Computational Statistics with Matlab

Mark Steyvers

May 13, 2011

Contents

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.

For this course however, you can hand in your answers in any way you see fit and sometimes, it might be useful to just hand in handwritten answers. [1]

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 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, such as 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.

[1] note: in this version, a new chapter 3 was inserted which is not fully completed yet

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 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 simpler 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 can operate efficiently with multivariate distributions.

1.1 Standard distributions

Some distributions are used so often, that they become part of a standard set of distributions supported by Matlab. 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, and to sample 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(μ, σ) distribution where $\mu = 100$ and $\sigma = 15$. To make things concrete, imagine that this distribution represents the observed variability of IQ coefficients in some population. The code shows how to display the probability density and the cumulative

Table 1.1: Examples of Matlab functions for evaluating probability density, cumulative density and drawing random numbers

Distribution	PDF	CDF	Random Number Generation
Normal	normpdf	normcdf	norm
Uniform (continuous)	unifpdf	unifcdf	unifrnd
Beta	betapdf	betacdf	betarnd
Exponential	exppdf	expcdf	exprnd
Uniform (discrete)	unidpdf	unidcdf	unidrnd
Binomial	binopdf	binocdf	binornd
Multinomial	mnpdf		mnrnd
Poisson	poisspdf	poisscdf	poissrnd

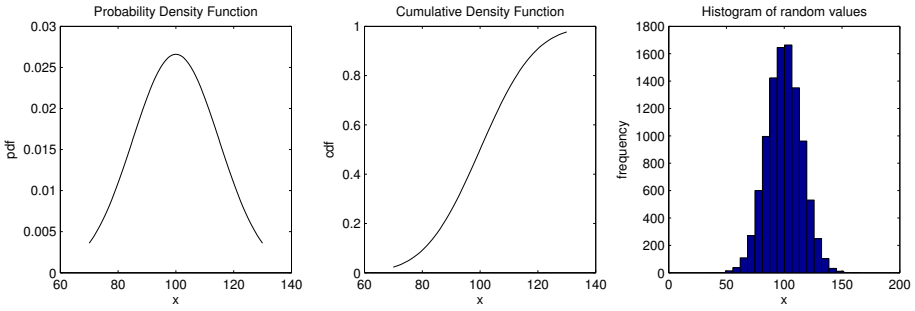

Figure 1.1: Illustration of the Normal(μ, σ) distribution where $\mu = 100$ and $\sigma = 15$.

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, θ) 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, θ, equals 0.7.

Exercises

1. Adapt the Matlab program in Listing 1.1 to illustrate the Beta(α, β) distribution where $\alpha = 2$ and $\beta = 3$. Similarly, show the Exponential(λ) distribution where $\lambda = 2$.

2. Adapt the matlab program above to illustrate the Binomial(N, θ) distribution where $N = 10$ and $\theta = 0.7$. Produce an illustration that looks similar to Figure 1.2.

3. Write a demonstration program to sample 10 values from a Bernoulli(θ) distribution with $\theta = 0.3$. Note that the Bernoulli distribution is one of the simplest discrete distri-

Listing 1.1: Matlab code to visualize Normal distribution.

```
%% 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' );
```

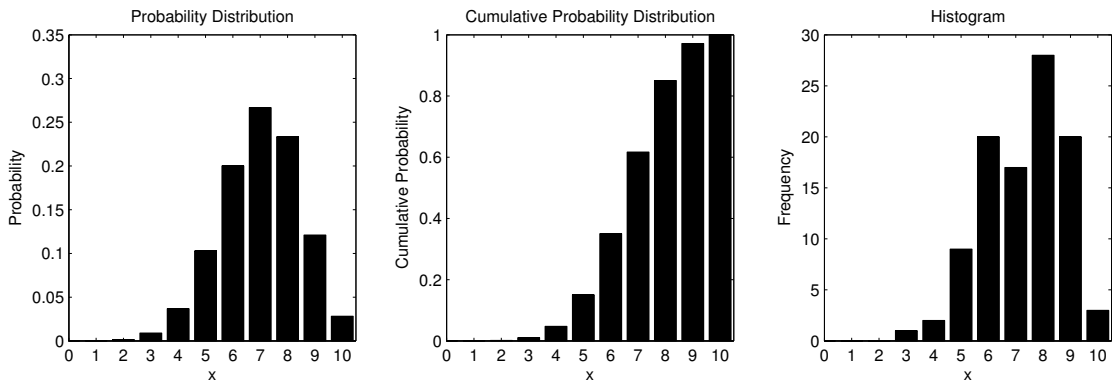

Figure 1.2: Illustration of the Binomial(N, θ) distribution where $N = 10$ and $\theta = 0.7$.

butions to simulate. There are only two possible outcomes, 0 and 1. With probability θ, the outcome is 1, and with probability $1 - \theta$, the outcome is 0. In other words, $p(X = 1) = \theta$, and $p(X = 0) = 1 - \theta$. 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 true/false questions, 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:

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

   Note that there are more advanced methods to seed the random number generator in Matlab using `RandStream` methods. For most applications in this course, such methods are not needed.

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 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 on 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 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 a 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 are 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).

X	$p(X)$	$F(X)$
0	0.000	0.000
1	0.100	0.100
2	0.090	0.190
3	0.095	0.285
4	0.200	0.485
5	0.175	0.660
6	0.190	0.850
7	0.050	0.900
8	0.100	1.000
9	0.000	1.000

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 `randsample`. The code produces the illustration shown in Figure 1.2.1.

Instead of using the built-in functions such as `randsample` or `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 $F(X = x) = p(X <= x)$. For discrete distributions, this can be done using simple summation. The cumulative probabilities of our example are shown in the right column of Table 1.2.1. In the inverse transform 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 is smaller than (or is equal to) the associated cumulative probability corresponds to the sampled outcome. Figure 1.2.1 shows an example with a uniform random deviate of $U = 0.8$ that leads 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.

```
% Simulate the distribution observed in the
% human random digit generation task

% probabilities for each digit
theta = [0.000;  ... % digit 0
         0.100;  ... % digit 1
         0.090;  ... % digit 2
         0.095;  ... % digit 3
         0.200;  ... % digit 4
         0.175;  ... % digit 5
         0.190;  ... % digit 6
         0.050;  ... % digit 7
         0.100;  ... % digit 8
         0.000 ] ...  % digit 9

% fix the random number generator
seed = 1; rand( 'state' , seed );

% let's say we draw K random values
K = 10000;
digitset = 0:9;
Y = randsample(digitset,K,true,theta);

```

```
% create a new figure
figure( 1 ); clf;

% 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' );
```

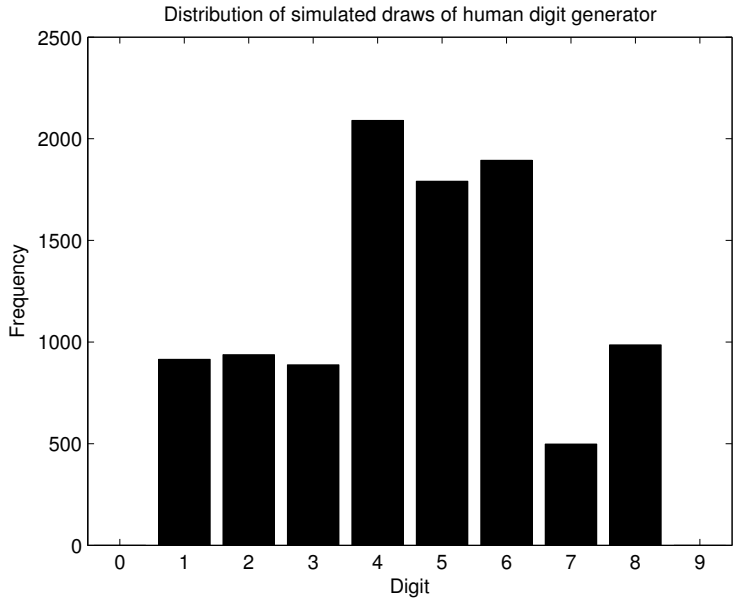

Figure 1.3: Illustration of the BINOMIAL(N, θ) distribution where $N = 10$ and $\theta = 0.7$.

Exercises

1. Create the Matlab program that implements the inverse tranform 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. 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`. Show how to use this function to sample digits according to the probabilities shown in Table 1.2.1.

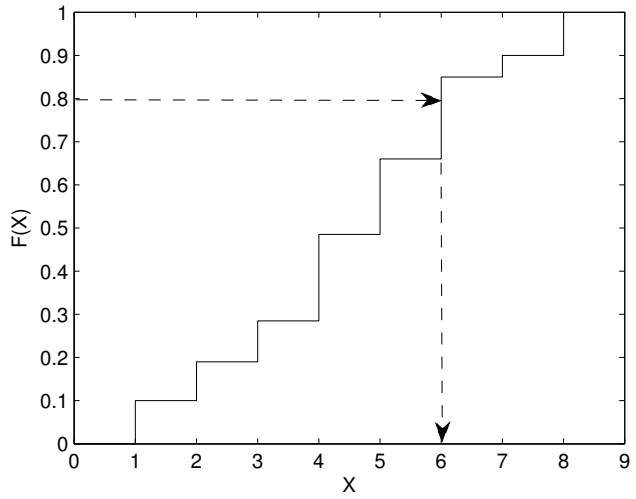

Figure 1.4: Illustration of the inverse transform procedure for generating discrete random variables. Note that we plot the cumulative probabilities for each outcome. If we sample a uniform random number of $U = 0.8$, then this yields a random value of $X = 6$

** 3 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 $F^{-1}(X)$ 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 $F(x|\lambda) = 1 - \exp(-x/\lambda)$. 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 Exponental(λ) 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 main advantage of the rejection sampling method is that it does not require any "burn-in" period. 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 $x^2 + y^2 > 1$. Note that in this procedure, 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 θ.

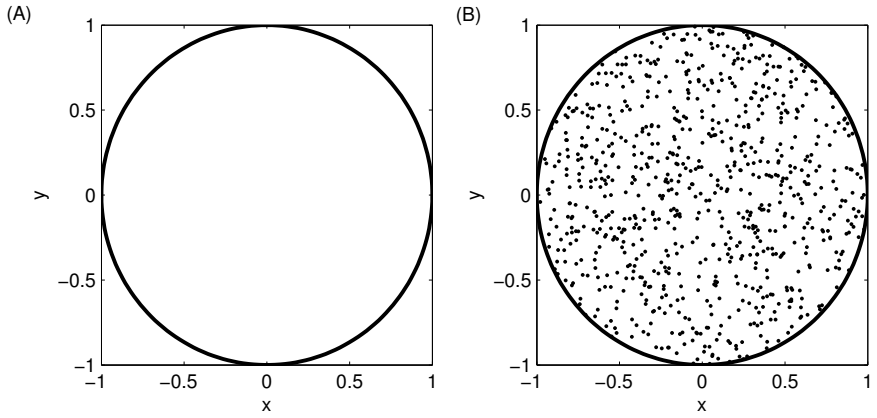

Figure 1.5: Sampling points uniformly from unit circle using rejection sampling

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.

