = -\log(1 - U)/\lambda \)
3. Repeat
Exercises

1. Implement the inverse transform sampling method for the exponential distribution. Sample a large number of values from this distribution, and show the distribution of these values. Compare the distribution you obtain against the exact distribution as obtained by the PDF of the exponential distribution (use the command `exppdf`).

**2** Matlab implements some of its own functions using Matlab code. For example, when you call the exponential random number generator `exprnd`, Matlab executes a function that is stored in its own internal directories. Please locate the Matlab function `exprnd` and inspect its contents. How does Matlab implement the sampling from the exponential distribution? Does it use the inverse transform method? Note that the path to this Matlab function will depend on your particular Matlab installation, but it probably looks something like

C:\Program Files\MATLAB\R2009B\toolbox\stats\exprnd.m

1.2.3 Rejection sampling

In many cases, it is not possible to apply the inverse transform sampling method because it is difficult to compute the cumulative distribution or its inverse. In this case, there are other options available, such as rejection sampling, and methods using Markov chain Monte Carlo approaches that we will discuss in the next chapter. The major advantage of the rejection sampling method is that it does not require any “burn-in” period, as is needed for Markov chain Monte Carlo approaches. Instead, all samples obtained during sampling can immediately be used as samples from the target distribution.

One way to illustrate the general idea of rejection sampling (also commonly called the “accept-reject algorithm”) is with Figure 1.5. Suppose we wish to draw points uniformly within a circle centered at (0, 0) and with radius 1. At first, it seems quite complicated to directly sample points within this circle in uniform fashion. However, we can apply rejection sampling by first drawing \((x, y)\) values uniformly from within the square surrounding the circle, and rejecting any samples for which \(x^2 + y^2 > 1\). Importantly, we used a very simple proposal distribution, such as the uniform distribution, as a basis for sampling from a much more complicated distribution.

Rejection sampling allows us to generate observations from a distribution that is difficult to sample from but where we can evaluate the probability of any particular sample. In other words, suppose we have a distribution \(p(\theta)\), and it is difficult to sample from this distribution directly, but we can evaluate the probability density or mass \(p(\theta)\) for a particular value of \(\theta\). The first choice that the researcher needs to make is the proposal distribution. The proposal distribution is a simple distribution \(q(\theta)\), that we can directly sample from. The idea is to evaluate the probability of the proposed samples under both the proposal distribution and the target distribution and reject samples that are unlikely under the target distribution relative to the proposal distribution.
CHAPTER 1. SAMPLING FROM RANDOM VARIABLES

Figure 1.5: Sampling points uniformly from unit circle using rejection sampling

Figure 1.6: Illustration of rejection sampling. The particular sample shown in the figure will be rejected
Figure 1.6 illustrates the procedure. We first need to find a constant $c$ such that $cq(\theta) \geq p(\theta)$ for all possible samples $\theta$. The proposal function $q(\theta)$ multiplied by the constant $c$ is known as the comparison distribution and will always lie “on top of” our target distribution. Finding the constant $c$ might be non-trivial, but let’s assume for now that we can do this using some calculus. We now draw a number $u$ from a uniform distribution between $[0, cq(\theta)]$. In other words, this is some point on the line segment between 0 and the height of the comparison distribution evaluated at our proposal $\theta$. We will reject the proposal if $u > p(\theta)$ and accept it otherwise. If we accept the proposal, the sampled value $\theta$ is a draw from the target distribution $p(\theta)$. Here is a summary of the computational procedure:

1. Choose a density $q(\theta)$ that is easy to sample from
2. Find a constant $c$ such that $cq(\theta) \geq p(\theta)$ for all $\theta$
3. Sample a proposal $\theta$ from proposal distribution $q(\theta)$
4. Sample a uniform deviate $u$ from the interval $[0, cq(\theta)]$
5. Reject the proposal if $u > p(\theta)$, accept otherwise
6. Repeat steps 3, 4, and 5 until desired number of samples is reached; each accepted sample is a draw from $p(\theta)$

The key to an efficient operation of this algorithm is to have as many samples accepted as possible. This depends crucially on the choice of the proposal distribution. A proposal distribution that is dissimilar to the target distribution will lead to many rejected samples, slowing the procedure down.

Exercises

1. Suppose we want to sample from a Beta($\alpha, \beta$) distribution where $\alpha = 2$ and $\beta = 1$. This gives the probability density $p(x) = 2x$ for $0 < x < 1$. Implement a rejection sampling algorithm in Matlab that samples from this distribution. You can use a simple uniform proposal distribution. The constant $c$ should be 2 in this case. Visualize the histogram of sampled values and verify that the distribution matches the histogram obtained by using Matlab’s betarnd sampling function. What is the percentage of accepted samples? How might we improve the rejection sampler

** 2 The procedure shown in Figure 1.5 forms the basis for the Box-Muller method for generating Gaussian distributed random variables. We first generate uniform coordinates $(x, y)$ from the unit circle using the rejection sampling procedure that rejects any $(x, y)$ pair with $x^2 + y^2 > 1$. Then, for each pair $(x, y)$ we evaluate the quantities $z_1 = x \left(\frac{-2\ln(x^2+y^2)}{x^2+y^2}\right)^{1/2}$ and $z_2 = y \left(\frac{-2\ln(x^2+y^2)}{x^2+y^2}\right)^{1/2}$. The values $z_1$ and $z_2$ are each Gaussian distributed with zero mean and unit variance. Write a Matlab program that
implements this Box-Muller method and verify that the sampled values are Gaussian distributed.
Chapter 2

Markov Chain Monte Carlo

The application of probabilistic models to data often leads to inference problems that require the integration of complex, high dimensional distributions. Markov chain Monte Carlo (MCMC), is a general computational approach that replaces analytic integration by summation over samples generated from iterative algorithms. Problems that are intractable using analytic approaches often are possible to solve using some form of MCMC, even with high-dimensional problems. The development of MCMC is arguably the most advance in the computational approach to statistics. While MCMC is very much an active research area, there are now some standardized techniques that are widely used. In this chapter, we will discuss two forms of MCMC: Metropolis-Hastings and Gibbs sampling. Before we go into these techniques though, we first need to understand the two main ideas underlying MCMC: Monte Carlo integration, and Markov chains.

2.1 Monte Carlo integration

Many problems in probabilistic inference require the calculation of complex integrals or summations over very large outcome spaces. For example, a frequent problem is to calculate the expectation of a function $g(x)$ for the random variable $x$ (for simplicity, we assume $x$ is a univariate random variable). If $x$ is continuous, the expectation is defined as:

$$E[g(x)] = \int g(x)p(x)dx$$

(2.1)

In the case of discrete variables, the integral is replaced by summation:

$$E[g(x)] = \sum g(x)p(x)dx$$

(2.2)

These expectations arise in many situations where we want to calculate some statistic of a distribution, such as the mean or variance. For example, with $g(x) = x$, we are calculating the mean of a distribution. Integration or summation using analytic techniques can become quite challenging for certain distributions. For example, the density $p(x)$ might have a
functional form that does not lend itself to analytic integration. For discrete distributions, the outcome space might become so large to make the explicit summation over all possible outcomes impractical.

The general idea of Monte Carlo integration is to use samples to approximate the expectation of a complex distribution. Specifically, we obtain a set of samples \( x^{(t)} \), \( t = 1, \ldots, N \), drawn independently from distribution \( p(x) \). In this case, we can approximate the expectations in 2.1 and 2.2 by a finite sum:

\[
E[g(x)] = \frac{1}{n} \sum_{t=1}^{n} g(x^{(t)})
\]  

(2.3)

In other words, we have now replaced analytic integration with summation over a suitably large set of samples. Generally, the accuracy of the approximation can be made as accurate as needed by increasing \( n \). Crucially, the precision of the approximation depends on the independence of the samples. When the samples are correlated, the effective sample size decreases. This is not an issue with the rejection sampler discussed in the last chapter, but a potential problem with MCMC approaches.

Exercises

1. Develop Matlab code to approximate the mean of a Beta(\( \alpha,\beta \)) distribution with \( \alpha = 3 \) and \( \beta = 4 \) using Monte Carlo integration. You can use the Matlab function \texttt{betarnd} to draw samples from a Beta distribution. You can compare your answer with the analytic solution: \( \alpha/(\alpha + \beta) \). [Note: this can be done with one line of Matlab code].

2. Similarly, approximate the variance of a Gamma(\( a,b \)) distribution with \( a = 1.5 \) and \( b = 4 \) by Monte Carlo integration. The Matlab command \texttt{gamrnd} allows you to sample from this distribution. Your approximation should get close to the theoretically derived answer of \( ab^2 \).

2.2 Markov Chains

A Markov chain is a stochastic process where we transition from one state to another state using a simple sequential procedure. We start a Markov chain at some state \( x^{(1)} \), and use a transition function \( p(x^{(t)}|x^{(t-1)}) \), to determine the next state, \( x^{(2)} \) conditional on the last state. We then keep iterating to create a sequence of states:

\[
x^{(1)} \rightarrow x^{(2)} \rightarrow \ldots \rightarrow x^{(t)} \rightarrow \ldots
\]  

(2.4)

Each such a sequence of states is called a Markov chain or simply chain. The procedure for generating a sequence of \( T \) states from a Markov chain is the following:

1. Set \( t = 1 \)
2. Generate a initial value \( u \), and set \( x^{(t)} = u \)

3. Repeat

\[
\begin{align*}
    t &= t + 1 \\
    \text{Sample a new value } u \text{ from the transition function } p(x^{(t)}|x^{(t-1)}) \\
    \text{Set } x^{(t)} &= u 
\end{align*}
\]

4. Until \( t = T \)

Importantly, in this iterative procedure, the next state of the chain at \( t + 1 \) is based only on the previous state at \( t \). Therefore, each Markov chain wanders around the state space and making a move to a new state that is only dependent on the last state. It is this local dependency what makes this procedure “Markov” or “memoryless”. As we will see, this is an important property when using Markov chains for MCMC.

When initializing each Markov chain, the chain will wander in state space around the starting state. Therefore, if we start a number of chains, each with different initial conditions, the chains will initially be in a state close to the starting state. This period is called the burnin. An important property of Markov chains is that the starting state of the chain no longer affects the state of the chain after a sufficiently long sequence of transitions (assuming that certain conditions about the Markov chain are met). At this point, the chain is said to reach its steady state and the states reflect samples from its stationary distribution. This property that Markov chains converge to a stationary distribution regardless of where we started (if certain regularity conditions of the transition function are met), is quite important. When applied to MCMC, it allow us to draw samples from a distribution using a sequential procedure but where the starting state of the sequence does not affect the estimation process.

**Example.** Figure 2.1 show an example of a Markov chain involving a (single) continuous variable \( x \). For the transition function, samples were taken from a Beta\((200(0.9x^{(t-1)} + 0.05), 200(1 - 0.9x^{(t-1)} - 0.05))\) distribution. This function and its constants are chosen somewhat arbitrarily, but help to illustrate some basic aspects of Markov chains. The process was started with four different initial values, and each chain was for continued for \( T = 1000 \) iterations. The two panels of the Figure show the sequence of states at two different time scales. The line colors represent the four different chains. Note that the first 10 iterations or so show a dependence of the sequence on the initial state. This is the burnin period. This is followed by the steady state for the remainder of the sequence (the chain would continue in the steady state if we didn’t stop it). How do we know exactly when the steady state has been reached and the chain converges? This is often not easy to tell, especially in high-dimensional state spaces. We will differ the discussion of convergence until later.