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 θ. 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 θ. We will reject the proposal if $u > p(\theta)$ and accept it otherwise. If we accept the proposal, the sampled value θ 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 θ
3. Sample a proposal θ 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

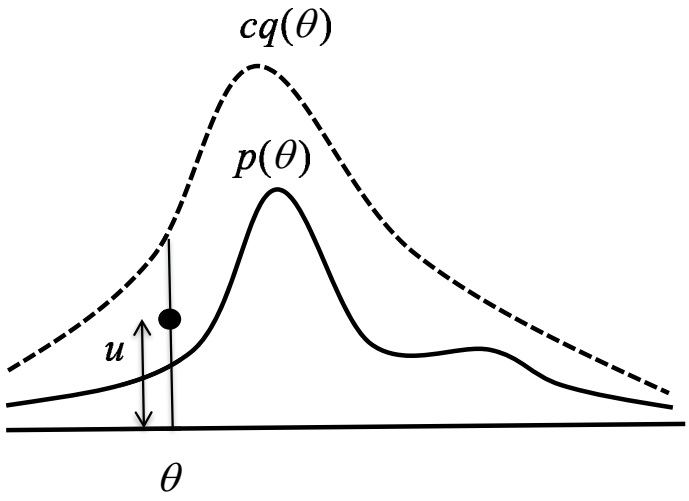

Figure 1.6: Illustration of rejection sampling. The particular sample shown in the figure will be rejected

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(α, β) distribution where $\alpha = 2$ and $\beta = 1$. This gives the probability density $p(x) = 2x$ for $0 < x < 1$. Assume that for whatever reason, we do not have access to the Beta random number generator that Matlab provides. Instead, the goal is to implement a rejection sampling algorithm in Matlab that samples from this distribution. For this exercise, use a simple uniform proposal distribution (even though this is not a good choice as a 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 become possible to solve using some form of MCMC, even with high-dimensional problems. The development of MCMC is arguably the biggest 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:

$$E[g(x)] = \int g(x)p(x)dx \tag{2.1}$$

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

$$E[g(x)] = \sum g(x)p(x)dx \tag{2.2}$$

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:

$$E[g(x)] = \frac{1}{n} \sum_{t=1}^{n} g(x^{(t)}) \tag{2.3}$$

In this procedure, 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(α, β) distribution with $\alpha = 3$ and $\beta = 4$ using Monte Carlo integration. You can use the Matlab function `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 `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:

$$x^{(1)} \rightarrow x^{(2)} \rightarrow \ldots \rightarrow x^{(t)} \rightarrow \ldots \tag{2.4}$$

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

   $t = t + 1$

   Sample a new value u from the transition function $p(x^{(t)}|x^{(t-1)})$

   Set $x^{(t)} = u$
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 the transition to a new state 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

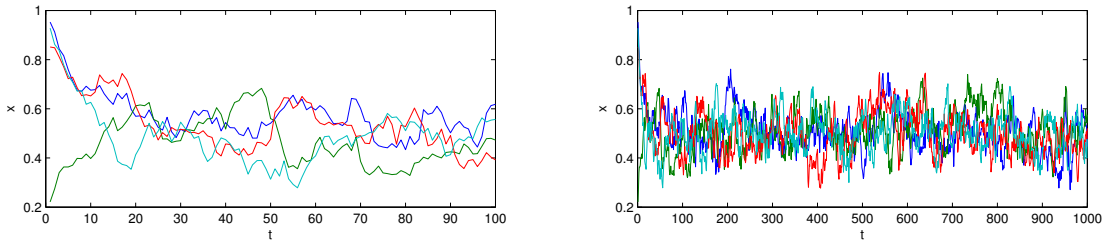

Figure 2.1: Illustration of a Markov chain starting with four different initial conditions. The right and left panes show the sequence of states at different temporal scales.

chain with four different initial values uniformly drawn from [0,1]. [tip: if X is a T x K matrix in Matlab such that $X(t,k)$ stores the state of the k^{th} Markov chain at the t^{th} iteration, the command `plot(X)` will simultaneously display the K sequences in different colors].

2.3 Putting it together: Markov chain Monte Carlo

The two previous sections discussed the main two ideas underlying MCMC, Monte Carlo sampling and Markov chains. Monte Carlo sampling allows one to estimate various characteristics of a distribution such as the mean, variance, kurtosis, or any other statistic of interest to a researcher. Markov chains involve a stochastic sequential process where we can sample states from some stationary distribution.

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. This is called the *target distribution*[1]. 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.

[1]The target distribution could be the posterior distribution over the parameters in the model or the posterior predictive distribution of a model (i.e., what future data could the model predict?), or any other distribution of interest to the researcher. Later chapters will hopefully make clearer that this abstract sounding concept of the *target distribution* can be something very important for a model.

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:

$$\theta^{(1)} \rightarrow \theta^{(2)} \rightarrow \ldots \rightarrow \theta^{(t)} \rightarrow \ldots \tag{2.5}$$

where $\theta^{(t)}$ represents the state of a Markov chain at iteration t. The samples from the chain, after burnin, 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 θ^* that is conditional on the previous state of the sampler[2]. The next step is to either accept the proposal or reject it. The probability of accepting the proposal is:

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

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 θ^* 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 the new proposal is more likely under the target distribution than the

[2]The proposal distribution is a distribution that is chosen by the researcher and good choices for the distribution depend on the problem. One important constraint for the proposal distribution is that it should *cover* the state space such that each potential outcome in state space has some non-zero probability under the proposal distribution

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 the 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 θ 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:

$$f(\theta) = \frac{1}{\pi(1+\theta^2)} \tag{2.7}$$

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

$$f(\theta) \propto \frac{1}{(1+\theta^2)} \tag{2.8}$$

Therefore, the Metropolis acceptance probability becomes

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

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 σ, 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.

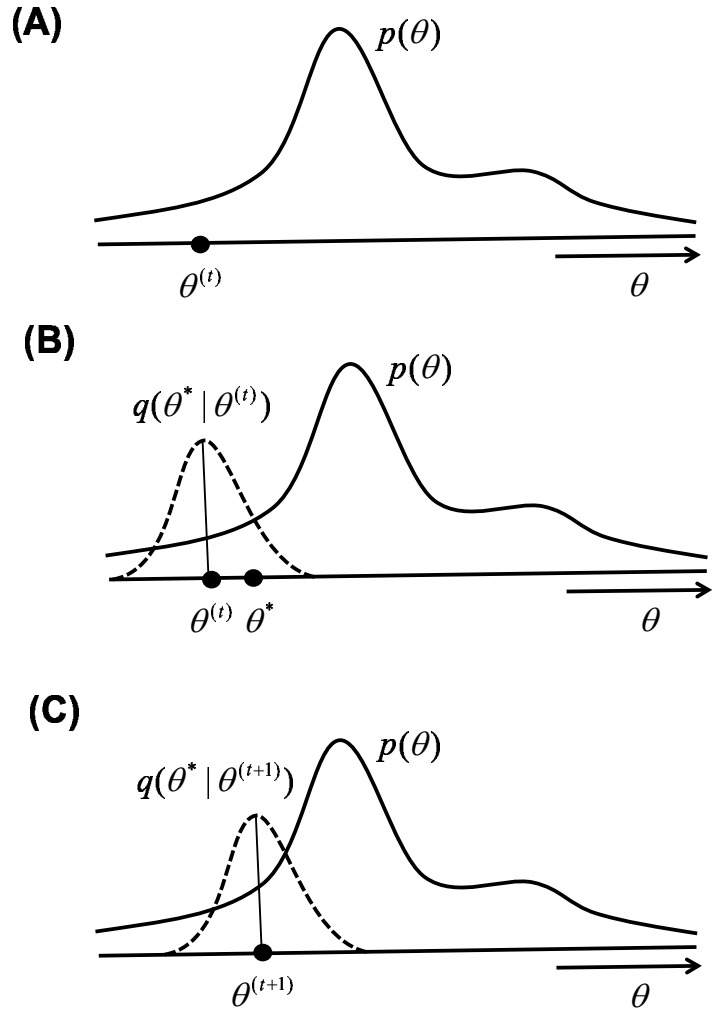

Figure 2.2: Illustration of the Metropolis sampler to sample from target density $p(\theta)$. (A) the current state of the chain is $\theta^{(t)}$. (B) a proposal distribution around the current state is used to generate a proposal θ^*. (C) the proposal was accepted and the new state is set equal to the proposal, and the proposal distribution now centers on the new state.

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 $\theta = -30$?

3. Calculate the proportion of samples that is accepted on average. Explore the effect of parameter σ on the average acceptance rate. Can you explain what is happening with the accuracy of the reconstructed distribution when σ is varied?

4. As a followup to the previous question, what is (roughly) the value of σ 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(μ, σ) 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.

```
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:

$$\alpha = \min\left(1, \frac{p(\theta^*)}{p(\theta^{(t-1)})} \frac{q(\theta^{(t-1)}|\theta^*)}{q(\theta^*|\theta^{(t-1)})}\right) \tag{2.10}$$

Listing 2.2: Matlab code to implement Metropolis sampler for Example 1

```
%% 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
t = 1;
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;
thetabins = linspace( thetamin , thetamax , nbins );
counts = hist( theta , thetabins );
bar( thetabins , counts/sum(counts) , 'k' );
xlim( [ thetamin thetamax ] );
xlabel( '\theta' ); ylabel( 'p(\theta)' );

%% Overlay the theoretical density
y  = cauchy( thetabins );
hold on;
plot( thetabins , 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' ); xlabel( '\theta' );
set( gca , 'YDir' , 'reverse' );
xlim( [ thetamin thetamax ] );
```

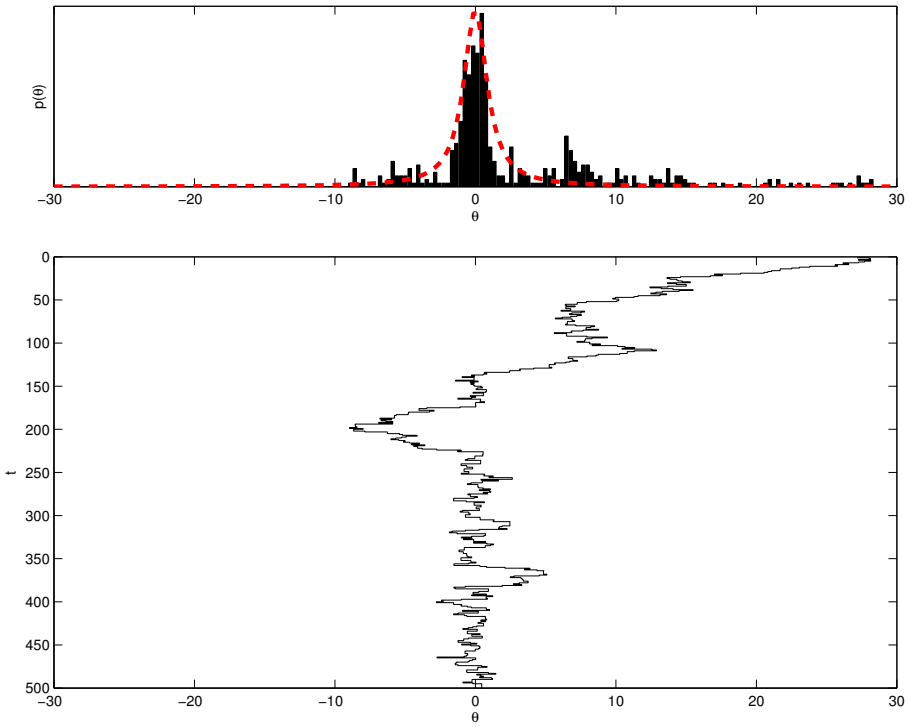

Figure 2.3: Simulation results where 500 samples were drawn from the Cauchy distribution using the Metropolis sampler. The upper panel shows the theoretical density in the dashed red line and the histogram shows the distribution of the samples. The lower panel shows the sequence of samples of one chain

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 $\theta^{(t)} = u$
3. Repeat

   $t = t + 1$

   Generate a proposal θ^* 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 `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 τ 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 τ 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 $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_N)$ to represent a random variable involving N components, and $\boldsymbol{\theta}^{(t)}$ represents the t^{th} state in our sampler. This leads to a generalization of the MH sampler where the scalar variables θ are now replaced by vectors $\boldsymbol{\theta}$:

1. Set $t = 1$
2. Generate an initial value $\boldsymbol{u} = (u_1, u_2, \ldots, u_N)$, and set $\boldsymbol{\theta}^{(t)} = \boldsymbol{u}$
3. Repeat

   $t = t + 1$

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

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

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

   If $u \leq \alpha$, accept the proposal and set $\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^*$, else set $\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)}$.
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 θ_1 and θ_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

Listing 2.3: Matlab code to implement bivariate density for Example 1

```
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 );
```

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 θ_1^* and θ_2^* are sampled from a Uniform(0,8) distribution. In other words, we sample proposals for $\boldsymbol{\theta}^*$ 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 $\frac{q(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\theta}^*)}{q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{(t-1)})} = 1$ 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.

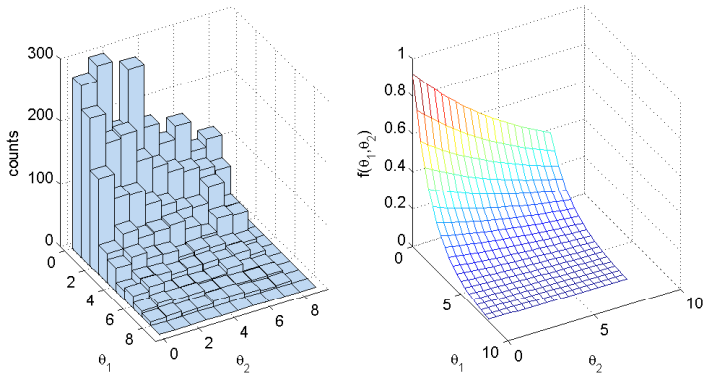

Figure 2.4: Results of a Metropolis sampler for the bivariate exponential (right) approximated with 5000 samples (left)

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

```
%% 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(1,1) = 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 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;
thetabins1 = linspace( thetamin(1) , thetamax(1) , nbins );
thetabins2 = linspace( thetamin(2) , thetamax(2) , nbins );
[ theta1grid , theta2grid ] = meshgrid( thetabins1 , thetabins2 );
ygrid  = bivexp( theta1grid , theta2grid );
mesh( theta1grid , theta2grid , ygrid );
xlabel( '\theta_1' ); ylabel('\theta_2' );
zlabel( 'f(\theta_1,\theta_2)' );
view(az, el);
```

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 of 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 give a simple example. Suppose that we are looking at the first five presidents: Washington, Adams, Jefferson, Madison, and Monroe. We will represent this true ordering by a vector $\boldsymbol{\omega} = (1, 2, 3, 4, 5) = (Washington, Adams, Jefferson, Madison, Monroe)$. 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 $\boldsymbol{\theta}$ is proportional to:

$$p(\boldsymbol{\theta}|\boldsymbol{\omega}, \lambda) \propto \exp\left(-d(\boldsymbol{\theta}, \boldsymbol{\omega})\lambda\right) \tag{2.12}$$

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

The problem is now to generate orderings $\boldsymbol{\theta}$ according to Mallows model, given the true ordering $\boldsymbol{\omega}$ and scaling parameter λ. This can be achieved in very simple ways using a Metropolis sampler. We start the sampler with $\boldsymbol{\theta}^{(1)}$ corresponding to a random permutation of items. At each iteration, we then make proposals $\boldsymbol{\theta}^*$ 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 $\boldsymbol{\theta}^*$ from the proposal distribution

$$q(\boldsymbol{\theta} = \boldsymbol{\theta}^*|\boldsymbol{\theta}^{(t-1)}) = \begin{cases} 1/\binom{N}{2} & \text{if } S(\boldsymbol{\theta}^*, \boldsymbol{\theta}^{(t-1)}) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

where $S(\boldsymbol{\theta}^*, \boldsymbol{\theta}^{(t-1)})$ 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

$$\alpha = \min\left(1, \frac{p(\boldsymbol{\theta}^*|\boldsymbol{\omega}, \lambda)}{p(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\omega}, \lambda)}\right) = \min\left(1, \frac{\exp\left(-d(\boldsymbol{\theta}^*, \boldsymbol{\omega})\lambda\right)}{\exp\left(-d(\boldsymbol{\theta}^{(t-1)}, \boldsymbol{\omega})\lambda\right)}\right). \tag{2.14}$$

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:

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

```
%% Chapter 2. Metropolis sampler for Mallows model
% 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
t = 1;
while t < T % Iterate until we have T samples
    t = t + 1;

    % Our proposal is the last ordering but with two items switched
```

```
22      lasttheta = theta(:,t-1); % Get the last theta
23      % Propose two items to switch
24      whswap = randperm( L ); whswap = whswap(1:2);
25      theta_star = lasttheta;
26      theta_star( whswap(1)) = lasttheta( whswap(2));
27      theta_star( whswap(2)) = lasttheta( whswap(1));
28
29      % calculate Kendall tau distances
30      dist1 = kendalltau( theta_star , omega );
31      dist2 = kendalltau( lasttheta  , omega );
32
33      % Calculate the acceptance ratio
34      pratio = exp( -dist1*lambda ) / exp( -dist2*lambda );
35      alpha = min( [ 1 pratio ] );
36      u = rand; % Draw a uniform deviate from [ 0 1 ]
37      if u < alpha  % Do we accept this proposal?
38          theta(:,t) = theta_star; % proposal becomes new theta
39      else
40          theta(:,t) = lasttheta; % copy old theta
41      end
42      % Occasionally print the current state
43      if mod( t,10 ) == 0
44          fprintf( 't=%3d\t' , t );
45          for j=1:L
46              fprintf( '%15s' , labels{ theta(j,t)} );
47          end
48          fprintf( '\n' );
49      end
50  end
```

```
t=400       Jefferson        Madison          Adams         Monroe     Washington
t=410      Washington         Monroe         Madison       Jefferson          Adams
t=420      Washington      Jefferson         Madison          Adams         Monroe
t=430       Jefferson         Monroe      Washington          Adams        Madison
t=440      Washington        Madison         Monroe          Adams      Jefferson
t=450       Jefferson     Washington          Adams        Madison         Monroe
t=460      Washington      Jefferson          Adams        Madison         Monroe
t=470          Monroe     Washington       Jefferson          Adams        Madison
t=480           Adams     Washington         Monroe       Jefferson        Madison
t=490           Adams        Madison       Jefferson         Monroe     Washington
t=500          Monroe          Adams         Madison       Jefferson     Washington
```

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 $\boldsymbol{\theta}$ involving all its components simultaneously, it might be computationally simpler to make proposals for individual components of $\boldsymbol{\theta}$, one at a time. This leads to a *componentwise* updating approach.

For example, suppose we have a bivariate distribution $\boldsymbol{\theta} = (\theta_1, \theta_2)$. 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 θ_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 θ_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 $\boldsymbol{u} = (u_1, u_2, \ldots, u_N)$, and set $\boldsymbol{\theta}^{(t)} = \boldsymbol{u}$
3. Repeat

   $t = t + 1$

   Generate a proposal θ_1^* from $q(\theta_1|\theta_1^{(t-1)})$

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

   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 θ_2^* from $q(\theta_2|\theta_2^{(t-1)})$

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

   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 $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. The parameter $\boldsymbol{\mu}$ is a 1 x N vector containing the mean in the N-dimensional space. The parameter $\boldsymbol{\Sigma}$ is a N x N covariance matrix. For our example, we will use a bivariate normal with $\boldsymbol{\mu} = (0, 0)$ and $\boldsymbol{\Sigma} = \left(\begin{smallmatrix} 1 & 0.3 \\ 0.3 & 1 \end{smallmatrix}\right)$.

The density of this distribution is illustrated in the right panel of Figure 2.5. The goal is to sample values $\boldsymbol{\theta}$ from this distribution. We use a componentwise Metropolis sampler with a normal distribution for the components θ_1 and θ_2. The proposal distribution is centered around the old value of each component and has standard deviation σ. The code in Listing 2.7 shows the Matlab implementation for this sampler. The output of this code is shown in Figure 2.5.

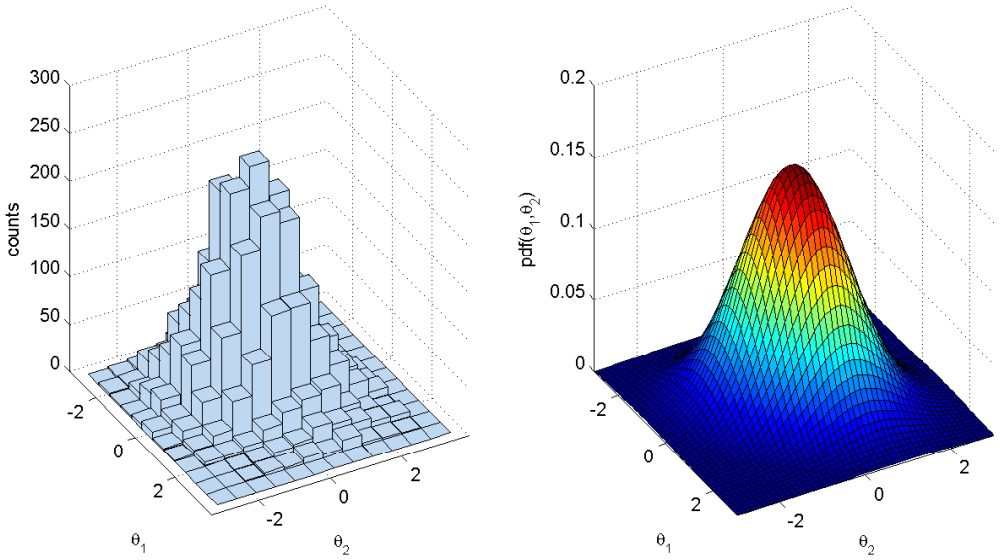

Figure 2.5: Results of the componentwise Metropolis sampler for the bivariate normal distribution. Left panel shows the approximation using 5000 samples. Right panel shows the theoretical density.

Listing 2.7: Implementation of componentwise Metropolis sampler for bivariate normal

```
%% Chapter 2. Metropolis procedure to sample from Bivariate Normal
% Component-wise updating. Use a normal proposal distribution

%% 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
```

```
15 state = zeros( 2 , T ); % Init storage space for the state of the sampler
16 theta1 = unifrnd( thetamin(1) , thetamax(1) ); % Start value for theta1
17 theta2 = unifrnd( thetamin(2) , thetamax(2) ); % Start value for theta2
18 t = 1; % initialize iteration at 1
19 state(1,t) = theta1; % save the current state
20 state(2,t) = theta2;
21
22 %% Start sampling
23 while t < T % Iterate until we have T samples
24     t = t + 1;
25
26     %% Propose a new value for theta1
27     new_theta1 = normrnd( theta1 , propsigma );
28     pratio = mvnpdf( [ new_theta1 theta2 ]  , mu , sigma  ) / ...
29              mvnpdf( [     theta1 theta2 ]  , mu , sigma  );
30     alpha = min( [ 1 pratio ] ); % Calculate the acceptance ratio
31     u = rand; % Draw a uniform deviate from [ 0 1 ]
32     if u < alpha  % Do we accept this proposal?
33         theta1 = new_theta1; % proposal becomes new value for theta1
34     end
35
36     %% Propose a new value for theta2
37     new_theta2 = normrnd( theta2 , propsigma );
38     pratio = mvnpdf( [ theta1 new_theta2  ]  , mu , sigma  ) / ...
39              mvnpdf( [ theta1     theta2  ]  , mu , sigma  );
40     alpha = min( [ 1 pratio ] ); % Calculate the acceptance ratio
41     u = rand; % Draw a uniform deviate from [ 0 1 ]
42     if u < alpha  % Do we accept this proposal?
43         theta2 = new_theta2; % proposal becomes new value for theta2
44     end
45
46     %% Save state
47     state(1,t) = theta1;
48     state(2,t) = theta2;
49 end
50
51 %% Display histogram of our samples
52 figure( 1 ); clf;
53 subplot( 1,2,1 );
54 nbins = 12;
55 thetabins1 = linspace( thetamin(1) , thetamax(1) , nbins );
56 thetabins2 = linspace( thetamin(2) , thetamax(2) , nbins );
57 hist3( state' , 'Edges' , {thetabins1 thetabins2} );
58 xlabel( '\theta_1' ); ylabel('\theta_2' ); zlabel( 'counts' );
59 az = 61; el = 30; view(az, el);
60
61 %% Plot the theoretical density
62 subplot(1,2,2);
63 nbins = 50;
64 thetabins1 = linspace( thetamin(1) , thetamax(1) , nbins );
65 thetabins2 = linspace( thetamin(2) , thetamax(2) , nbins );
```