**Exercises**

1. Develop Matlab code to implement the Markov chain as described in the example. Create an illustration similar to one of the panels in Figure 2.1. Start the Markov
Figure 2.1: Illustration of a Markov chain starting with four different initial conditions. The right and left panes show the sequence of states at different temporal scales.

The goal of MCMC is to design a Markov chain such that the stationary distribution of the chain is exactly the distribution that we are interesting in sampling from, called the target distribution. In other words, we would like the states sampled from some Markov chain to also be samples drawn from the target distribution. The idea is to use some clever methods for setting up the transition function such that no matter how we initialize each chain, we will convergence to the target distribution. There are a number of methods that achieve this goal using relatively simple procedures. We will discuss Metropolis, Metropolis-Hastings, and Gibbs sampling.

2.4 Metropolis Sampling

We will start by illustrating the simplest of all MCMC methods: the Metropolis sampler. This is a special case of the Metropolis-Hastings sampler discussed in the next section. Suppose our goal is to sample from the target density \( p(\theta) \), with \(-\infty < \theta < \infty\). The Metropolis sampler creates a Markov chain that produces a sequence of values:

\[
\theta^{(1)} \rightarrow \theta^{(2)} \rightarrow \ldots \rightarrow \theta^{(t)} \rightarrow \ldots
\]
where $\theta^{(t)}$ represents the state of a Markov chain at iteration $t$. The samples from the chain, after burnin, start to reflect samples from the target distribution $p(\theta)$.

In the Metropolis procedure, we initialize the first state, $\theta^{(1)}$ to some initial value. We then use a proposal distribution $q(\theta|\theta^{(t-1)})$ to generate a candidate point $\theta^*$ that is conditional on the previous state of the sampler. The next step is to either accept the proposal or reject it. The probability of accepting the proposal is:

$$\alpha = \min\left(1, \frac{p(\theta^*)}{p(\theta^{(t-1)})}\right)$$  \hspace{1cm} (2.6)

To make a decision on whether to actually accept or reject the proposal, we generate a uniform deviate $u$. If $u \leq \alpha$, we accept the proposal and the next state is set equal to the proposal: $\theta^{(t)} = \theta^*$. If $u > \alpha$, we reject the proposal, and the next state is set equal to the old state: $\theta^{(t)} = \theta^{(t-1)}$. We continue generating new proposals conditional on the current state of the sampler, and either accept or reject the proposals. This procedure continues until the sampler reaches convergence. At this point, the samples $\theta^{(t)}$ reflect samples from the target distribution $p(\theta)$. Here is a summary of the steps of the Metropolis sampler:

1. Set $t = 1$
2. Generate a initial value $u$, and set $\theta^{(t)} = u$
3. Repeat
   - $t = t + 1$
   - Generate a proposal $\theta^*$ from $q(\theta|\theta^{(t-1)})$
   - Evaluate the acceptance probability $\alpha = \min\left(1, \frac{p(\theta^*)}{p(\theta^{(t-1)})}\right)$
   - Generate a $u$ from a Uniform(0,1) distribution
   - If $u \leq \alpha$, accept the proposal and set $\theta^{(t)} = \theta^*$, else set $\theta^{(t)} = \theta^{(t-1)}$.
4. Until $t = T$

Figure 2.2 illustrates the procedure for a sequence of two states. To intuitively understand why the process leads to samples from the target distribution, note that 2.6 will always accept a new proposal if the new proposal is more likely under the target distribution than the old state. Therefore, the sampler will move towards the regions of the state space where the target function has high density. However, note that if the new proposal is less likely than than the current state, it is still possible to accept this “worse” proposal and move toward it. This process of always accepting a “good” proposal, and occasionally accepting a “bad” proposal insures that the sampler explores the whole state space, and samples from all parts of a distribution (including the tails).

A key requirement for the Metropolis sampler is that the proposal distribution is symmetric, such that $q(\theta = \theta^{(t)}|\theta^{(t-1)}) = q(\theta = \theta^{(t-1)}|\theta^{(t)})$. Therefore, the probability of proposing some new state given the old state, is the same as proposing to go from the new state back to
the old state. This symmetry holds with proposal distributions such as the Normal, Cauchy,
Student-t, as well as uniform distributions. If this symmetry does not hold, you should use
a Metropolis-Hastings sampler discussed in the next section.

A major advantage of the Metropolis sampler is that Equation 2.6 involves only a ratio of
densities. Therefore, any terms independent of \( \theta \) in the functional form of \( p(\theta) \) will drop out. Therefore, we do not need to know the normalizing constant of the density or probability mass
function. The fact that this procedure allows us to sample from unnormalized distributions
is one of its major attractions. Sampling from unnormalized distributions frequently happens
in Bayesian models, where calculating the normalization constant is difficult or impractical.

**Example 1.** Suppose we wish to generate random samples from the Cauchy distribution
(note that here are better ways to sample from the Cauchy that do not rely on MCMC, but
we just use as an illustration of the technique). The probability density of the Cauchy is
given by:

\[
f(\theta) = \frac{1}{\pi(1 + \theta^2)}
\]  
(2.7)

Note that because we do not need any normalizing constants in the Metropolis sampler, we
can rewrite this to:

\[
f(\theta) \propto \frac{1}{1 + \theta^2}
\]  
(2.8)

Therefore, the Metropolis acceptance probability becomes

\[
\alpha = \min\left(1, \frac{1 + [\theta(t)^2]}{1 + [\theta^*]^2}\right)
\]  
(2.9)

We will use the Normal distribution as the proposal distribution. Our proposals are generated
from a Normal\((\theta(t), \sigma)\) distribution. Therefore, the mean of the distribution is centered on
the current state and the parameter \( \sigma \), which needs to be set by the modeler, controls the
variability of the proposed steps. This is an important parameter that we will investigate
in the the Exercises. Listing 2.1 show the Matlab function that returns the unnormalized
density of the Cauchy distribution. Listing 2.2 shows Matlab code that implements the
Metropolis sampler. Figure 2.3 shows the simulation results for a single chain run for 500
iterations. The upper panel shows the theoretical density in the dashed red line and the
histogram shows the distribution of all 500 samples. The lower panel shows the sequence of
samples of one chain.

**Exercises**

1. Currently, the program in Listing 2.2 takes all states from the chain as samples to
approximate the target distribution. Therefore, it also includes samples while the
chain is still “burning in”. Why is this not a good idea? Can you modify the code
such that the effect of burnin is removed?

2. Explore the effect of different starting conditions. For example, what happens when
we start the chain with \( \theta = -30 \)?
Figure 2.2: Illustration of the Metropolis sampler to sample from target density $p(\theta)$. (A) the current state of the chain is $\theta^{(t)}$. (B) a proposal distribution around the current state is used to generate a proposal $\theta^*$. (C) the proposal was accepted and the new state is set equal to the proposal, and the proposal distribution now centers on the new state.
3. Calculate the proportion of samples that is accepted on average. Explore the effect of parameter $\sigma$ on the average acceptance rate. Can you explain what is happening with the accuracy of the reconstructed distribution when $\sigma$ is varied?

4. As a followup to the previous question, what is (roughly) the value of $\sigma$ that leads to a 50% acceptance rate? It turns out that this is the acceptance rate for which the Metropolis sampler, in the case of Gaussian distributions, converges most quickly to the target distribution.

5. Suppose we apply the Metropolis sampler to a Normal($\mu, \sigma$) density as the target distribution, with $\mu = 0$ and $\sigma = 1$. Write down the equation for the acceptance probability, and remove any proportionality constants from the density ratio. [note: the Matlab documentation for `normpdf` shows the functional form for the Normal density.

** 6 Modify the code such that the sequences of multiple chains (each initialized differently) are visualized simultaneously.

Listing 2.1: Matlab function to evaluate the unnormalized Cauchy.

```matlab
function y = cauchy( theta )
    %% Returns the unnormalized density of the Cauchy distribution
    y = 1 ./ (1 + theta.^2);
```

2.5 Metropolis-Hastings Sampling

The Metropolis-Hasting (MH) sampler is a generalized version of the Metropolis sampler in which we can apply symmetric as well as asymmetric proposal distributions. The MH sampler operates in exactly the same fashion as the Metropolis sampler, but uses the following acceptance probability:

$$
\alpha = \min\left(1, \frac{p(\theta^*)}{p(\theta^{(t-1)})} \frac{q(\theta^{(t-1)}|\theta^*)}{q(\theta^*|\theta^{(t-1)})}\right)
$$

The MH sampler has the additional ratio of $\frac{q(\theta^{(t-1)}|\theta^*)}{q(\theta^*|\theta^{(t-1)})}$ in 2.10. This corrects for any asymmetries in the proposal distribution. For example, suppose we have a proposal distribution with a mean centered on the current state, but that is skewed in one direction. If the proposal distribution prefers to move say left over right, the proposal density ratio will correct for this asymmetry.

Here is a summary of the steps of the MH sampler:

1. Set $t = 1$

2. Generate an initial value $u$, and set $\theta^{(t)} = u$
Listing 2.2: Matlab code to implement Metropolis sampler for Example 1

```matlab
%% Chapter 2. Use Metropolis procedure to sample from Cauchy density

%% Initialize the Metropolis sampler
T = 500; % Set the maximum number of iterations
sigma = 1; % Set standard deviation of normal proposal density
thetamin = -30; thetamax = 30; % define a range for starting values
theta = zeros(1, T); % Init storage space for our samples
seed = 1; rand('state', seed); randn('state', seed); % set the random seed
theta(1) = unifrnd(thetamin, thetamax); % Generate start value

%% Start sampling
while t < T % Iterate until we have T samples
    t = t + 1;
    % Propose a new value for theta using a normal proposal density
    theta_star = normrnd(theta(t-1), sigma);
    % Calculate the acceptance ratio
    alpha = min([1 cauchy(theta_star) / cauchy(theta(t-1))]);
    % Draw a uniform deviate from [0 1]
    u = rand;
    % Do we accept this proposal?
    if u < alpha
        theta(t) = theta_star; % If so, proposal becomes new state
    else
        theta(t) = theta(t-1); % If not, copy old state
    end
end

%% Display histogram of our samples
figure(1); clf;
subplot(3,1,1);
nbins = 200;
theatabins = linspace(thetamin, thetamax, nbins);
counts = hist(theta, theatabins);
bar(theatabins, counts/sum(counts), 'k');
xlim([thetamin thetamax]);
xlabel('\theta'); ylabel('p(\theta)

%% Overlay the theoretical density
y = cauchy(theatabins);
hold on;
plot(theatabins, y/sum(y), 'r--', 'LineWidth', 3);
set(gca, 'YTick', []);

%% Display history of our samples
subplot(3,1,2:3);
stairs(theta, 1:T, 'k-');
ylabel('t'); xxlabel('\theta');
set(gca, 'YDir', 'reverse');
xlim([thetamin thetamax]);
```

Figure 2.3: Simulation results where 500 samples were drawn from the Cauchy distribution using the Metropolis sampler. The upper panel shows the theoretical density in the dashed red line and the histogram shows the distribution of the samples. The lower panel shows the sequence of samples of one chain.
3. Repeat

\[ t = t + 1 \]

Generate a proposal \( \theta^* \) from \( q(\theta | \theta^{(t-1)}) \)

Evaluate the acceptance probability \( \alpha = \min \left( 1, \frac{p(\theta^*)}{p(\theta^{(t-1)})} \frac{q(\theta^{(t-1)} | \theta^*)}{q(\theta^* | \theta^{(t-1)})} \right) \)

Generate a \( u \) from a Uniform(0,1) distribution

If \( u \leq \alpha \), accept the proposal and set \( \theta^{(t)} = \theta^* \), else set \( \theta^{(t)} = \theta^{(t-1)} \).