```
66 [ theta1grid , theta2grid ] = meshgrid( thetabins1 , thetabins2 );
67 zgrid  = mvnpdf( [ theta1grid(:) theta2grid(:)] , mu , sigma );
68 zgrid = reshape( zgrid , nbins , nbins );
69 surf( theta1grid , theta2grid , zgrid );
70 xlabel( '\theta_1' ); ylabel('\theta_2' );
71 zlabel( 'pdf(\theta_1,\theta_2)' );
72 view(az, el);
73 xlim([thetamin(1) thetamax(1)]); ylim([thetamin(2) thetamax(2)]);
```

Exercises

1. Modify the code for Example 1 to implement a component-wise sampler. Use a uniform proposal distribution for both θ_1 and θ_2. Visualize the distribution in the same way as Figure 2.4. Calculate the acceptance probability separately for θ_1 and θ_2. Why is the acceptance probability for θ_2 higher than for θ_1?

2. Investigate the effect of the scaling parameter λ 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 distribition under Mallows 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 $p(\boldsymbol{\theta}|\boldsymbol{\omega}, \lambda) = \frac{1}{\psi(\lambda)}\exp\left(-d(\boldsymbol{\theta}, \boldsymbol{\omega})\lambda\right)$ where $\psi(\lambda)$ is a normalizing function where $\psi(\lambda) = \prod_{i=1}^{N} \frac{1-\exp(-i\lambda)}{1-\exp(-\lambda)}$. Therefore, it is possible with small N to calculate the exact probability of any ordering $\boldsymbol{\theta}$. 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.

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 improved computational efficiency. An additional advantage is that the researcher does not have to specify a proposal distribution, leaving some guessing work out of the MCMC procedure.

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 $\boldsymbol{u} = (u_1, u_2)$, and set $\boldsymbol{\theta}^{(t)} = \boldsymbol{u}$
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 $\boldsymbol{\theta} = (\theta_1, \theta_2)$ having a bivarite normal distribution

$$\boldsymbol{\theta} \sim \text{Norm}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.15}$$

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

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \tag{2.16}$$

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

$$\theta_1 \sim \text{Norm}(\mu_1 + \rho(\theta_2 - \mu_2), \sqrt{1 - \rho^2}) \tag{2.17}$$

$$\theta_2 \sim \text{Norm}(\mu_2 + \rho(\theta_1 - \mu_1), \sqrt{1-\rho^2}) \tag{2.18}$$

This leads to the following Gibbs sampling procedure:

1. Set $t = 1$
2. Generate an initial value $\boldsymbol{u} = (u_1, u_2)$, and set $\boldsymbol{\theta}^{(t)} = \boldsymbol{u}$
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 $\boldsymbol{\mu} = (0, 0)$ 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).

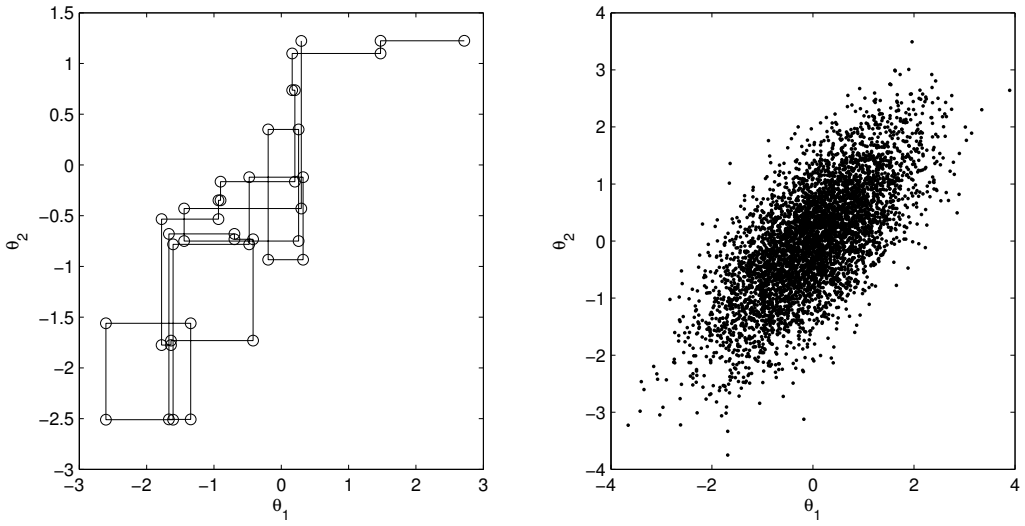

Figure 2.6: Results of a Gibbs sampler for the bivariate normal distribution. Left panel shows the progression of the sampler state for the first 20 iterations. Right panel shows a scatterplot of 5000 sampled values.

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 $\boldsymbol{\mu} = (0, 0)$ 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`).

Chapter 3

Basic concepts in Bayesian Data Analysis

Up to this point, we have illustrated some Markov chain Monte Carlo techniques but it might not be clear how this is useful for the purpose of modeling or data analysis. In this chapter, we will show that MCMC can be used in the context of Bayesian data analysis. The goal will be to find parameter values in a probabilistic model that best explain the data. We will start with some very simple probabilistic models that involve a few parameters. Through examples, we can see how to "fit" models to data from a Bayesian perspective. In the next chapter, we will generalize these results to a class of probabilistic models that can be defined by directed edges among variables (so called directed graphical models).

To start, let's assume that the data we want to analyze can be represented by D, and that the parameters of a model are represented by θ. Let's also assume that we have a model that is designed to explain the data in a generative sense. This means that we can define the likelihood $p(D|\theta)$ of the model. This is essentially how likely the data is according to some instantiation of the model in terms of its model parameters θ. Note the direction of this conditional probability. The likelihood gives an explanation of the data in terms of the parameters. In this coursebook, we will assume that it will be possible to evaluate the probability mass or density of this likelihood. Therefore, given some instantiation of the parameters, we should be able, given the definition of our model, to say what the likelihood is [1].

The primary goal in any modeling analysis is to choose parameters of a model such that we can best explain the data with the model. There are a number of approaches to choose model parameters.

[1] Note that in some models, the likelihood cannot be evaluated. These might involve probabilistic models in which there only is a process that can be used to simulate how samples of the data can be generated. In these cases, we need to resort to different techniques such as approximate bayesian inference, a topic that is outside the scope of this course book

3.1 Parameter Estimation Approaches

3.1.1 Maximum Likelihood

A very basic approach to parameter estimation is *maximum likelihood (ML)*. The goal here is to find the parameter estimates that maximize the likelihood of the data given the model that operates with those parameter estimates. This corresponds to:

$$\theta_{ML} = \arg\max_{\theta_i} p(D|\theta_i)$$

where the values θ_i vary across the range of allowable values. The ML estimate corresponds to the values of the parameters that best agree with the actually observed data. Although this seems like a very straightforward way of estimating parameters in a probabilistic model, there are some serious drawbacks to this approach. For example, when we deal with small samples sizes, the ML estimate might not even be defined. It might also be nontrivial to find the maximum of the likelihood function when there are many parameters in the model. In this case, numerical techniques that explore the gradient of the likelihood function might get stuck in suboptimal solutions (e.g., local modes) or show a sensitivity to the choice of starting values. For our purposes, it is most important to realize that this estimation procedure does not correspond to a Bayesian approach.

3.1.2 Maximum a posteriori

A Bayesian approach to parameter estimation is to treat θ as a random variable. We associate a *prior distribution* with this variable: $p(\theta)$. The prior distribution tells us how likely the parameter values are when we have not observed any data yet. In a Bayesian context, the goal is to estimate the *posterior distribution* over parameters given our data. This is also known as *posterior inference*. In other words, we would like to know $p(\theta|D)$. Using Bayes rule, we can evaluate this conditional probability with:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{3.1}$$

For our purposes, we rarely need to explicitly evaluate the denominator $p(D)$. Therefore, we can ignore any constant of proportionality and just write:

$$\underbrace{p(\theta|D)}_{posterior} \propto \underbrace{p(D|\theta)}_{likelihood}\underbrace{p(\theta)}_{prior} \tag{3.2}$$

Note that the posterior distribution is specified by a simple product of the likelihood (how likely is the data given the model that uses these parameter estimates) and the prior (how likely are these parameter values to begin with?). In Bayesian data analysis, one way to apply a model to data is to find the *maximum a posteriori (MAP)* parameter values. The goal here is to find the parameter estimates that maximize the posterior probability of the

parameters given the data. In other words, we find the mode of the posterior distribution. This corresponds to:

$$\begin{aligned} \theta_{MAP} &= \arg\max_{\theta_i} p(\theta_i|D) \\ &= \arg\max_{\theta_i} \frac{p(D|\theta_i)p(\theta_i)}{p(D)} \\ &= \arg\max_{\theta_i} p(D|\theta_i)p(\theta_i) \end{aligned} \tag{3.3}$$

This looks similar to the maximum likelihood estimation procedure. The difference is that the prior over parameters will influence parameter estimation. This procedure will find parameters that make the data likely (a large likelihood) but that also are likely to begin with (large prior probability). Although it seems that the prior should not play a role in parameter estimation, there are some good reasons to prefer the MAP over the ML estimation, especially when only few data points are available or when the data is degenerate in some way (e.g., the data shows no variation). There is a large literature on this, and we will not review it here.

3.1.3 Posterior Sampling

Finally, the Bayesian approach that we are really interested in is posterior sampling. Note that with the MAP approach, we get a *single* set of parameter values for a model. Therefore, we are characterizing the posterior distribution with the mode of this distribution. This seems conceptually very clear and useful but it has some drawbacks. For example, suppose there are multiple sets of parameter values that all have very high posterior probability. Shouldn't we try to find these other sets of parameter values as well? Also, suppose there are tradeoffs between parameter values. For example, suppose we have a model with two parameters A and B and we achieve high posterior probability by either setting A to a high value and B to a low value or the other way around, setting A to a low and B to a high value. In a MAP estimate, we are not getting an understanding of such parameter correlations, but we can measure such correlations using posterior sampling.

In a more comprehensive Bayesian approach, the goal is to characterize the full posterior distribution and not to simply find the mode of this distribution. In some cases, we might be able to find an analytic expression for the posterior distribution. However, in many cases, we have to resort to sampling techniques, such as MCMC, to get samples from the posterior distribution. These samples can be used to calculate a number of things, such as means, variances and other moments of the distribution. We can also check whether there are any correlations between parameters.

Here is one key insight: if you understand that MCMC can be used to draw samples from a distribution, you should realize that MCMC can be used to get samples from the posterior distribution. Therefore, all the methods that we have discussed in the previous chapter can be used for the problem of posterior inference.

3.2 Example: Estimating a Weibull distribution

Suppose we have some univariate data that consist of response time measurements in an experiment. To model this data, let's assume that the Weibull distribution will be a useful descriptive model. The pdf for the Weibull is:

$$p(y|a,b) = ba^{-b}y^{b-1}e^{-(\frac{y}{a})^b}I_{(0,\text{inf})}(y) \tag{3.4}$$

This distribution has two parameters: a controls the scale of the distribution and b controls the shape of the distribution. Let's say that our data is represented by y_i, where $i \in 1..N$, and N is the number of observations. In our model, let's assume that each observation is an independent draw from a Weibull distribution. Given these assumptions, we now have a likelihood function for our data. Given that each observation is assumed to be an independent draw from a Weibull with parameters a and b, the likelihood of all observations is given by:

$$\begin{aligned} p(y_1,...,y_N|a,b) &= p(y_1|a,b)p(y_2|a,b)...p(y_N|a,b) \\ &= \prod_i p(y_i|a,b) \\ &= \prod_i ba^{-b}y_i^{b-1}e^{-(\frac{y_i}{a})^b}I_{(0,\text{inf})}(y_i) \end{aligned} \tag{3.5}$$

We are not completely done with our model description however. In any Bayesian analysis, we also need to specify prior distributions over parameters. Let's assume that parameters a and b are independently distributed according to an exponential distribution with parameters λ_a and λ_b respectively. The pdfs for these prior distributions are:

$$\begin{aligned} p(a|\lambda_a) &= \frac{1}{\lambda_a}e^{-\frac{a}{\lambda_a}} \\ p(a|\lambda_b) &= \frac{1}{\lambda_b}e^{-\frac{b}{\lambda_b}} \end{aligned} \tag{3.6}$$

We should point out that the model expressed above can be described in much simpler terms using sampling notation:

$$\begin{aligned} y_i|a,b &\sim \text{Weibull}(a,b) \\ a &\sim \text{Exp}(\lambda_a) \\ b &\sim \text{Exp}(\lambda_b) \end{aligned} \tag{3.7}$$

where we simply have used the terms "Weibull" amd "Exp" as notations for the full functional form in Equation 3.4 and Equation 3.6.

We now have a fully specified model where we have a likelihood as well as a prior on all parameters. The goal is to do posterior inference and find the posterior distribution over parameters conditional on the data. We previously stated that:

$$\underbrace{p(\theta|D)}_{posterior} \propto \underbrace{p(D|\theta)}_{likelihood}\underbrace{p(\theta)}_{prior} \tag{3.8}$$

In our case, θ=(a,b) and D=(y_1,...,y_N). Therefore, we can write:

$$\underbrace{p(a,b|y_1,...,y_N)}_{posterior} \propto \underbrace{p(y_1,...,y_N|a,b)}_{likelihood}\underbrace{p(a,b)}_{prior} \tag{3.9}$$

Note that we have decided to use independent priors over parameters a and b. This allows us to write:

$$\underbrace{p(a,b|y_1,...,y_N)}_{posterior} \propto \underbrace{p(y_1,...,y_N|a,b)}_{likelihood}\underbrace{p(a)p(b)}_{prior} \tag{3.10}$$

To see what the functional form of this posterior distribution looks like, we can insert equations 3.5 into the likelihood term and 3.6 into the prior terms. This leads to

$$\underbrace{p(a,b|y_1,...,y_N)}_{posterior} \propto \underbrace{\left(\prod_i ba^{-b}y_i^{b-1}e^{-\left(\frac{y_i}{a}\right)^b}I_{(0,\text{inf})}(y_i)\right)}_{likelihood}\underbrace{\frac{1}{\lambda_a}e^{-\frac{a}{\lambda_a}}\frac{1}{\lambda_b}e^{-\frac{b}{\lambda_b}}}_{prior} \tag{3.11}$$

This expression is not amenable to analytic techniques. However, we can use sampling techniques such as MCMC to draw samples from this posterior distribution. It is important to realize that although the expression in 3.11 looks daunting, we can evaluate the unnormalized density of this function for any set of parameter values a and b and observations y_1,...,y_N (remember that we only need unnormalized densities in MCMC, which is an appealing feature of this approach).

Our posterior distribution is bivariate as it involves our two parameters a and b. Suitable MCMC approaches are the Gibbs sampler and the bivariate MH sampler from the previous chapter. Because the conditional distributions in 3.11 do not correspond to any analytic expression, we have to use a Metropolis Hastings sampler. We will choose a component-wise updating scheme. Here is a high-level description of the MH procedure:

1. Set $t = 1$
2. Generate an initial value for a and b
3. Repeat

   $t = t + 1$

   Do a MH step on a

   Generate a proposal a^* from a suitable proposal distribution

   Evaluate the acceptance probability α using a^*, a, and b

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

   If $u \leq \alpha$, accept the proposal and set $a = a^*$

   Do a MH step on b

Generate a proposal b^* from a suitable proposal distribution

Evaluate the acceptance probability α using b^*, b, and a

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

If $u \leq \alpha$, accept the proposal and set $b = b^*$

4. Until $t = T$

Exercises

1. What might be suitable proposal distributions for a and b in the MH sampler outlined above?

2. Give the matlab code for evaluating the acceptance probability for the two MH steps in the MH sampler? There is no need to write out the full functional form. Just use matlab functions such as `wblpdf` and `exppdf` in your expression.

3. The likelihood ratio in the expression for the acceptance probability of the MH sampler can often contain products over a large set of terms. For example, we might have the expression $\frac{\prod_i p(y_i|a,b)}{\prod_i p(y_i|a^*,b)}$ where we first evaluate the product in numerator, then the product in denominator and then take the ratio of these products. This is mathematically equivalent to $\prod_i \frac{p(y_i|a,b)}{p(y_i|a^*,b)}$ where we take each individual ratio and then multiply. For computational reasons, why is it better to use the latter approach?

4. Implement the full Metropolis-Hastings sampler from this example in Matlab. If you get stuck, you can use the supplied code `ch3_posteriorinference_weibull.m`. You can use $\lambda_a = \lambda_b = 1$. For the data y, you can use the dataset provided in the matlab file `rtdata1` supplied on the website. In this dataset, there are 100 observations total.

5. With your MH sampler, estimate the mean of a and b in the posterior distribution

6. With the statistics toolbox, you can also get parameter estimates from a maximum likelihood procedure. Use the `wblfit` to find the ML estimates for the dataset provided by in the matlab file `rtdata1`. Are the answers you get from ML different from the Bayesian procedure?

7. Assume that you have very sparse data. Instead of 100 observations, you now have only one observation. Let's say $y_1 = 2$ and N=1. What are the mean posterior estimates you get for this data using your MH procedure? What does matlab's maximum likelihood procedure give you?

** 8 Explore the effect of the parameters of the prior, λ_1 and λ_2.

3.3 Example: Logistic Regression

tba

3.4 Example: Mallows Model

tba

Chapter 4

Directed Graphical Models

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 modes, 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.

4.1 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.

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,

$$p(x) = \sum_y p(x, y). \tag{4.1}$$

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

$$\begin{aligned} p(x, y) &= p(y|x)p(x) \\ &= p(x|y)p(y). \end{aligned} \tag{4.2}$$

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:

$$\begin{aligned} p(x, y, z) &= p(x|y, z)p(y, z) \\ &= p(x|y, z)p(y|z)p(z). \end{aligned} \tag{4.3}$$

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

$$\begin{aligned} p(y|x) &= \frac{p(x|y)p(y)}{p(x)} \\ &= \frac{p(x|y)p(y)}{\sum_{y'} p(x|y')p(y')} \end{aligned} \tag{4.4}$$

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:

$$p(y|x) \propto p(x|y)p(y) \tag{4.5}$$

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:

$$p(x, y) = p(x)p(y). \tag{4.6}$$

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:

$$p(x|y) = p(x). \tag{4.7}$$

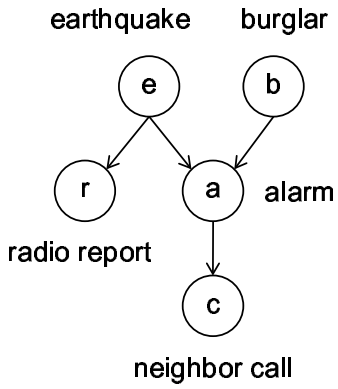

Figure 4.1: A graphical model that illustrates the conditional dependencies between variables.

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 *conditionally independent* of z if the following holds:

$$p(x, y|z) = p(x|z)p(y|z). \tag{4.8}$$

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

$$p(x|z, y) = p(x|z). \tag{4.9}$$

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 graphical models.

Exercises

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

4.2 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 4.1. There are five 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: $p(c|a, b) = p(c|a)$.

4.2.1 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 4.2. 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

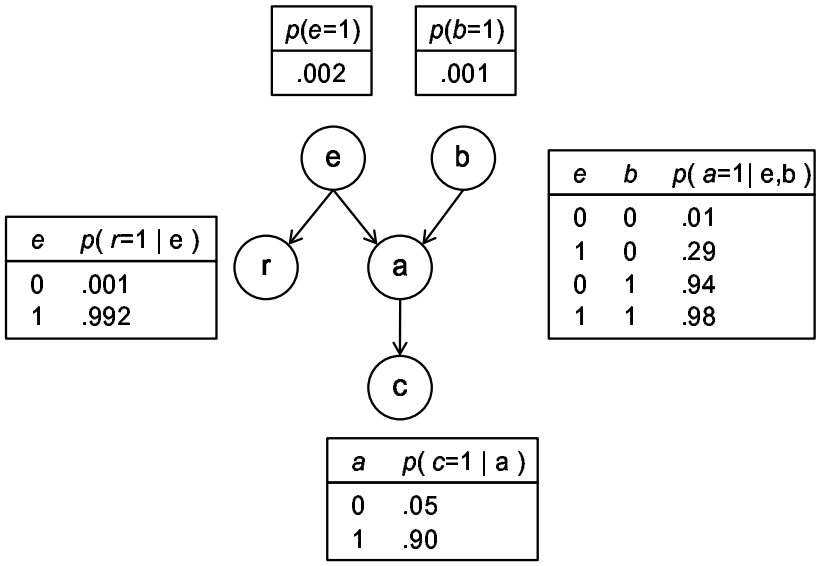

Figure 4.2: The conditional probability tables (CPTs) for the graphical model example.

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 of alarm and therefore, we need to sum over all possibilities regarding earthquakes. Using the sum and product rules, we can calculate this by