4. Until \( t = T \)

The fact that asymmetric proposal distributions can be used allows the Metropolis-Hastings procedure to sample from target distributions that are defined on a limited range (other than the uniform for which Metropolis sampler can be used). With bounded variables, care should be taken in constructing a suitable proposal distribution. Generally, a good rule is to use a proposal distribution has positive density on the same support as the target distribution. For example, if the target distribution has support over \( 0 \leq \theta < \infty \), the proposal distribution should have the same support.

Exercise

1. Suppose a researcher investigates response times in an experiment and finds that the Weibull(\( a, b \)) distribution with \( a = 2 \), and \( b = 1.9 \) captures the observed variability in response times. Write a Matlab program that implements the Metropolis-Hastings sampler in order to sample response times from this distribution. The pdf for the Weibull is given by the Matlab command \( \text{wblpdf} \). Create a figure that is analogous to Figure 2.3. You could use a number of different proposal distributions in this case. For this exercise, use samples from a Gamma(\( \theta^{(t)} \tau, 1/\tau \)) distribution. This proposal density has a mean equal to \( \theta^{(t)} \) so it is “centered” on the current state. The parameter \( \tau \) controls the acceptance rate of the sampler – it is a precision parameter such that higher values are associated with less variability in the proposal distribution. Can you find a value for \( \tau \) to get (roughly) an acceptance rate of 50%? Calculate the variance of this distribution using the Monte Carlo approach with the samples obtained from the Metropolis-Hastings sampler. If you would like to know how close your approximation is, use online resources to find the analytically derived answer.

2.6 Metropolis-Hastings for Multivariate Distributions

Up to this point, all of the examples we discussed involved univariate distributions. It is fairly straightforward though to generalize the MH sampler to multivariate distributions. There are two different ways to extend the procedure to sample random variables in multidimensional spaces.
2.6.1 Blockwise updating

In the first approach, called blockwise updating, we use a proposal distribution that has the same dimensionality as the target distribution. For example, if we want to sample from a probability distribution involving \( N \) variables, we design a \( N \)-dimensional proposal distribution, and we either accept or reject the proposal (involving values for all \( N \) variables) as a block. In the following, we will use the vector notation \( \mathbf{\theta} = (\theta_1, \theta_2, \ldots, \theta_N) \) to represent a random variable involving \( N \) components, and \( \mathbf{\theta}^{(t)} \) represents the \( t \)th state in our sampler. This leads to a generalization of the MH sampler where the scalar variables \( \theta \) are now replaced by vectors \( \mathbf{\theta} \):

1. Set \( t = 1 \)
2. Generate an initial value \( \mathbf{u} = (u_1, u_2, \ldots, u_N) \), and set \( \mathbf{\theta}^{(t)} = \mathbf{u} \)
3. Repeat
   - \( t = t + 1 \)
   - Generate a proposal \( \mathbf{\theta}^* \) from \( q(\mathbf{\theta}|\mathbf{\theta}^{(t-1)}) \)
   - Evaluate the acceptance probability \( \alpha = \min\left(1, \frac{p(\mathbf{\theta}^*)}{p(\mathbf{\theta}^{(t-1)})} \frac{q(\mathbf{\theta}^{(t-1)}|\mathbf{\theta}^*)}{q(\mathbf{\theta}^{(t-1)}|\mathbf{\theta}^{(t-1)})}\right) \)
   - Generate a \( u \) from a Uniform(0,1) distribution
   - If \( u \leq \alpha \), accept the proposal and set \( \mathbf{\theta}^{(t)} = \mathbf{\theta}^* \), else set \( \mathbf{\theta}^{(t)} = \mathbf{\theta}^{(t-1)} \).
4. Until \( t = T \)

Example 1 (adopted from Gill, 2008). Suppose we want to sample from the bivariate exponential distribution

\[
p(\theta_1, \theta_2) = \exp\left(- (\lambda_1 + \lambda)\theta_1 - (\lambda_2 + \lambda)\theta_2 - \lambda \max(\theta_1, \theta_2)\right) \tag{2.11}
\]

For our example, we will restrict the range of \( \theta_1 \) and \( \theta_2 \) to [0,8] and the set the constants to the following: \( \lambda_1 = 0.5, \lambda_2 = 0.1, \lambda + 0.01, \max(\theta_1, \theta_2) = 8 \). This bivariate density is visualized in Figure 2.4, right panel. The Matlab function that implements this density function is shown in Listing 2.3. To illustrate the blockwise MH sampler, we use a uniform proposal distribution, where proposals for \( \theta_1^* \) and \( \theta_2^* \) are sampled from a Uniform(0,8) distribution. In other words, we sample proposals for \( \mathbf{\theta}^* \) uniformly from within a box. Note that with this particular proposal distribution, we are not conditioning our proposals on the previous state of the sampler. This is known as an independence sampler. This is actually a very poor proposal distribution but leads to a simple implementation because the ratio \( \frac{q(\mathbf{\theta}^{(t-1)}|\mathbf{\theta}^*)}{q(\mathbf{\theta}^*|\mathbf{\theta}^{(t-1)})} = 1 \) and therefore disappears from the acceptance ratio. The Matlab code that implements the sampler is shown in Listing 2.4. Figure 2.4, left panel shows the approximated distribution using 5000 samples.

Example 2. Many researchers have proposed probabilistic models for order information. Order information can relate to preference rankings over political candidates, car brands
and icecream flavors, but can also relate to knowledge about the relative order of items along some temporal or physical dimension. For example, suppose we ask individuals to remember the chronological order of US presidents. Steyvers, Lee, Miller, and Hemmer (2009) found that individuals make a number of mistakes in the ordering of presidents that can be captured by simple probabilistic models, such as Mallows model. To explain Mallows model, let’s say that we are looking at the first five presidents: Washington, Adams, Jefferson, Madison, and Monroe. We will represent this true ordering by a vector \( \omega = (1, 2, 3, 4, 5) = \text{Washington, Adams, Jefferson, Madison, Monroe} \). Mallows model now proposes that the remembered orderings tend to be similar to the true ordering, with very similar orderings being more likely than dissimilar orderings. Specifically, according to Mallows model, the probability that an individual remembers an ordering \( \theta \) is proportional to:

\[
p(\theta | \omega, \lambda) \propto \exp \left( -d(\theta, \omega) \lambda \right)
\]  

In this equation, \( d(\theta, \omega) \) is the Kendall tau distance between two orderings. This distance measures the number of adjacent pairwise swaps that are needed to bring the two orderings into alignment. For example, if \( \theta = (\text{Adams, Washington, Jefferson, Madison, Monroe}) \), then \( d(\theta, \omega) = 1 \) because one swap is needed to make the two orderings identical. Similarly, if \( \theta = (\text{Adams, Jefferson, Washington, Madison, Monroe}) \), then \( d(\theta, \omega) = 2 \) because two swaps are needed. Note that in Kendall tau distance, only adjacent items can be swapped. The scaling parameter \( \lambda \) controls how sharply peaked the distribution of remembered orderings is around the true ordering. Therefore, by increasing \( \lambda \), the model makes it more likely that the correct ordering (or something similar) will be produced.

The problem is now to generate orderings \( \theta \) according to Mallows model, given the true ordering \( \omega \) and scaling parameter \( \lambda \). This can be achieved in very simple ways using a Metropolis sampler. We start the sampler with \( \theta^{(1)} \) corresponding to a random permutation.
of items. At each iteration, we then make proposals \( \theta^* \) that slightly modify the current state. This can be done in a number of ways. The idea here is to use a proposal distribution where the current ordering is permuted by transposing any randomly chosen pair of items (and not just adjacent items). Formally, we draw proposals \( \theta^* \) from the proposal distribution

\[
q(\theta = \theta^* | \theta^{(t-1)}) = \begin{cases} 
\frac{1}{N^2} & \text{if } S(\theta^*, \theta^{(t-1)}) = 1 \\
0 & \text{otherwise} 
\end{cases}
\]

where \( S(\theta^*, \theta^{(t-1)}) \) is the Cayley distance. This distance counts the number of transpositions of any pair of items needed to bring two orderings into alignment (therefore, the difference with the Kendall tau distance is that any pairwise swap counts as one, even nonadjacent swaps). This is just a complicated way to describe a very simple proposal distribution: just swap two randomly chosen items from the last ordering, and make that the proposed ordering.

Because the proposal distribution is symmetric, we can use the Metropolis sampler. The acceptance probability is

\[
\alpha = \min \left( 1, \frac{p(\theta^* | \omega, \lambda)}{p(\theta^{(t-1)} | \omega, \lambda)} \right) = \min \left( 1, \frac{\exp(-d(\theta^*, \omega) \lambda)}{\exp(-d(\theta^{(t-1)}, \omega) \lambda)} \right).
\]

The Matlab implementation of the Kendall tau distance function is shown in Listing 2.5. The Matlab code for the Metropolis sampler is shown in Listing 2.6. Currently, the code does not do all that much. It simply shows what the state is every 10 iterations for a total of 500 iterations. Here is some sample output from the program:

<table>
<thead>
<tr>
<th>t</th>
<th>Jefferson</th>
<th>Madison</th>
<th>Adams</th>
<th>Monroe</th>
<th>Washington</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Jefferson</td>
<td>Madison</td>
<td>Adams</td>
<td>Monroe</td>
<td>Washington</td>
</tr>
<tr>
<td>410</td>
<td>Washington</td>
<td>Monroe</td>
<td>Madison</td>
<td>Jefferson</td>
<td>Adams</td>
</tr>
<tr>
<td>420</td>
<td>Washington</td>
<td>Jefferson</td>
<td>Madison</td>
<td>Adams</td>
<td>Monroe</td>
</tr>
<tr>
<td>430</td>
<td>Jefferson</td>
<td>Monroe</td>
<td>Washington</td>
<td>Adams</td>
<td>Madison</td>
</tr>
<tr>
<td>440</td>
<td>Washington</td>
<td>Madison</td>
<td>Monroe</td>
<td>Adams</td>
<td>Jefferson</td>
</tr>
<tr>
<td>450</td>
<td>Jefferson</td>
<td>Washington</td>
<td>Adams</td>
<td>Madison</td>
<td>Monroe</td>
</tr>
<tr>
<td>460</td>
<td>Washington</td>
<td>Jefferson</td>
<td>Adams</td>
<td>Madison</td>
<td>Monroe</td>
</tr>
<tr>
<td>470</td>
<td>Monroe</td>
<td>Washington</td>
<td>Jefferson</td>
<td>Adams</td>
<td>Madison</td>
</tr>
<tr>
<td>480</td>
<td>Adams</td>
<td>Washington</td>
<td>Monroe</td>
<td>Jefferson</td>
<td>Madison</td>
</tr>
<tr>
<td>490</td>
<td>Adams</td>
<td>Madison</td>
<td>Jefferson</td>
<td>Monroe</td>
<td>Washington</td>
</tr>
<tr>
<td>500</td>
<td>Monroe</td>
<td>Adams</td>
<td>Madison</td>
<td>Jefferson</td>
<td>Washington</td>
</tr>
</tbody>
</table>