$$\begin{aligned} p(a = 1|b = 1) &= p(a = 1|b = 1, e = 0)p(e = 0) + p(a = 1|b = 1, e = 1)p(e = 1) \\ &= (0.94)(0.998) + (0.98)(0.002) \\ &= 0.9401 \end{aligned}$$

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:

$$\begin{aligned} p(e=1|r=1) &= \frac{p(r=1|e=1)p(e=1)}{p(r=1|e=1)p(e=1)+p(r=1|e=0)p(e=0)} \\ &= \frac{(.992)(0.002)}{(.992)(0.002)+(.001)(.998)} \\ &= .6653 \end{aligned}$$

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: $p(b=1|c=1)$.

4.2.2 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 $p(b|e,a) \neq p(b|a)$.

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.

4.2.3 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:

$$p(x_1, \ldots, x_K) = \prod_{k=1}^{K} p(x_k|\text{pa}_k) \tag{4.10}$$

where pa_k represents the realizations of variables that are parents of the kth 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 $p(x_k|\text{pa}_k)$ 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 called *directed acyclic graphs* or *DAGs*. These are subject to an important restriction: there cannot be any directed cycles in the graph, whereby following the arrows from one node leads us to back to the same node. In other words, there cannot be any (directed) loops in the graph.

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

$$\begin{aligned} p(a = 1, b = 0, &c = 1, e = 1, r = 1) \\ &= p(c = 1|a = 1)p(a = 1|e = 1, b = 0)p(r = 1|e = 1)p(e = 1)p(b = 0) \\ &= (0.90)(0.29)(0.992)(0.002)(0.999) \\ &= 0.0005173. \end{aligned}$$

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 4.3, panel C. Suppose we want to know whether x and y are independent

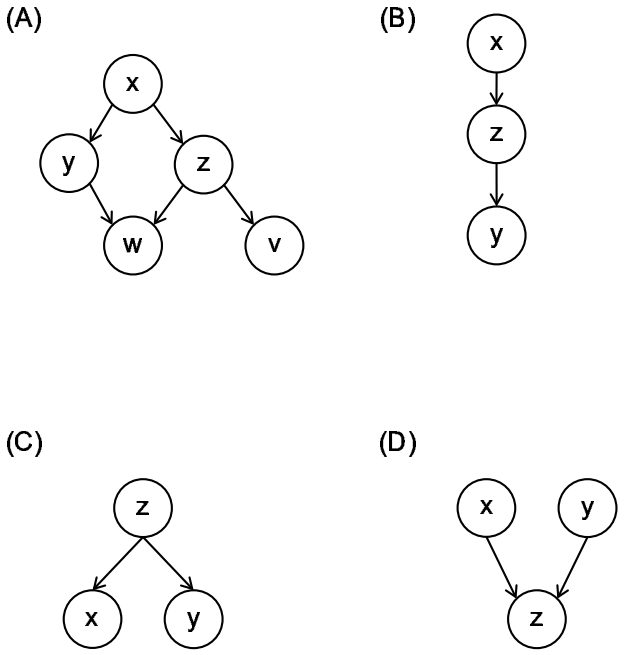

Figure 4.3: Four example graphical models

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:

$$\begin{aligned} p(x, y|z) &= \frac{p(x, y, z)}{p(z)} \\ &= \frac{p(x|z)p(y|z)p(z)}{p(z)} \\ &= p(x|z)p(y|z) \end{aligned}$$

Therefore, the conditional independence holds in this case.

Exercises

1. Consider the graphical model in Figure 4.3, 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 4.3, determine if x and y are conditionally independent of z. Also, in what cases are x and y independent?

4.3 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 Normal(μ, σ) distribution with the same mean μ and standard deviation σ. The corresponding graphical model is shown in Figure 4.4, 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 4.4, 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.

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 τ of the normal distribution, where $\sigma = \sqrt{1/\tau}$. Figure 4.4, panel C shows how we can include this in our model. We use a double-bordered node for σ to indicate that the variable is deterministic. Therefore, if we know τ, the value of σ 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 τ directly to y_i and have omitted the node for σ 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 μ and σ.

4.3.1 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

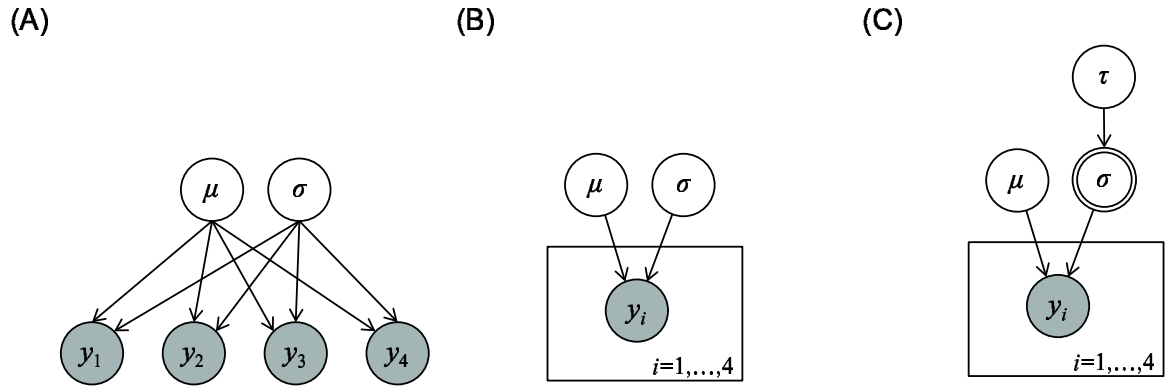

Figure 4.4: Examples of graphical models where shaded and unshaded nodes indicate observed and unobserved variables respectively. The models in panel A are B are identical. The plate notation in panel B provides a more compact representation of models with repeated sampling steps. Panel C illustrates a deterministic variable as indicated by the double-bordered node.

Irvine, California". Although we as experimenters know the true answer to each question, the goal of the modeling is to estimate the true answer μ 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 μ, which is a latent variable in the model. We also assume that each individual is associated with a separate standard deviation σ_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 τ_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:

$$\begin{aligned} y_{ij} &\sim \text{Norm}(\mu, \sigma_j) \\ \sigma_j &= 1/\sqrt{\tau_j} \\ \tau_j &\sim \text{Gamma}(a, b) \end{aligned} \tag{4.11}$$

where a and b are *hyperparameters* in the model. The associated graphical model is shown in Figure 4.5. Note that there 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.

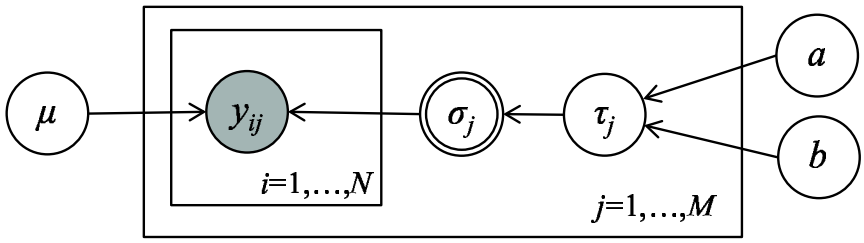

Figure 4.5: A consensus model for normally distributed data.

Exercises

1. Suppose in the model in Figure 4.4, 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 4.4, 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 4.11. 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 μ_k. To simplify the model, we assume that each individual has an uncertainty σ_j regardless of what question is asked. Visualize the corresponding graphical model.

6. Suppose we have the following model:

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

   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 5

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.

5.1 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 5.1A. 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 $\boldsymbol{y} = (y_1, \ldots, y_N)$ represent the set of observed variables in a graphical

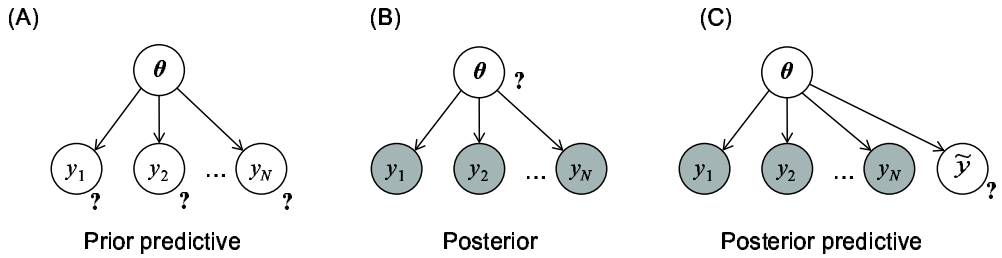

Figure 5.1: Three inference problems in graphical models. The question marks indicate the variables that need to be inferred for each problem.

model. Let $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M)$ be the set of latent variables in a model. The prior predictive distribution corresponds to:

$$\begin{aligned} p(\boldsymbol{y}) &= \int p(\boldsymbol{y}, \boldsymbol{\theta}) d\boldsymbol{\theta} \\ &= \int p(\boldsymbol{\theta}) p(\boldsymbol{y}|\boldsymbol{\theta}) d\boldsymbol{\theta} \end{aligned} \tag{5.1}$$

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 5.1 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 work 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.

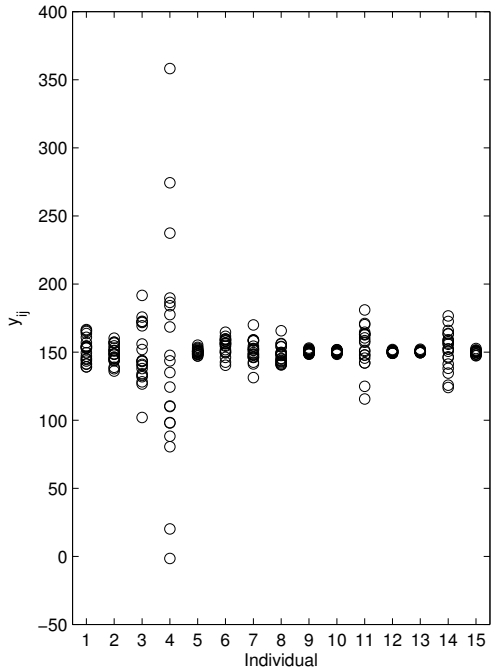

Figure 5.2: Data pattern sampled from the prior predictive distribution for the consensus model with normal distributions. The model used $\mu = 150$, $a = 0.3$, $b = 3.33$, $M = 15$ and $N = 20$.

Exercises

1. Consider the alarm network discussed earlier and shown in 4.2. 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 4.11. 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 5.2. 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 σ_j but all estimates are centered around 150. Save the resulting data pattern to a file. You'll need it for a later exercise.

5.2 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 5.1B for a canonical example of this problem). In Bayesian modeling, this corresponds to the problem of finding the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y}) = p(\theta_1, \theta_2, \ldots, \theta_M|y_1, \ldots, y_N)$. 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.

5.2.1 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 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 $c = 0$.

5.2.2 MCMC Sampling

We can consider the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y}) = p(\theta_1, \theta_2, \ldots, \theta_M|y_1, \ldots, y_N)$ 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 θ_1 and θ_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 $p(\theta_1|\boldsymbol{y}, \theta_2 = v_2)$. 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 $f(\theta_2|\boldsymbol{y}, \theta_1 = v_1)$. 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 $v_1 \sim p(\theta_1|\boldsymbol{y}, \theta_2 = v_2)$

   Sample $v_2 \sim p(\theta_2|\boldsymbol{y}, \theta_1 = v_1)$

   Update the state $\boldsymbol{\theta}_1^{(t)} = v_1$ and $\boldsymbol{\theta}_2^{(t)} = v_2$

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 $\boldsymbol{\theta}^{(t)}$ 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, ..., v_M, for each latent variable
3. Set the initial state $\boldsymbol{\theta}^{(t)} = \boldsymbol{v}$
4. Repeat
   $t = t + 1$
   $j = 0$
   Repeat
   $j = j + 1$
   Sample $v_j \sim p(\theta_j | \boldsymbol{y}, \boldsymbol{\theta}_{-j} = \boldsymbol{v}_{-j})$
   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 $p(\theta_j | \boldsymbol{y}, \boldsymbol{\theta}_{-j} = \boldsymbol{v}_{-j})$ 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 5.3, panel A. Suppose in our sampler, we are at the step where need to sample an assignment for the third latent variable θ_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 4.10, giving us:

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

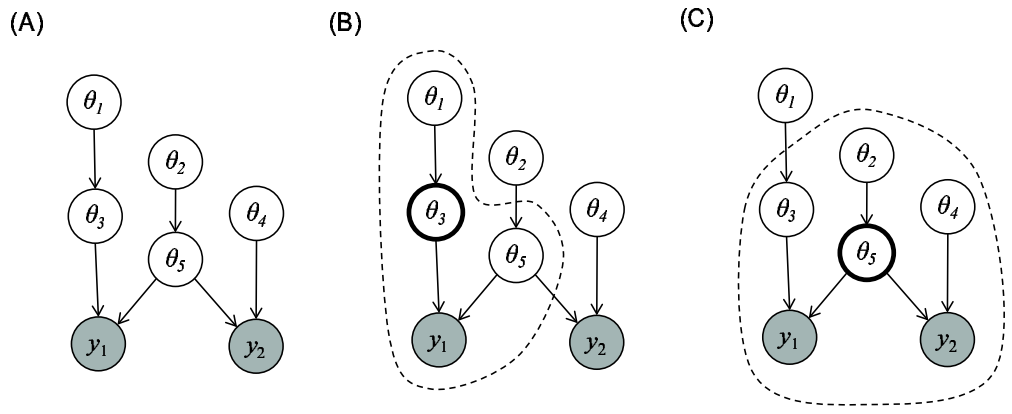

Figure 5.3: Illustration of Markov blankets for an example graphical model (A). The Markov blankets for θ_3 and θ_5 are shown in (B) and (C) respectively.

Now, note that we are looking for a distribution for θ_3. Any term on the right-hand side that does not depend on θ_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 θ_3, θ_1, θ_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 5.3, panel B shows the Markov blanket for θ_3. We can use similar reasoning to derive the Markov blanket for θ_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*.

5.2.3 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

$$\begin{aligned} y_{ij} &\sim \text{Normal}(\mu, 1/\sqrt{\tau_j}) \\ \tau_j &\sim \text{Gamma}(a, b) \end{aligned} \tag{5.2}$$

where we have removed the latent variable σ_j and instead directly placed the precision term τ_j into the normal distribution. Therefore, we have the latent variables τ_j, where $j = 1, \ldots, M$, and the latent variable μ. 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 $p(\tau_j|\boldsymbol{y}, \boldsymbol{\tau}_{-j}, \mu, a, b)$ to get values for the individual precision variables. Using Markov blankets, we can derive that $p(\tau_j|\boldsymbol{y}, \boldsymbol{\tau}_{-j}, \mu) = p(\tau_j|y_{1j}, \ldots, y_{Nj}, \mu, a, b)$. We also