### 2.6.2 Componentwise updating

A potential problem with the blockwise updating approach is that it might be difficult to find suitable high-dimensional proposal distributions. A related problem is that blockwise updating can be associated with high rejection rates. Instead of accepting or rejecting a proposal for \( \theta \) involving all its components simultaneously, it might be computationally simpler to make proposals for individual components of \( \theta \), one at a time. This leads to a componentwise updating approach.
For example, suppose we have a bivariate distribution $\theta = (\theta_1, \theta_2)$. We first initialize the sampler with some suitable values for $\theta_1^{(1)}$ and $\theta_2^{(1)}$. At each iteration $t$, we first make a proposal $\theta^*_1$ depending on the last state $\theta_1^{(t-1)}$. We then evaluate the acceptance ratio comparing the likelihood of $(\theta_1^*, \theta_2^{(t-1)})$ against $(\theta_1^{(t-1)}, \theta_2^{(t-1)})$. Note that in this proposal, we have only varied the first component and kept the second component constant. In the next step, we make a proposal $\theta^*_2$ depending on the last state $\theta_2^{(t-1)}$. We then evaluate the acceptance ratio comparing the likelihood of $(\theta_1^{(t)}, \theta_2^*)$ against $(\theta_1^{(t)}, \theta_2^{(t-1)})$. Importantly, in this second step, we are holding the first component constant, but this is the updated value from the first step. Therefore, what happens in the second step is conditional on what happens in the first step. Here is a summary of the steps for a bivariate componentwise MH sampler:

1. Set $t = 1$

2. Generate an initial value $u = (u_1, u_2, \ldots, u_N)$, and set $\theta^{(t)} = u$

3. Repeat
   
   $t = t + 1$

   Generate a proposal $\theta^*_1$ from $q(\theta_1|\theta_1^{(t-1)})$

   Evaluate the acceptance probability $\alpha = \min\left(1, \frac{p(\theta_1^*, \theta_2^{(t-1)})}{p(\theta_1^{(t-1)}, \theta_2^{(t-1)})} \frac{q(\theta_1^{(t-1)}|\theta_1^*)}{q(\theta_1^{(t-1)}|\theta_1)}\right)$

   Generate a $u$ from a Uniform(0,1) distribution

   If $u \leq \alpha$, accept the proposal and set $\theta_1^{(t)} = \theta_1^*$, else set $\theta_1^{(t)} = \theta_1^{(t-1)}$.

   Generate a proposal $\theta^*_2$ from $q(\theta_2|\theta_2^{(t-1)})$

   Evaluate the acceptance probability $\alpha = \min\left(1, \frac{p(\theta_1^{(t)}, \theta_2^*)}{p(\theta_1^{(t)}, \theta_2^{(t-1)})} \frac{q(\theta_2^{(t-1)}|\theta_2^*)}{q(\theta_2^{(t-1)}|\theta_2)}\right)$

   Generate a $u$ from a Uniform(0,1) distribution

   If $u \leq \alpha$, accept the proposal and set $\theta_2^{(t)} = \theta_2^*$, else set $\theta_2^{(t)} = \theta_2^{(t-1)}$.

4. Until $t = T$

**Example 3.** We will illustrate the componentwise sampler by sampling from a multivariate normal distribution. Please note that this just serves as an example – there are much better and direct ways to sample from a multivariate normal distribution. The multivariate normal distribution is parametrized by $\mu$ and $\Sigma$. The parameter $\mu$ is a 1 x $N$ vector containing the mean in the $N$-dimensional space. The parameter $\Sigma$ is a $N \times N$ covariance matrix. For our example, we will use a bivariate normal with $\mu = (0, 0)$ and $\Sigma = \begin{pmatrix} 1 & 0.3 \\ 0.3 & 1 \end{pmatrix}$. The density of this distribution is illustrated in the right panel of Figure 2.5. The goal is to sample values $\theta$ from this distribution. We use a componentwise Metropolis sampler with a normal distribution for the components $\theta_1$ and $\theta_2$. The proposal distribution is centered
around the old value of each component and has standard deviation $\sigma$. The code in Listing 2.7 shows the Matlab implementation for this sampler. The output of this code is shown in Figure 2.5.

![Figure 2.5: Results of the componentwise Metropolis sampler for the bivariate normal distribution. Left panel shows the approximation using 5000 samples. Right panel shows the theoretical density.](image)

**Exercises**

1. Modify the code for Example 1 to implement a component-wise sampler. Use a uniform proposal distribution for both $\theta_1$ and $\theta_2$. Visualize the distribution in the same way as Figure 2.4. Calculate the acceptance probability separately for $\theta_1$ and $\theta_2$. Why is the acceptance probability for $\theta_2$ higher than for $\theta_1$?

2. Investigate the effect of the scaling parameter $\lambda$ in Mallows model. How do the sampled orders change as a function of this parameter?

3. Modify the Metropolis sampler for Mallows model and sample from a 10 as opposed to 5 dimensional space of orderings.

**4** The purpose of this exercise is to compare the approximated distribution under Mal- lows with the exact distribution. Note that Equation 2.12 showed the probability for Mallows model but dropped the constant of proportionality. The exact probability under Mallows model is $p(\theta | \omega, \lambda) = \frac{1}{\psi(\lambda)} \exp (-d(\theta, \omega) \lambda)$ where $\psi(\lambda)$ is a normalizing
function where \( \psi(\lambda) = \prod_{i=1}^{N} \frac{1}{\exp(-i\lambda)} \). Therefore, it is possible with small \( N \) to calculate the exact probability of any ordering \( \theta \). Run the sampler in Listing 2.6 and calculate a frequency distribution of samples (i.e., what number of times does each unique sequence occur in the sampler?). Compare these frequencies against the exact probability under Mallows model.

Listing 2.3: Function to evaluate bivariate exponential distribution

```matlab
function y = bivexp( theta1, theta2 )

%% Returns the density of a bivariate exponential function

lambda1 = 0.5; % Set up some constants
lambda2 = 0.1;
lambda = 0.01;
maxval = 8;
y = exp( -(lambda1+lambda)*theta1 - (lambda2+lambda)*theta2 - lambda*maxval );
```

Listing 2.4: Blockwise Metropolis-Hastings sampler for bivariate exponential distribution

```matlab
%% Chapter 2. Metropolis procedure to sample from Bivariate Exponential
% Blockwise updating. Use a uniform proposal distribution

%% Initialize the Metropolis sampler
T = 5000; % Set the maximum number of iterations
thetamin = [ 0 0 ]; % define minimum for theta1 and theta2
thetamax = [ 8 8 ]; % define maximum for theta1 and theta2
seed=1; rand('state', seed); randn('state', seed); % set the random seed
theta = zeros( 2, T ); % Init storage space for our samples
theta(:,1) = unifrnd( thetamin(1), thetamax(1) ); % Start value for theta1
theta(:,1) = unifrnd( thetamin(2), thetamax(2) ); % Start value for theta2

%% Start sampling

%% % Start sampling

while t < T % Iterate until we have T samples
  t = t + 1;
  % Propose a new value for theta
  theta_star = unifrnd( thetamin, thetamax );
  pratio = bivexp( theta_star(1), theta_star(2) ) / ... 
            bivexp( theta(1,t-1), theta(2,t-1) );
  alpha = min( [ 1 pratio ] ); % Calculate the acceptance ratio
  u = rand; % Draw a uniform deviate from [ 0 1 ]
  if u < alpha % Do we accept this proposal?
    theta(:,t) = theta_star; % proposal becomes new value for theta
  else
    theta(:,t) = theta(:,t-1); % copy old value of theta
  end
end
```
%% Display histogram of our samples
figure( 1 ); clf;
subplot( 1,2,1 );
nbins = 10;
thetabins1 = linspace( thetamin(1) , thetamax(1) , nbins );
thetabins2 = linspace( thetamin(2) , thetamax(2) , nbins );
hist3( theta' , 'Edges' , {thetabins1 thetabins2} );
xlabel( '\theta_1' ); ylabel('\theta_2'); zlabel( 'counts' );
az = 61; el = 30;
view(az, el);

%% Plot the theoretical density
subplot(1,2,2);
nbins = 20;
therabins1 = linspace( thetamin(1) , thetamax(1) , nbins );
therabins2 = linspace( thetamin(2) , thetamax(2) , nbins );
[ thetalgrid , thetagrid2grid ] = meshgrid( therabins1 , therabins2 );
ygrid = bivexp( thetalgrid , thetagrid2grid );
mesh( thetalgrid , thetagrid2grid , ygrid );
xlabel( '\theta_1' ); ylabel( '\theta_2' );
zlabel( 'f(\theta_1,\theta_2)' );
view(az, el);

Listing 2.5: Function to evaluate Kendall tau distance
function tau = kendalltau( order1 , order2 )
%% Returns the Kendall tau distance between two orderings
% Note: this is not the most efficient implementation
[dummy , ranking1 ] = sort( order1(:)' , 2 , 'ascend' );
[dummy , ranking2 ] = sort( order2(:)' , 2 , 'ascend' );
N = length( ranking1 );
[ ii , jj ] = meshgrid( 1:N , 1:N );
ok = find( jj(:) > ii(:) );
ii = ii( ok );
jj = jj( ok );
nok = length( ok );
sign1 = ranking1( jj ) > ranking1( ii );
sign2 = ranking2( jj ) > ranking2( ii );
tau = sum( sign1 ~= sign2 );

Listing 2.6: Implementation of Metropolis-Hastings sampler for Mallows model
function samples orderings from a distribution over orderings
% Initialize model parameters
lambda = 0.1; % scaling parameter
labels = { 'Washington' , 'Adams' , 'Jefferson' , 'Madison' , 'Monroe' };
omega = [ 1 2 3 4 5 ]; % correct ordering
L = length( omega ); % number of items in ordering

%%% Initialize the Metropolis sampler
T = 500; % Set the maximum number of iterations
seed=1; rand( 'state', seed ); randn( 'state', seed ); % set the random seed
theta = zeros( L , T ); % Init storage space for our samples
theta(:,1) = randperm( L ); % Random ordering to start with

%%% Start sampling

% Our proposal is the last ordering but with two items switched
lasttheta = theta(:,t-1); % Get the last theta
% Propose two items to switch
whswap = randperm( L ); whswap = whswap(1:2);
theta_star = lasttheta;
theta_star( whswap(1)) = lasttheta( whswap(2));
theta_star( whswap(2)) = lasttheta( whswap(1));

% calculate Kendall tau distances
dist1 = kendalltau( theta_star , omega );
dist2 = kendalltau( lasttheta , omega );

% Calculate the acceptance ratio
pratio = exp( -dist1*lambda ) / exp( -dist2*lambda );
alpha = min( [ 1 pratio ] );
if u < alpha % Do we accept this proposal?
    theta(:,t) = theta_star; % proposal becomes new theta
else
    theta(:,t) = lasttheta; % copy old theta
end
% Occasionally print the current state
if mod( t,10 ) == 0
    fprintf( 't=%3d\n', t );
    for j=1:L
        fprintf( '%15s', labels{ theta(j,t)} );
    end
    fprintf( '\n' );
end
end