need to sample from a conditional distribution $p(\mu|\boldsymbol{y}, \boldsymbol{\tau}, a, b)$. Again, using the conditional independence relations, we can derive that $p(\mu|\boldsymbol{y}, \boldsymbol{\tau}, a, b) = p(\mu|\boldsymbol{y}, \boldsymbol{\tau})$. This leads to the following general MCMC procedure where the state at iteration t consists of sampled values $\boldsymbol{\tau}^{(t)}$and $\mu^{(t)}$:

1. Set $t = 1$
2. Generate initial values $v_1, \ldots, v_M$ for the precision variables $\boldsymbol{\tau}$
3. Set the initial state $\boldsymbol{\tau}^{(t)} = \boldsymbol{v}$
4. Generate initial value v_{M+1} for variable μ
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 $\boldsymbol{\tau}^{(t)} = \boldsymbol{v}$

   Sample $v_{M+1} \sim p(\mu|\boldsymbol{y}, \boldsymbol{\tau} = \boldsymbol{\tau}^{(t)})$

   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:

$$\tau_j|y_{1j}, \ldots, y_{Nj}, \mu, a, b \sim \text{Gamma}\left(a + \frac{N}{2}, \frac{1}{b + \sum_{i=1}^{N}(y_{ij} - \mu)^2/2}\right) \tag{5.3}$$

We can also use conjugacy to derive a simple form for the conditional distribution for μ:

$$\mu|\boldsymbol{y}, \boldsymbol{\tau} \sim \text{Normal}\left(\frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\tau_j y_{ij}}{N\sum_{j=1}^{M}\tau_j}, \frac{1}{\sqrt{N\sum_{j=1}^{M}\tau_j}}\right) \tag{5.4}$$

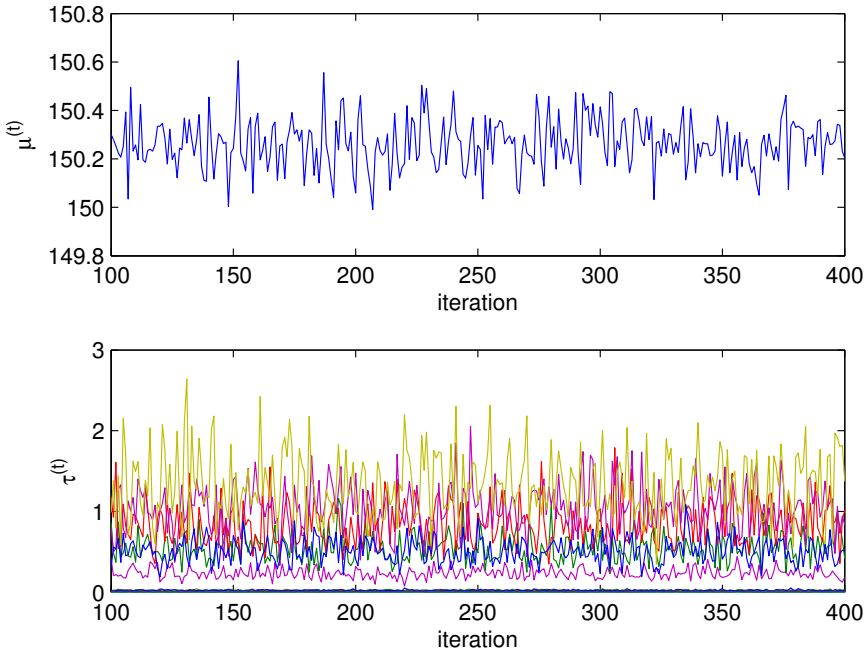

Figure 5.4: Illustration of the Gibbs sampler for the consensus model. Top panel shows the samples drawn for μ. Top panel shows the samples drawn for all individual precision variables.

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 5.4 shows the samples drawn from the posterior distribution over μ and $\boldsymbol{\tau}$ for this synthetic dataset. Note that the first 100 samples were discarded and that the sampler was run for 400 iterations. Figure 5.5 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 Figure5.4 and Figure 5.5.

2. What is the inferred mean when you average the sampled values for the last 300 iter-

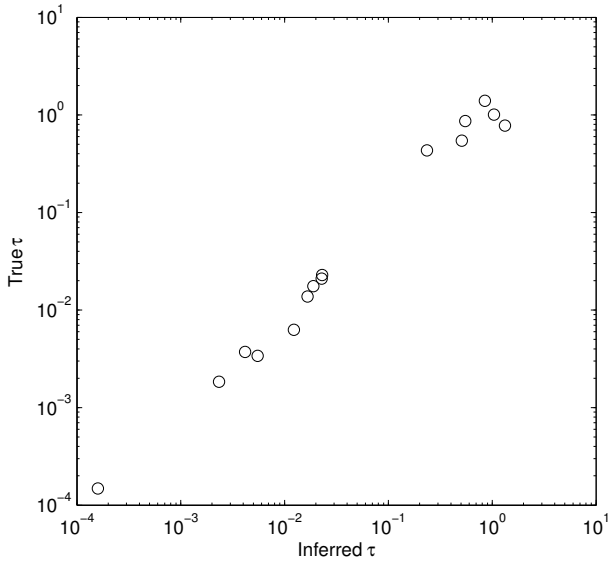

Figure 5.5: Correspondence between the inferred precisions for each (simulated) individual and the true values that were drawn in the prior predictive distribution

ations? 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?

5.2.4 Example: Posterior Inference for the consensus model with contaminants

The beauty of graphical models is that it is often easy to make the model more psychologically plausible and applicable to real-world data. For example, suppose in the consensus model that the estimates generated by individuals occasionally contain noise that are unrelated to the true distribution. This could be because individuals might be mistyping numbers, forgetting what the question was, or any number of reasons that are deeply troubling to researchers who conduct these experiments. Let's introduce a new latent variable in the model, x_{ij} that represents the status of each datapoint. If $x_{ij} = 1$, the ith estimate for individual j is assumed to be a contaminant, and when $x_{ij} = 0$, the estimate is generated from the consensus model. Let's assume that if a contaminant is produced, it is sampled from a Uniform(y_{min}, y_{max}) distribution. Finally, let's assume that γ is the probability that a particular estimate is a contaminant – we will assume that this probability is the same across all individuals. This leads to the following model:

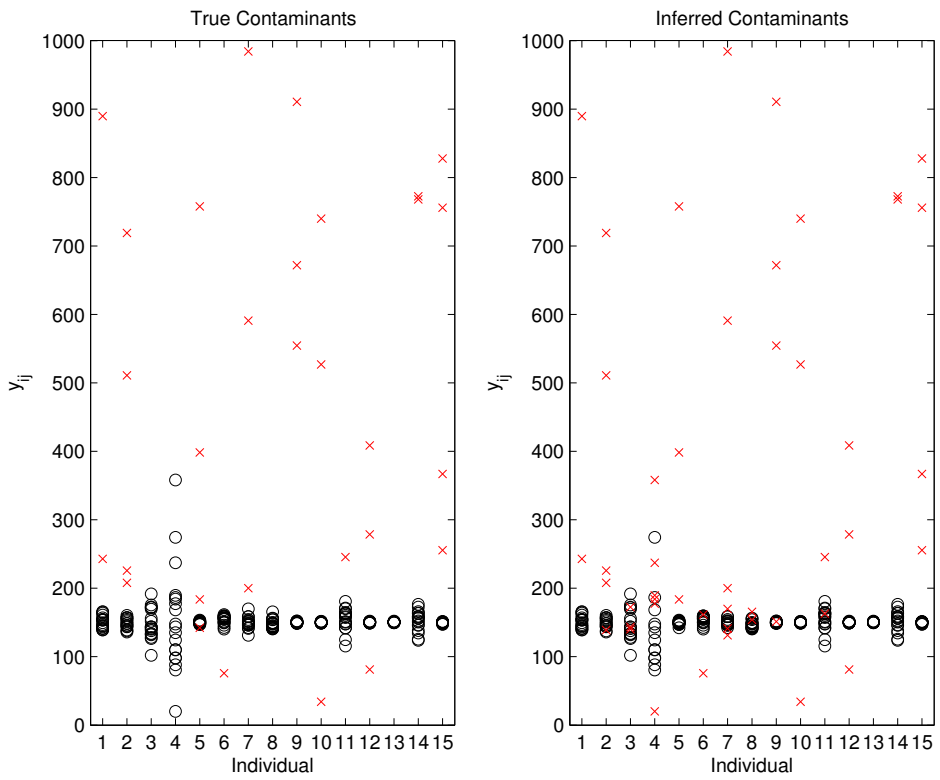

Figure 5.6: Illustration of the data from the consensus model with contaminants. The left panel shows the synthetic data produced from the model where the cross symbols indicate contaminants. The right panel shows the same data but with the inferred contaminants at the 400th iteration from a Gibbs sampler.

$$
\begin{aligned}
x_{ij} &\sim \text{Bernoulli}(\gamma) \\
y_{ij} &\sim \begin{cases} \text{Norm}(\mu, 1/\sqrt{\tau_j}) & x_{ij} = 0 \\ \text{Uniform}(y_{min}, y_{max}) & x_{ij} = 1 \end{cases} \\
\tau_j &\sim \text{Gamma}(a, b)
\end{aligned}
\tag{5.5}
$$

Exercises

1. Create a synthethic dataset from the consensus model with contaminants. You can keep the same parameters as before (a=0.3, b=3.33, $\mu = 150$, M=15, N=20) and assume that y_{min}=0, y_{max}=1000, and $\gamma = 0.1$. Visualize the data as in Figure 5.6, left panel where the cross marks indicate the estimates associated with contaminants.

2. Consider the problem of posterior inference for the consensus model with contaminants. It might be helpful to first consider the inference for the latent variables τ_j and μ. Suppose we actually tell the inference algorithm what the true values for x_{ij} are. How should we change the Equations 5.3 and 5.4?

3. Next we need to know how to calculate the conditional probability $p(x_{ij}|y_{ij}, \mu, \tau_j)$. Use Bayes rule to find the two terms that this probability is *proportional* to. Write out how this conditional probability can be evaluated when $x_{ij} = 1$ and $x_{ij} = 0$.

4. Write the program that performs the posterior inference over all latent variables τ_j, μ, and x_{ij}. Use the last iteration of the sampler to visualize the inferred contaminants similar to Figure 5.6.

** 5 Instead of taking just a single sample from the Gibbs sampler to interpret the data (as we did in the last exercise), it is much better to average across many samples. For example, we can calculate the *probability* that a particular estimate was assigned to the contaminant model. Calculate this probability and modify the visualization to indicate this probability for each estimate.

** 6 Consider the basic consensus model again without contaminants. We have only considered the Normal distribution for data generation. In spite of many comments in the literature that the Normal distribution is a natural choice to model real-world data, it is often the case that many measurements are not Normally distributed. They might be restricted to positive numbers only or might have heavy tails, leading to extreme measurements. Change the model to incorporate different data generating distributions, such as the Gamma distribution (which is bounded at zero) and the Student-t distribution (which has heavy tails). With these distributions, we can no longer use conjugacy to sample from "nice" distributions. However, we can use the Metropolis-Hastings sampler to sample each individual latent parameter conditional on the state for all other latent parameters. Implement such a sampler. Once you have reached this stage, I can give you some real estimation data to test your model on.

Chapter 6

Sequential Monte Carlo

In the preceding chapters, we have investigated inference methods that allow researchers to apply probabilistic models to data in a posthoc fashion – the modeling can start as soon as the data collection has finished. This is known as *batch processing*. In many applications however, there is no natural end point to the data collection. The data arrive gradually over time and the goal is to draw inferences about latent variables given the data observed up to the current point in time. In this case, we could still opt to apply batch processing methods such as MCMC. For example, in a naive application of MCMC methods, we could rerun the inference algorithm every time we get a new observation. Although this procedure leads to proper inferences about the data, it is computationally inefficient. It requires the explicit storage of all data observed to this point which might be impractical if we are dealing with situations in which large amounts of data needs to be processed. More importantly, processing time increases with every new observation because the batch processing procedure is applied to *all* data observed up to this point. For this reason, it is desirable to use inference methods where the processing time does not depend on time itself – each new observed data point takes a constant amount of processing time.

Sequential Monte Carlo (SMC) methods are a general class of Monte Carlo sampling techniques for sequential inference problems. They allow for the *online processing* of data which lead to time and memory requirements that are constant in the total number of observations received over time. A typical application for SMC methods are tracking problems (also known as *filtering* problems) in robotics and vision, where the goal is to estimate the state of a system that changes over time using a sequence of noisy measurements. For example, in object tracking, the goal is to continually track the position of an object on the basis of noisy measurements about the object's position. In this case, it is not sufficient to only use the latest measurement in the inference about the current position. Objects typically move smoothly over time so all recent measurements are useful to make inferences about the current position.

Recently, SMC techniques have also been introduced in other areas of cognitive science including language, decision-making and categorization. For example, Pearl, Goldwater and Steyvers have recently used SMC methods for online word segmentation problems, where

the problem is to extract words from a continuous stream of speech. Brown and Steyvers applied SMC methods such as particle filters to model decision-making strategies in non-stationary environments. Griffiths and Sanborn investigated how particle filters can explain order effects observed in categorization tasks. Finally, Vul has applied particle filters to model human performance in an object-tracking task.