Listing 2.7: Implementation of componentwise Metropolis sampler for bivariate normal
%% Parameters of the Bivariate normal
mu = [ 0 0 ];
sigma = [ 1 0.3; 0.3 1 ];

%% Initialize the Metropolis sampler
T = 5000; % Set the maximum number of iterations
propsigma = 1; % standard dev. of proposal distribution
thetamin = [ -3 -3 ]; % define minimum for theta1 and theta2
thetamax = [ 3 3 ]; % define maximum for theta1 and theta2
seed=1; rand( 'state', seed ); randn( 'state',seed ); % set the random seed
theta = zeros( 2 , T ); % Init storage space for our samples
theta(1,1) = unifrnd( thetamin(1) , thetamax(1) ); % Start value for theta1
theta(2,1) = unifrnd( thetamin(2) , thetamax(2) ); % Start value for theta2

%% Start sampling
t = 1;
while t < T % Iterate until we have T samples
    t = t + 1;
    % Propose a new value for theta1
    theta_star = normrnd( theta(1,t-1) , propsigma );
    pratio = mvnpdf( [ theta_star theta(2,t-1) ] , mu , sigma ) / ...
    mvnpdf( [ theta(1,t-1) theta(2,t-1) ] , mu , sigma );
    alpha = min( [ 1 pratio ] ); % Calculate the acceptance ratio
    u = rand; % Draw a uniform deviate from [ 0 1 ]
    if u < alpha % Do we accept this proposal?
        theta(1,t) = theta_star; % proposal becomes new value for theta1
    else
        theta(1,t) = theta(1,t-1); % copy old value of theta1
    end

    % Propose a new value for theta2
    theta_star = normrnd( theta(2,t-1) , propsigma );
    pratio = mvnpdf( [ theta(1,t) theta_star ] , mu , sigma ) / ...
    mvnpdf( [ theta(1,t) theta(2,t-1) ] , mu , sigma );
    alpha = min( [ 1 pratio ] ); % Calculate the acceptance ratio
    u = rand; % Draw a uniform deviate from [ 0 1 ]
    if u < alpha % Do we accept this proposal?
        theta(2,t) = theta_star; % proposal becomes new value for theta2
    else
        theta(2,t) = theta(2,t-1); % copy old value for theta2
    end

end

%% Display histogram of our samples
figure( 1 ); clf;
subplot( 1,2,1 );
nbins = 12;
thetabins1 = linspace( thetamin(1) , thetamax(1) , nbins );
thetabins2 = linspace( thetamin(2) , thetamax(2) , nbins );
2.7 Gibbs Sampling

A drawback of the Metropolis-Hastings and rejection samplers is that it might be difficult to tune the proposal distribution. In addition, a good part of the computation is performed producing samples that are rejected and not used in the approximation. The Gibbs sampler is a procedure in which all samples are accepted, leading to much improved computational efficiency. However, the Gibbs sampler can only be applied in situations where we know the full conditional distributions of each component in the multivariate distribution conditioned on all other components. In some cases, these conditional distributions are simply not known, and Gibbs sampling cannot be applied. However, in many Bayesian models, distributions are used that lend themselves to Gibbs sampling.

To illustrate the Gibbs sampler, we look at the bivariate case where we have a joint distribution \( f(\theta_1, \theta_2) \). The key requirement for the Gibbs sampler is that we can derive the two conditional distributions \( f(\theta_1 | \theta_2 = \theta_2^{(t)}) \) and \( f(\theta_2 | \theta_1 = \theta_1^{(t)}) \) – that is, the distribution of each variable conditioned on a specific realization of the other variable. It is also required that we can sample from these distributions. We first initialize the sampler with some suitable values for \( \theta_1^{(1)} \) and \( \theta_2^{(1)} \). At each iteration \( t \), we follow steps that are very similar to the componentwise MH sampler. In the first step, we first sample a new value for \( \theta_1^{(t)} \) that is conditional on \( \theta_2^{(t-1)} \), the previous state of the other component. We do this by sampling a proposal from \( f(\theta_1 | \theta_2 = \theta_2^{(t-1)}) \). As opposed to Metropolis-Hastings, we will always accept this proposal so the new state can be immediately updated. In the second step, we sample a new value for \( \theta_2^{(t)} \) that is conditional on \( \theta_1^{(t)} \), the current state of the other component. We do this by sampling a proposal from \( f(\theta_2 | \theta_1 = \theta_1^{(t)}) \). Therefore, the procedure involves iterative conditional sampling – we keep going back and forth by sampling the new state of a variable by conditioning on the current values for the other component. Here is a summary...
of the Gibbs sampling procedure:

1. Set \( t = 1 \)

2. Generate an initial value \( \mathbf{u} = (u_1, u_2) \), and set \( \mathbf{\theta}(t) = \mathbf{u} \)

3. Repeat
   
   \( t = t + 1 \)
   
   Sample \( \theta_1^{(t)} \) from the conditional distribution \( f(\theta_1 | \theta_2 = \theta_2^{(t-1)}) \)
   
   Sample \( \theta_2^{(t)} \) from the conditional distribution \( f(\theta_2 | \theta_1 = \theta_1^{(t)}) \)

4. Until \( t = T \)

**Example 1.** In Example 3 from the previous section, we applied a Metropolis sampler for the bivariate normal distribution. This distribution can be sampled more efficiently using Gibbs sampling (although still not as efficiently as with direct methods). We start with \( \mathbf{\theta} = (\theta_1, \theta_2) \) having a bivariate normal distribution

\[
\mathbf{\theta} \sim \text{Norm}(\mathbf{\mu}, \mathbf{\Sigma})
\]  

(2.15)

with mean vector \( \mathbf{\mu} = (\mu_1, \mu_2) \) and covariance matrix \( \mathbf{\Sigma} \). For this example, let’s assume the following covariance structure:

\[
\mathbf{\Sigma} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}
\]  

(2.16)

Using analytic methods, we can derive the conditional distributions for each component of \( \mathbf{\theta} \) conditional on a realization for the other component:

\[
\theta_1 \sim \text{Norm}(\mu_1 + \rho(\theta_2 - \mu_2), \sqrt{1 - \rho^2})
\]  

(2.17)

\[
\theta_2 \sim \text{Norm}(\mu_2 + \rho(\theta_1 - \mu_1), \sqrt{1 - \rho^2})
\]  

(2.18)

This leads to the following Gibbs sampling procedure:

1. Set \( t = 1 \)

2. Generate an initial value \( \mathbf{u} = (u_1, u_2) \), and set \( \mathbf{\theta}(t) = \mathbf{u} \)

3. Repeat
   
   \( t = t + 1 \)
   
   Sample \( \theta_1^{(t)} \) from \( \text{Norm}(\mu_1 + \rho(\theta_2^{(t-1)} - \mu_2), \sqrt{1 - \rho^2}) \)
   
   Sample \( \theta_2^{(t)} \) from \( \text{Norm}(\mu_2 + \rho(\theta_1^{(t)} - \mu_1), \sqrt{1 - \rho^2}) \)

4. Until \( t = T \)

Figure 2.6 shows the results of a simulation with \( \mathbf{\mu} = (0, 0) \) and \( \rho = 0.7 \). The sampler was run with a single chain for 5000 iterations. Instead of producing 3D histograms as in Figure 2.7, this figure just shows a scatter plot of all individual samples (right panel) and a progression of the sampler state for the first 20 iterations (left panel).
Figure 2.6: Results of a Gibbs sampler for the bivariate normal distribution. Left panel shows the progression of the sampler state for the first 20 iterations. Right panel shows a scatterplot of 5000 sampled values.

Exercises

1. Implement the Gibbs sampler from Example 1 in Matlab and produce an illustration similar to Figure 2.6. You can use the parameters $\mu = (0, 0)$ and $\rho = 0.7$. What happens when $\rho = -0.7$?

2. Compare the output you get from the Gibbs sampler with random samples that are drawn from Matlab’s own multivariate normal random number generator (using command `mvnrnd`).
Graphical models are a general class of probabilistic models that allow researchers to reason with uncertain data in the presence of latent variables. Latent variables in cognitive science can include any information that is unobservable, such as attentional states, knowledge representations, contents of memory, and brain states. In short, anything that is relevant to explain the observed data but was itself not observed, can become a latent variable in a graphical model.

In a graphical model, the researcher posits certain local statistical dependencies between the random variables that correspond to the latent variables and observed data. We will restrict ourselves to directed graphical models, in which the statistical dependency between random variables is based on directional relationships. These models are also known as Bayesian networks and belief networks, although these terms all have somewhat different connotations. Another class of graphical models, not discussed here, are undirected graphical models, such as Markov random fields. These models are useful when it is natural to express the relationship between variables using undirected relationships.

It is important to realize that graphical models form a very large class of models. Many models that are typically not described as graphical models can be reinterpreted within a graphical modeling framework. For example, many forms of neural networks can be usefully analyzed with graphical models. Similarly, many process models proposed by cognitive psychologists often are framed as stochastic processes that can be couched as graphical models.

In this chapter, we will first review some basic rules and concepts related to probability theory. We will then go into the important concepts of graphical models that are needed to solve inference problems.

### 3.1 A Short Review of Probability Theory

There are a few basic rules that form the basis for all probabilistic machinery to solve inference problems in graphical models. Here, we just briefly review these rules. For an in depth introduction into probability theory, you can consult a number of books and online resources.
resources.

In the sum rule, we can write the marginal probability of $x$ as a sum over the joint distribution of $x$ and $y$ where we sum over all possibilities of $y$,

$$ p(x) = \sum_y p(x, y). \quad (3.1) $$

In the product rule, we can rewrite a joint distribution as a product of a conditional and marginal probability

$$ p(x, y) = p(y|x)p(x) = p(x|y)p(y). \quad (3.2) $$

In the chain rule, the product rule is applied repeatedly to give expressions for the joint probability involving more than two variables. For example, the joint distribution over three variables can be factorized into a product of conditional probabilities:

$$ p(x, y, z) = p(x|y, z)p(y, z) = p(x|y) p(y|z) p(z). \quad (3.3) $$

From these rules, we can obtain Bayes rule, which plays a central role in graphical models:

$$ p(y|x) = \frac{p(x|y)p(y)}{p(x)} = \frac{p(x|y)p(y)}{\sum_{y'} p(x|y')p(y')} \quad (3.4) $$

Importantly, using approximate inference techniques, we rarely need to explicitly evaluate the denominator $p(x)$. Instead, we can just evaluate the conditional probability up to a constant of proportionality:

$$ p(y|x) \propto p(x|y)p(y) \quad (3.5) $$

Finally, there are the concepts of independence and conditional independence. Two variables are said to be independent if their joint distribution factorizes into a product of two marginal probabilities:

$$ p(x, y) = p(x)p(y). \quad (3.6) $$

Note that if two variables are uncorrelated, that does not mean they are statistically independent. There are many ways to measure statistical association between variables and correlation is just one of them. However, if two variables are independent, this will ensure there is no correlation between them. Another consequence of independence is that if $x$ and $y$ are independent, the conditional probability of $x$ given $y$ is just the probability of $x$:

$$ p(x|y) = p(x). \quad (3.7) $$
In other words, by conditioning on a particular $y$, we have learned nothing about $x$ because of independence. Two variables $x$ and $y$ and said to be \textit{conditionally independent} of $z$ if the following holds:

$$p(x, y | z) = p(x | z) p(y | z).$$

Therefore, if we learn about $z$, $x$ and $y$ become independent. Another way to write that $x$ and $y$ are conditionally independent of $z$ is:

$$p(x | z, y) = p(x | z).$$

In other words, if we condition on $z$, and now also learn about $y$, this is not going to change the probability of $x$. It is important to realize that conditional independence between $x$ and $y$ does not imply independence between $x$ and $y$. We will see some examples of the difference between independence and conditional independence in the context of Bayesian networks.

\section*{Exercises}

1. Use the chain rule to give the general factorization of a joint distribution involving 4 variables.

\section*{3.2 The Burglar Alarm Example}

In the following example, we will give a classic example of a graphical model from the AI and Machine Learning literature: the burglar alarm network. This example has been described in many sources, and originates from Judea Pearl’s early work on reasoning with uncertainty.

In this example, consider a burglar alarm installed in your home. This alarm is set off somewhat reliably by the presence of a burglar. However, it can also be set off by an earthquake. Currently, you are traveling in your car so you cannot directly hear the alarm. However, your neighbor has said that she will call you on your cell phone if she hears the
alarm. Finally, while you are traveling in your car, you are also listening to the news on the radio and presumably, if there was an earthquake, you would probably hear about it on the radio. Suppose you now get a call from your neighbor, what is the probability that there is a burglar in your house?

This example sets up a number of issues on how we should reason with uncertainty when there are multiple variables involved. To model this situation, we can use a graphical model as shown in Figure 3.1. There are four random variables in this example: earthquake, burglar, alarm, neighbor call and radio report. For simplicity, let’s assume these are binary variables; either the alarm goes off or it doesn’t, the neighbor calls, or she doesn’t, etc. The illustration also shows how we can replace these words by letter codes (e.g., e, b, etc).

In the graphical model, the nodes are random variables and arrows indicate conditional dependencies between variables. For example, the network encodes the intuition that the status of the alarm depends on earthquakes and burglars, and that the neighbor’s call depends on the alarm. It is useful to think of these conditional dependencies as causal relationships between variables – the burglar might cause the alarm to go off, which in turn might cause your neighbor to call. This is how Judea Pearl originally thought of graphical models – as ways to reason probabilistically about causes and effects. However, you should keep in mind that graphical models can also be constructed in the absence of any causal interpretation.

The most important idea behind graphical models is that the graph structure implies a number of independence and conditional independence relations. For example, the illustration shows that we have assumed that earthquakes are independent of burglaries – one event doesn’t cause the other event. In other words, we have implied \( p(e, b) = p(e)p(b) \). Note that this assumption might not be true in the real world, but this is assumed in the network. Also, we have assumed that it is the alarm that might cause the neighbor to call, and not the burglary itself or an earthquake event. This implies a conditional independence relationship – we have assumed that the neighbor call is independent of the burglary event, conditional on knowing something about the alarm. In other words, we have assumed that if one already knows about the alarm, knowing about the burglar will not change the probability of the neighbor calling. Specifically, the network implies the following conditional independence relationship: \( p(c|a,b) = p(c|a) \).

### 3.2.1 Conditional probability tables

Up to this point, we haven’t said anything about the exact functional relationship between variables. There are many ways to parametrize the conditional probability relationships in graphical models. In our example, we have only binary variables with outcomes that can be represented by 1 (“true”) and 0 (“false”). One simple way to parametrize the relationships is by conditional probability tables. This is illustrated in Figure 3.2. For the root nodes in the network, we only have to specify the prior probabilities of each outcome. We have assumed that the probability of an earthquake (a priori) is .002. Similarly, we have assumed that the probability of a burglary is 0.001. For the alarm event, we need to specify a conditional probability that depends only on the parent nodes. We have assumed that the alarm
might still go off even in the absence of earthquakes or burglaries (with probability 0.01). Furthermore, we have assumed that an earthquake by itself might trigger the alarm (with probability 0.29), that a burglary by itself more reliably triggers the alarm (with probability 0.94) and that the combination of an earthquake and burglary raises the probability to 0.98. Furthermore, we have assumed that our neighbor is likely to call (with probability 0.90) if the alarm goes off, but not always. In addition, with probability 0.05, the neighbor can call even if there is no alarm (the neighbor might be calling about something else).

With all the conditional probabilities specified, we can now engage in probabilistic inference. We can calculate the probability of one event if we know about the outcomes of other events. We can reason from causes to effects but also from effects to causes. Because our example is so simple, we can use exact methods to calculate any probability of interest, just by using some standard rules in probability theory, such as the sum, product and Bayes rule.

For example, suppose we want to calculate the probability that the alarm is triggered knowing that the burglary is taking place. In other words, we need to know

\[ p(a = 1 | b = 1) \]

Note that in this question, we have not said anything about earthquakes, another potential trigger for the alarm and therefore, we need to sum over all possibilities regarding earthquakes.

Using the sum and product rules, we can calculate this by

\[
p(a = 1 | b = 1) = p(a = 1 | b = 1, e = 0)p(e = 0) + p(a = 1 | b = 1, e = 1)p(e = 1)
\]

\[
= (0.94)(0.998) + (0.98)(0.002)
\]

\[
= 0.9401
\]

Suppose we hear a radio report on an earthquake, what is the probability that an earthquake happened, i.e., \( p(e = 1 | r = 1) \)? Note that this probability is not 1 because we have assumed that earthquakes are likely but not guaranteed to be reported. Note also that we cannot
directly read off this probability from the conditional probability table because the table shows the probability of a report conditional on an earthquake and not the other way around. In this case, we can use Bayes rule to “flip” the conditional probabilities as follows:

\[
p(e = 1|r = 1) = \frac{p(r = 1|e = 1)p(e = 1)}{p(r = 1|e = 1)p(e = 1) + p(r = 1|e = 0)p(e = 0)}
\]

\[
= \frac{(.992)(0.002)}{(.992)(0.002) + (.001)(.998)}
\]

\[
= .6653
\]

**Exercises**

1. Calculate the probability that the alarm is not triggered if a burglary is taking place: \(p(a = 0|b = 1)\). For this and the following exercises, show how you derived the answer, and do not just give a single numeric answer.

2. Calculate the probability that you get a call from your neighbor if a burglary is taking place: \(p(c = 1|b = 1)\). Note that the answer from the previous exercise will be part of this calculation.

3. Calculate the probability that there is a burglary if the alarm is triggered: \(p(b = 1|a = 1)\).

4. Are the two events, radio report and burglary independent?

** 5 Calculate the probability that there is a burglary if you get a call from your neighbor: \(p(b = 1|c = 1)\).

**3.2.2 Explaining away**

It is important to realize that if two events are a priori independent, that does not necessarily mean that they are still independent if we gain additional knowledge about another event. In other words, two events that are independent might become conditionally dependent on another outcome. This leads to the explaining away phenomenon.

For example, in the burglar alarm network, the earthquake and burglary events are independent, a priori. Suppose we now learn that the alarm has indeed been triggered. Are the earthquake and burglary events still independent conditional on this extra bit of information? It turns out they are not. To see this, suppose we also know that an earthquake has taken place. This will in fact lower the probability of a burglary. Intuitively, the earthquake explains away the fact that the alarm has been triggered. Similarly, if we know that an earthquake has definitely not happened, this will raise the probability of the burglary event.

More formally, in this network, we have independence \(p(b|e) = p(b)\) but not conditional independence \(p(b|e, a) \neq p(b|a)\).
CHAPTER 3. GRAPHICAL MODELS

Exercises

1. Calculate the probabilities \( p(b = 1|a = 1, e = 1) \) and \( p(b = 1|a = 1) \). Note that the last probability was already calculated in a previous exercise. Verify that the first probability is much lower than the second probability.

3.2.3 Joint distributions and independence relationships

The directed relationships between random variables in a graphical model can be translated to a joint distribution over all random variables. Suppose we have a network with \( K \) variables \( x_1, x_2, \ldots, x_K \), the joint distribution can be calculated by:

\[
p(x_1, \ldots, x_K) = \prod_{k=1}^{K} p(x_k|p_{a_k})
\]  

(3.10)

where \( p_{a_k} \) represents the realizations of variables that are parents of the \( k \)th node in the network. Therefore, the joint distribution can be factorized as a product of probability of each node conditional on its parents. Note that if a node \( k \) has no parents (it is a root node), the conditional probability \( p(x_k|p_{a_k}) \) becomes \( p(x_k) \). We can turn a particular factorization of a joint distribution into a graphical representation. The key here is that sometimes, it is useful to visually inspect the graph structure to understand the properties of a graphical model. At other times, it is useful to inspect the implied factorization of the joint distribution.

The directed graphs we are considering are subject to an important restriction. There cannot be any directed cycles in the graph, whereby following the arrows from one node leads us back to the same node. In other words, there cannot be any loops in the graph. Such graphs are called directed acyclic graphs or DAGs.

For the alarm example, the joint distribution factorizes as follows:

\[
p(a, b, c, e, r) = p(c|a)p(a|e, b)p(r|e)p(e)p(b).
\]

Therefore, the probability of a particular set of outcomes can easily be evaluated. If we want to know the probability of getting the neighbors call while the alarm is triggered and there is an earthquake but no burglary and there is an earthquake report on the radio, we can write

\[
p(a = 1, b = 0, c = 1, e = 1, r = 1) \\
= p(c = 1|a = 1)p(a = 1|e = 1, b = 0)p(r = 1|e = 1)p(e = 1)p(b = 0) \\
= (0.90)(0.29)(0.992)(0.002)(0.999) \\
= 0.0005173.
\]

For small networks, it is relatively easy to just visually read off the independence relationships from a network. Instead of visual inspection, one can also use the joint distribution to derive all the conditional independence relations. For example, consider the graphical model in Figure 3.3, panel C. Suppose we want to know whether \( x \) and \( y \) are independent
Figure 3.3: Four example graphical models

conditional on $z$, such that $p(x, y|z) = p(x|z)p(y|z)$. We can use the product rule to write the conditional probability $p(x, y|z)$ as follows:

$$p(x, y|z) = \frac{p(x, y, z)}{p(z)}$$

We can replace the joint distribution in the numerator by the factorization that the network implies:

$$p(x, y|z) = \frac{p(x, y, z)}{p(z)}$$
$$= \frac{p(x|z)p(y|z)p(z)}{p(z)}$$
$$= p(x|z)p(y|z)$$

Therefore, the conditional independence holds in this case.

**Exercises**

1. Consider the graphical model in Figure 3.3, panel A. Is this a directed acyclic graph? Give the factorization of the joint probability.

2. For the graphical models B and D in Figure 3.3, determine if $x$ and $y$ are conditionally independent of $z$. Also, in what cases are $x$ and $y$ independent?
3.3 Graphical Model Notation

With the alarm example, we have already seen some of the basic ways in which we can represent conditional distributions as directed links between nodes in a graph. In this section, we will elaborate the graphical model notation that is needed to represent most probabilistic models. Unfortunately, graphical model notation is not fully standardized, and different authors use slightly different notations.

Suppose we have a situation where we have observed four outcomes in our experiment: $y_1$, $y_2$, $y_3$, and $y_4$. We now create a model for our data that incorporates our beliefs about how the data was generated. In this simple example, suppose we believe that the four observed outcomes were all generated by a $\text{Normal}(\mu, \sigma)$ distribution with the same mean $\mu$ and standard deviation $\sigma$. The corresponding graphical model is shown in Figure 3.4, panel A. The nodes corresponding to our observations are shaded in gray. The nodes corresponding to the latent variables are not shaded. This is an important notational convenience that helps to visualize the relationships between observed and latent variables.

In hierarchical models where the same sampling steps are repeated many times over, it becomes unwieldy to visualize represent the full tree that includes each individual sampling step. We can simplify this using plates. Figure 3.4, panel B, shows how the model in panel A can be represented more compactly using plate notation. The plate represents a repeated sampling step where the subscript $i$ indexes the particular observed value of $y_i$.

Another notational convenience is to distinguish between variables that have probabilistic or deterministic relationships with their parent variables. For example, suppose that in our experiment, we are really interested in the precision $\tau$ of the normal distribution, where $\sigma = \sqrt{1/\tau}$. Figure 3.4, panel C shows how we can include this in our model. We use a double-bordered node for $\sigma$ to indicate that the variable is deterministic. Therefore, if we know $\tau$, the value of $\sigma$ is automatically determined (and vice versa) – there is no uncertainty associated with their relationship. In this particular example, we could have drawn an arrow from $\tau$ directly to $y_i$ and have omitted the node for $\sigma$ altogether. However, in many cases, it is useful (and necessary) to specifically include any deterministic variables in the model.

In order to communicate graphical models to other researchers, the graphs help to visualize the dependencies between variables but do not provide the exact probabilistic or functional dependence between variables. To do this, we can use probability notation such as $y_i \sim \text{Normal}(\mu, \sigma)$, where the “$\sim$” symbol can be read as “distributed as”. This is equivalent to writing $p(y_i) = \text{Normal}(y_i|\mu, \sigma)$ or $p(y_i|\mu, \sigma) = \text{Normal}(y_i|\mu, \sigma)$ to explicitly represent the conditional dependence on $\mu$ and $\sigma$.

3.3.1 Example: Consensus Modeling with Gaussian variables

Suppose we have an experiment where we have $M$ individuals participating in an estimation task. In this estimation task, individuals are given questions where the answers consist of a single number. For example, they might be asked to estimate the number of dots in a display, or give an answer to a general knowledge question such as “how many people live in
Irvine, California”. Although we as experimenters know the true answer to each question, the goal of the modeling is to estimate the true answer \( \mu \) on the basis of all the estimates given by individuals. This setup is similar to Cultural Consensus models developed by Batchelder and colleagues, except that we apply this to questions with continuous as opposed to discrete answers (such as true/false or multiple choice questions). In this consensus model, we are assuming that each individual is giving \( N \) repeated estimates for the same question. Therefore, the observed data consists of estimates \( y_{ij} \) where \( i = 1, \ldots, N \) indexes the repetition and \( j = 1, \ldots, M \) indexes the individual. In the model for this task, we assume that all estimates are Normally distributed centered around the mean \( \mu \), which is a latent variable in the model. We also assume that each individual is associated with a separate standard deviation \( \sigma_j \). This allows the model to associate different levels of uncertainty for each individual. To complete the model, we put a Gamma\((a,b)\) prior on \( \tau_j \), which is the precision associated with individual \( j \). We map from precision to standard deviation simply by \( \sigma_j = 1/\sqrt{\tau_j} \). This leads to the following model:

\[
\begin{align*}
y_{ij} & \sim \text{Norm}(\mu, \sigma_j) \\
\sigma_j & = 1/\sqrt{\tau_j} \\
\tau_j & \sim \text{Gamma}(a,b)
\end{align*}
\]

where \( a \) and \( b \) are hyperparameters in the model. The associated graphical model is shown in Figure 3.5. Note that there are two plates in the model. The inner plate indicates the repeated samples for a given individual, and the outer plate indicates the sampling across individuals. The fact that plates can be nested is quite helpful to create compact representations for hierarchical models.

Figure 3.4: Examples of graphical models where shaded and unshaded nodes indicate observed and unobserved variables respectively. The models in panel A are B are identical. The plate notation in panel B provides a more compact representation of models with repeated sampling steps. Panel C illustrates a deterministic variable as indicated by the double-bordered node.
CHAPTER 3. GRAPHICAL MODELS

Figure 3.5: A consensus model for normally distributed data.

Exercises

1. Suppose in the model in Figure 3.4, panel A, the experimenter lost the observed value $y_3$ because of a programming error. How would this change the graphical model?

2. Write down the joint distribution $p(y_1, y_2, y_3, y_4, \mu, \sigma)$ for the model in Figure 3.4, panel B (equivalent to the model in panel A). Instead of writing one long sequence of terms, it is helpful to use product notation (i.e., $\prod$)

3. Write down the joint distribution $p(\mu, \tau_j, y_{11}, y_{12}, \ldots, y_{NM})$ for the consensus model in Figure 3.11. Again, use product notation.

4. For the consensus model in the example, how would the graphical model change if the means are not latent variables but are observed instead?

5. The consensus model in the example models the situation for just a single question. Suppose we want to expand the model to $K$ separate questions, where each question is associated with a single latent truth $\mu_k$. To simplify the model, we assume that each individual has an uncertainty $\sigma_j$ regardless of what question is asked. Visualize the corresponding graphical model.

6. Suppose we have the following model:

$$
\begin{align*}
  z_i &\sim \text{Bernoulli}(1/2) \\
  \phi_0 & = 1/2 \\
  \phi_{i1} & \sim \text{Gaussian}(\mu, \lambda) \\
  \mu & \sim \text{Uniform}(0.5, 1) \\
  \lambda & \sim \text{Gamma}(0.001, 0.001) \\
  \theta_i & = \begin{cases}
    \phi_0 & z_i = 0 \\
    \phi_{i1} & z_i = 1
  \end{cases} \\
  k_i & \sim \text{Binomial}(\theta_i, n)
\end{align*}
$$

Write down the corresponding graphical model when $k_i$ and $n$ are the only observed variables, and $i = 1, \ldots, p$. Note that this model was taken from the coursebook by Lee and Wagenmakers.
Chapter 4

Approximate Inference in Graphical Models

A standard application of graphical models is to find the distributions for the latent variables that might explain the observed data. In a Bayesian framework, this corresponds to estimating the posterior distribution over the latent variables. These inferred distributions can be used to test theoretical assumptions about some underlying process (e.g. a cognitive process). Another application of graphical models is to produce predictive distributions. This can be useful to simulate what the model believes a priori the data should look like, or how the model believes any future data should look like after learning about some observed data. Such simulations can be useful to the modeler to test the assumptions of some theoretical framework and to make predictions that can be tested with empirical research.

4.1 Prior predictive distributions

The *prior predictive* is the distribution the model predicts over the observed variables, before any data is considered. In other words, these are the predictions the model makes there is no observed data to condition on (see Figure 4.1A. This corresponds to a graphical model where the observed variables are turned (temporarily) into latent variables such that the whole graphical model consists of latent variables. This can be useful in a number of ways. In some cases, a model will predict data patterns that the researcher knows will either never occur in an experiment or occur only rarely. This can happen when the model is too complicated or when the researcher fails to incorporate some important constraints in the model. Another reason to consider the prior predictive distribution is to create synthetic datasets for testing purposes. Before applying any probabilistic model to real data, it is often helpful to do posterior inference on the model on data that were produced by itself. This helps to check the inference procedures that were used. If the inference works properly, the model should infer distributions over the latent variables that are similar to the ones that were sampled when producing the artificial data.

Specifically, let \( \mathbf{y} = (y_1, \ldots, y_N) \) represent the set of observed variables in a graphical
model. Let $\theta = (\theta_1, \ldots, \theta_M)$ be the set of latent variables in a model. The prior predictive distribution corresponds to:

$$p(y) = \int p(y, \theta) d\theta = \int p(\theta)p(y|\theta) d\theta$$  \hspace{1cm} (4.1)$$

Therefore, we integrate over all possible combinations of the latent variables and weight the predictions that flow from the latent variables with the prior probability of the latent variables. Note that integration is replaced by summation for discrete latent variables in the network.

In some cases, the integrals in Equation 4.1 can be evaluated analytically. In this section, we will focus instead on a simple procedure that approximates the prior predictive using Monte Carlo sampling. This procedure is based on ancestral sampling. The idea is very simple. To draw one sample from the prior predictive, we start by sampling from the root nodes in the network (these are the nodes that have no parents). If the root nodes are constants in the model set by the researcher, we don’t sample and keep those values. In the next step, we draw samples from the child nodes of the root nodes. In this case, we are drawing from distributions that are conditional on the sampled values of the parent nodes. On each subsequent step, we draw samples from nodes for which the parent nodes already were sampled and stop when all variables have an assigned sample. This process of ancestral sampling repeats for a number of iterations until we have drawn enough samples. The important aspect in ancestral sampling is to always start at the root nodes at the “top” of the network, and to work your way down to the “bottom” of the network, and at each step to only sample from a node whose parent nodes have been sampled. Note that this method is just based on Monte Carlo sampling and not Markov chain Monte Carlo. Therefore, there is no burnin period – all samples drawn in this procedure can be accepted as samples from the prior predictive distribution.
Figure 4.2: Data pattern sampled from the prior predictive distribution for the consensus model with normal distributions. The model used $\mu = 150$, $a = 0.3$, $b = 3.33$, $M = 15$ and $N = 20$. 
CHAPTER 4. APPROXIMATE INFERENCE IN GRAPHICAL MODELS

Exercises

1. Consider the alarm network discussed earlier and shown in 3.2. Suppose we want to calculate the prior predictive distribution for node \( c \), the neighbor call. Use the ancestral sampling method to draw 10,000 samples from the prior predictive distribution. On the basis of these samples, what is the predicted probability, a priori, of a neighbor call? (** what is the exact probability?).

2. Consider the consensus model described in Equation 3.11. Suppose that \( \mu = 150 \) and that the hyperparameters \( a \) and \( b \) are set to 0.3 and 3.33 respectively. Suppose that the number of individuals \( M = 15 \) and the number of repetitions per individual, \( N = 20 \). Intuitively, this corresponds to a situation where 15 individuals all give 20 independent estimates that are centered around the true answer of 150. Draw a single sample from the prior predictive distribution over \( y_{ij} \). You can visualize the results as in Figure 4.2. This figure shows some possible data pattern from the prior predictive (keep in mind that your results will look different because a different random seed is used). It shows that some individuals are highly variable in their estimates (corresponding to a high \( \sigma_j \) but all estimates are centered around 150. Save the resulting data pattern to a file. You’ll need it for a later exercise.

4.2 Posterior distributions

One of the most important uses for graphical models is to explain the observed data through a set of latent variables. In other words, the problem is to find distributions over latent variables such that they are likely to reproduce the observed data (see Figure 4.1B for a canonical example of this problem). In Bayesian modeling, this corresponds to the problem of finding the posterior distribution \( p(\theta | y) = p(\theta_1, \theta_2, \ldots, \theta_M | y_1, \ldots, y_N) \). In some simple graphical models, this can be calculated exactly using Bayes rule. For example, in the alarm network, it was relatively straightforward to calculate the probability of a burglary given a phone call from the neighbor. In other cases, message passing schemes can be used to calculate the exact posterior distribution. We will not go into these approaches in this chapter. Instead, we will discuss two sampling approaches.

4.2.1 Rejection Sampling

This approach is one the simplest methods to do posterior sampling and it only works when the observed variables are discrete. We mention this method primarily because of its simplicity. The idea is that one uses the ancestral sampling method to simulate the prior predictive distribution but the procedure is changed such that all samples that lead to values of the observed variables that are unequal to the actual observed values are discarded. When any mismatch occurs for any of the observed values, new samples need to be drawn. When the observed values are a priori very unlikely in the model, it leads to a highly inefficient approach because the majority of samples will be rejected.
Exercise

1. Consider again the alarm network. Suppose the observed data consists of node \( c \), the neighbor calling. Suppose we have just heard the phone ringing and it is the neighbor calling. Use rejection sampling to calculate the posterior probability of a burglary, \( p(b = 1|c = 1) \). Note that you can use the code you developed to draw samples from the prior predictive distribution. You just have to modify it to reject samples that lead to the outcome \( c = 1 \).

4.2.2 MCMC Sampling

We can consider the posterior distribution \( p(\theta|y) = p(\theta_1, \theta_2, \ldots, \theta_M|y_1, \ldots, y_N) \) as simply some multivariate distribution that needs to be sampled from. In Chapter 2, we have considered two approaches to sample from multivariate distributions: the Gibbs sampler and the componentwise Metropolis-Hastings sampler. We can use these approaches to sample from posterior distributions. The only difference is that now we are conditioning on observed data and that we can often simplify the sampling steps by making use of the conditional independence relationships in graphical models.

Two latent variables

To illustrate the general MCMC sampler, we look at a case involving two latent variables \( \theta_1 \) and \( \theta_2 \). We represent instantiations (i.e. samples) of these variables by \( v_1 \) and \( v_2 \) respectively. We represent the state of the sampler at iteration \( t \) with \( \theta_1^{(t)} \) and \( \theta_2^{(t)} \). For the first iteration, we initialize the sampler with some suitable values \( v_1 \) and \( v_2 \) and set the initial state \( \theta_1^{(1)} = v_1 \) and \( \theta_2^{(1)} = v_2 \). At each iteration \( t \), we first sample a new value \( v_1 \) for the first latent variable from the conditional distribution \( p(\theta_1|y, \theta_2 = v_2) \). We then update the state \( \theta_1^{(t)} = v_1 \) with the sampled value. In the second step, we sample a new value \( v_2 \) for the second latent variable from the conditional distribution \( f(\theta_2|y, \theta_1 = v_1) \). We then update the state \( \theta_2^{(t)} = v_2 \) with the sampled value. This completes one iteration of the sampler. Here is a summary of the sampling procedure for two latent variables:

1. Set \( t = 1 \)

2. Generate initial values \( v_1, v_2 \) for the two latent variables

3. Set the initial state \( \theta_1^{(t)} = v_1 \) and \( \theta_2^{(t)} = v_2 \)

4. Repeat
   \[
   t = t + 1
   \]
   Sample \( v_1 \sim p(\theta_1|y, \theta_2 = v_2) \)
   Sample \( v_2 \sim p(\theta_2|y, \theta_1 = v_1) \)
   Update the state \( \theta_1^{(t)} = v_1 \) and \( \theta_2^{(t)} = v_2 \)
5. Until $t = T$

**Multiple latent variables**

This procedure can be generalized to $M$ latent variables. In one iteration of the sampler, each latent variable $j$ is visited and assigned a value by drawing from the distribution of that variable conditional on the assignments to all other latent variables as well as the observed data. In our notation, we will write $-j$ to refer to all variables other than $j$. After all latent variables are assigned a value, we update the state $\theta^{(t)}$ with these values. Here is a description of the general MCMC sampling approach for posterior distributions:

1. Set $t = 1$
2. Generate initial values $v_1, v_2, \ldots, v_M$, for each latent variable
3. Set the initial state $\theta^{(1)} = v$
4. Repeat
   - $t = t + 1$
   - $j = 0$
   - Repeat
     - $j = j + 1$
     - Sample $v_j \sim p(\theta_j | y, \theta_{-j} = v_{-j})$
     - Update the state $\theta_j^{(t)} = v_j$
   - Until $j = M$
5. Until $t = T$

**Markov Blankets**

At the heart of the general MCMC approach for posterior inference is the conditional distribution $p(\theta_j | y, \theta_{-j} = v_{-j})$ where we sample a value for a latent variable conditioned on the instantiations of all other latent variables as well as the data. When calculating this distribution, one can take advantage of the conditional independence relationships in the graphical model. To illustrate this, look at the example graphical model in Figure 4.3, panel A. Suppose in our sampler, we are at the step where need to sample an assignment for the third latent variable $\theta_3$ by sampling from the conditional $p(\theta_3 | \theta_1, \theta_2, \theta_4, \theta_5, y_1, y_2)$. Using the product rule, we can write this as proportional to the joint distribution

$$p(\theta_3 | \theta_1, \theta_2, \theta_4, \theta_5, y_1, y_2) \propto p(\theta_3, \theta_1, \theta_2, \theta_4, \theta_5, y_1, y_2)$$

We can simplify the joint distribution using the general factorization in 3.10, giving us:

$$p(\theta_3 | \theta_1, \theta_2, \theta_4, \theta_5, y_1, y_2) \propto p(\theta_1)p(\theta_2)p(\theta_4)p(\theta_3 | \theta_1)p(\theta_5 | \theta_2)p(y_1 | \theta_3, \theta_5)p(y_2 | \theta_5, \theta_4)$$
Figure 4.3: Illustration of Markov blankets for an example graphical model (A). The Markov blankets for $\theta_3$ and $\theta_5$ are shown in (B) and (C) respectively.

Now, note that we are looking for a distribution for $\theta_3$. Any term on the right-hand side that does not depend on $\theta_3$ can be removed from the equation, leading to

$$p(\theta_3|\theta_1, \theta_2, \theta_4, \theta_5, y_1, y_2) \propto p(\theta_3|\theta_1)p(y_1|\theta_3, \theta_5)$$

On the right-hand side, we only have the terms $\theta_3$, $\theta_1$, $\theta_5$, and $y_1$. All other latent variables and observed data are irrelevant in this conditional distribution. We call the variables that are relevant for the conditional distribution the Markov blanket. Figure 4.3, panel B shows the Markov blanket for $\theta_3$. We can use similar reasoning to derive the Markov blanket for $\theta_5$, as shown in panel C.

Instead of going through these calculations each time to determine the Markov blanket, we can use a simple definition. The Markov blanket for a node includes all parents, all children and all parents of these children.

4.2.3 Example: Posterior Inference for the consensus model with normally distributed variables

To give an example of a posterior inference procedure, we revisit the consensus model discussed earlier. The model is

$$y_{ij} \sim \text{Normal}(\mu, 1/\sqrt{\tau_j})$$

$$\tau_j \sim \text{Gamma}(a, b)$$

where we have removed the latent variable $\sigma_j$ and instead directly placed the precision term $\tau_j$ into the normal distribution. Therefore, we have the latent variables $\tau_j$, where $j = 1, \ldots, M$, and the latent variable $\mu$. We will treat the variables $a$ and $b$ as constants in the model and we will not sample them. In our MCMC sampler, we will need to able to sample from a conditional distribution $p(\tau_j|y, \tau_{-j}, \mu, a, b)$ to get values for the individual precision variables. Using Markov blankets, we can derive that $p(\tau_j|y, \tau_{-j}, \mu) = p(\tau_j|y_{ij}, \ldots, y_{Nj}, \mu, a, b)$. We also
need to sample from a conditional distribution \( p(\mu | y, \tau, a, b) \). Again, using the conditional independence relations, we can derive that \( p(\mu | y, \tau, a, b) = p(\mu | y, \tau) \). This leads to the following general MCMC procedure where the state at iteration \( t \) consists of sampled values \( \tau^{(t)} \) and \( \mu^{(t)} \):

1. Set \( t = 1 \)
2. Generate initial values \( v_1, \ldots, v_M \) for the precision variables \( \tau \)
3. Set the initial state \( \tau^{(t)} = v \)
4. Generate initial value \( v_{M+1} \) for variable \( \mu \)
5. Set the initial state \( \mu^{(t)} = v_{M+1} \)
6. Repeat
   \[ t = t + 1 \]
   Sample \( v_1 \sim p(\tau_1 | y_{11}, \ldots, y_{N1}, \mu = v_{M+1}, a, b) \)
   Sample \( v_2 \sim p(\tau_2 | y_{12}, \ldots, y_{N2}, \mu = v_{M+1}, a, b) \)
   ...
   Sample \( v_M \sim p(\tau_M | y_{1M}, \ldots, y_{NM}, \mu = v_{M+1}, a, b) \)
   Update the state \( \tau^{(t)} = v \)
   Sample \( v_{M+1} \sim p(\mu | y, \tau = \tau^{(t)}) \)
   Update the state \( \mu^{(t)} = v_{M+1} \)
7. Until \( t = T \)

The next step is to figure out how to actually draw samples from these conditional distributions. One solution would be to do Metropolis-Hastings for each step. However, the structure of the model allows a Gibbs sampling solution where we can directly sample from the conditional distribution. Using conjugacy from Bayesian statistics (which we will discuss in a later section), the conditional distribution for the precision parameter is based on the following distribution:

\[
\tau_j | y_{1j}, \ldots, y_{Nj}, \mu, a, b \sim \text{Gamma} \left( a + \frac{N}{2}, \frac{1}{b + \sum_{i=1}^{N} (y_{ij} - \mu)^2 / 2} \right) \quad (4.3)
\]

We can also use conjugacy to derive a simple form for the conditional distribution for \( \mu \):

\[
\mu | y, \tau \sim \text{Normal} \left( \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} \tau_j y_{ij}}{N \sum_{j=1}^{M} \tau_j}, \frac{1}{\sqrt{N \sum_{j=1}^{M} \tau_j}} \right) \quad (4.4)
\]
CHAPTER 4. APPROXIMATE INFERENCE IN GRAPHICAL MODELS

These sampling equations can be used in the Gibbs sampler for the model. To test the Gibbs sampler, we sampled synthetic data using the prior predictive distribution. For this synthetic dataset, we used $\mu = 150$ and set the hyperparameters $a$ and $b$ to 0.3 and 3.33 respectively. We simulated the model with $M = 15$ individuals $N = 20$ repetitions per individual. Figure 4.4 shows the samples drawn from the posterior distribution over $\mu$ and $\tau$ for this synthetic dataset. Note that the first 100 samples were discarded and that the sampler was run for 400 iterations. Figure 4.5 shows the correlation between the inferred precision parameters (averaged over the last 300 iterations) and the precision parameters that were sampled in the prior predictive distribution. As you can observe, the model is able to (roughly) recover the precisions for each (simulated) individual.

Exercises

1. Write a Matlab program that implements the Gibbs sampler for the consensus model. The dataset can be based on the synthetic data you created in an earlier exercise. Use the parameter and sampler settings as specified above. Create visualizations that are similar to Figure 4.4 and Figure 4.5.

2. What is the inferred mean when you average the sampled values for the last 300 iter-
Figure 4.5: Correspondence between the inferred precisions for each (simulated) individual and the true values that were drawn in the prior predictive distribution. How does this inferred mean compare with the mean calculated by averaging over all observed data values? When would you expect to see a major difference?