In this chapter, we will discuss two SMC methods to estimate the hidden state of a dynamically changing system: *decayed MCMC* and *particle filters*. As the name suggests, sequential Monte Carlo approaches are based on sampling – the hidden state can be estimated on the basis of a set of samples that approximate posterior distributions. We will not go into methods such as Kalman filters and Extended Kalman filters because they make Gaussian assumptions about state transitions and measurement noise that are often unrealistic. In addition, the computational overhead to estimate Kalman and Extended Kalman filters might also be prohibitive in some instances. One desirable feature of SMC approaches such as particle filters is that the approximation can be made arbitrarily precise by varying the number of particles. For some low-dimensional problems, very good approximations can be obtained with a small number of particles (e.g. 50) leading to computationally efficient procedures.

Exercise

1. To get a good sense of the tracking problem in robotics and vision, watch the YouTube videos named "Particle filter tracking with adaptive parameters estimation" and "SIR particle filter using color statistics". If you want to see a funny tracking application involving belly dancing, do a search for "Human Motion Tracking - Belly".

6.1 Hidden Markov Models

A standard approach to probabilistic modeling with dynamics is to use hidden Markov models (HMMs). In an earlier chapter, we already introduced Markov models. The main difference between Markov models and HMMs is, not surprisingly, that the state in an HMM is hidden (i.e. a latent variable).

The graphical model of a HMM is shown in Figure 6.1. The variable x_t represents the latent system state of the system at time t. In object tracking, the state could correspond to the position and direction of the object. The variable y_t represents the observed set of measurements at time t. In object tracking applications, measurements could relate to any kind of sensor data. In the graphical model, we have assumed that the system state evolves over time according to $p(x_t|x_{t-1})$, a first order Markov process where the state of system at time t only depends on the state at time $t-1$. For example, we can assume that the object's new position is only related to the last position (i.e., it is a memoryless system). It is also assumed that the noisy measurements y at time t are determined by $p(y_t|x_t)$. Therefore, the observations at time y are determined solely by the state of the system at time t – the

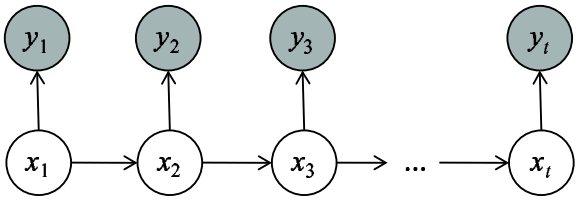

Figure 6.1: Graphical model

measurements are conditionally independent given the system state. Finally, to complete the model, we also need to specify a prior (x_0) over states at the start of measurement.

6.1.1 Example HMM with discrete outcomes and states

To get started with an simple HMM, suppose we have a model that tracks the emotional state of a person. In this model, the hidden state is discrete and consists of two possible outcomes: "Happy" and "Sad". We will assume that the emotional state of a person can change over time, sometimes unpredictably (as in real life). We will use a simple change process where with probability a, the state changes and with probability $1 - a$, the state remains the same. For example, when $a = 0.1$, we expect that on average 1 out of 10 trials will be associated with a change in emotional state. We also need to specify a prior $p(x_0)$ over states at the first time step which we assume to be uniform. To complete the model, we will also assume that the observations are discrete and consist of only two possible outcomes: "Smile" and "Grimace". As you might suspect, we parametrize the model such that smiles are more likely in a happy state and grimaces more likely in a sad state. Specifically, we will assume that with probability b, a person produces an outcome that is consistent with their emotional state. Instead of using symbols, we will represent "Happy" and "Smile" by the number 2 and "Sad" and "Grimace" by the number 1. Therefore, $x_t = 2$ represents a happy state at time t and $y_t = 1$ represents an observed grimace at time t. This leads to the following HMM:

$$\begin{aligned} p(x_t = j|x_{t-1}) &= \begin{cases} a & j \neq x_{t-1} \\ (1-a) & j = x_{t-1} \end{cases} \\ p(y_t = k|x_t) &= \begin{cases} b & k = x_t \\ (1-b) & k \neq x_t \end{cases} \end{aligned} \tag{6.1}$$

Figure 6.2 shows a synthetic dataset produced by the model over 200 time steps when $a = 0.1$, $b = 0.8$. Note that smiles are more likely in the happy state but can also occasionally occur in the sad state. Also, notice how the latent state changes 14 times over a period of 200 trials.

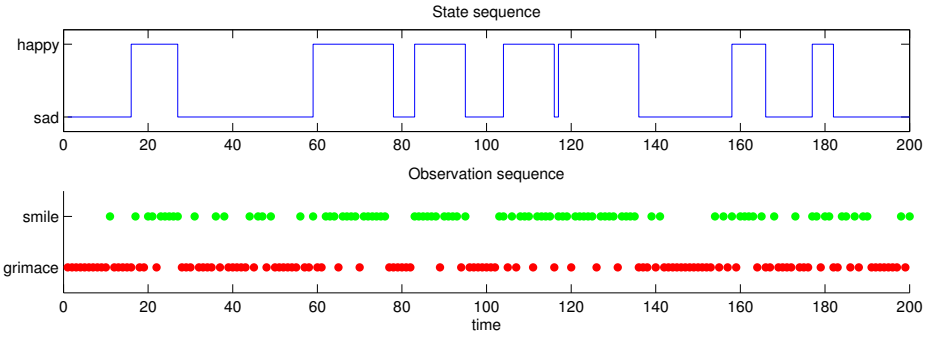

Figure 6.2: Sequence of hidden states and observed outcomes produced by an HMM

Exercises

1. Write a Matlab program that generates a synthetic dataset from the HMM model specified above. Use $a = 0.1$, $b = 0.8$ and generate an output sequence for 200 iterations. Visualize results in a way similar to Figure 6.2. Save the resulting output sequence as well as hidden state sequence to a file.

2. Repeat the last exercise but use the Matlab function `hmmgenerate` to create the output sequence.

6.1.2 Viterbi Algorithm

Before we go into SMC techniques, we will first discuss a standard batch processing approach to estimate hidden states in HMMs: the Viterbi algorithm. This algorithm estimates the state sequence $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_t$ that is most likely to have generated the output sequence $y_1, y_2, \ldots, y_t$. We will not go into the details of the operation of the algorithm but just note a few of its properties. First, the algorithm requires knowledge of the probabilities $p(x_t|x_{t-1})$ as well as $p(y_t|x_t)$. Therefore, it needs to know how states can evolve over time and how observations are produced from the hidden state. Second, the Viterbi algorithm is a maximum likelihood estimator of the posterior – it only gives the mode of the posterior distribution over state sequences but it does not give the full posterior distribution or samples from the posterior distribution as with MCMC methods. Finally, the Viterbi algorithm returns the estimate of the *full* state sequence. Therefore, it gives estimates for all hidden states up to the current time step. This is in contrast to SMC algorithms that are designed to estimate only the current hidden state.

Figure 6.3 shows the estimated state sequence from the Viterbi algorithm applied to the data from the example HMM for emotional states. Note that the algorithm is fairly accurate in recovering the state sequence, but the results are obviously dependent on the parameterization of the model.

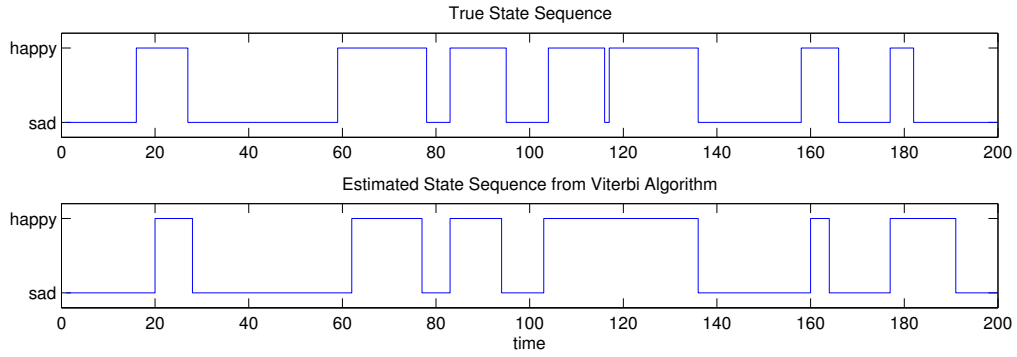

Figure 6.3: Estimated state sequence for example HMM compared with the true state sequence

Exercises

1. Write a Matlab program that estimates the most likely state sequence from a synthetic dataset produced in the last exercise. Matlab implements the Viterbi algorithm with the function `hmmviterbi`.

2. Suppose we measure the accuracy of the estimate by the percentage of cases where the estimated state matches the true state. What is the accuracy according to this measure?

6.2 Bayesian Filtering

In many applications of HMMs, the observed data arrives sequentially over time leading to online inference problems such as *Bayesian filtering*. In this case, we want to estimate the posterior distribution $p(x_t|y_1, \ldots, y_t)$. This is the distribution over the just the current state at time t given all measurements up to and including time t. Note that we are *not* asking here about the full history of the system state, but only the latest state. If the full history is needed, it might be best to use MCMC methods. In the filtering task, SMC methods can be applied efficiently because the results of the inference procedure at time t can be re-used again at time $t + 1$.

Exercises

1. In this exercise, we will explore the use of the Viterbi algorithm as a batch processing approach to the filtering task. Change the Matlab code from the last exercise to implement the Bayesian filtering task. Therefore, simulate the case where the data arrives sequentially over time and at each time, the Viterbi algorithm is applied to all data observed up to time t where t varies from 1 to 200. You can use the estimate of the last hidden state as a estimate of the filtered state. Note that you will therefore

apply the Viterbi algorithm exactly 200 times. Visualize the filtered estimates over time by plotting the sequence of estimated hidden states up to t as a function of t (essentially you are creating a triangular display). Compare this with the true state sequence.

2. Estimate the time it takes at each iteration of the algorithm. What is (roughly) the relationship between the size of the dataset t and the time it takes to get a filtered estimate with the Viterbi algorithm? What is the problem highlighted by this relationship?

3. Measure the accuracy in the filtering task defined by the percentage of cases where the filtered estimate matches the true hidden state. Why is the accuracy in this case lower than in the exercise where the Viterbi algorithm was applied once to the full dataset.

6.3 Particle Filters

In the SMC approach to filtering, the posterior distribution over the current latent state is based on an approximation from a finite set of samples. Particle filtering is one form of SMC. In particle filters, each particle represents a sample or hypothesis about the current latent state. As the number of particles increases, the approximation improves and starts to resemble to the exact Bayesian posterior. Although a low number of particles is not desirable for engineering applications, as it can lead to poor approximations, they can be very useful to explain suboptimal human performance in a number of sequential tasks (e.g. Brown and Steyvers, 2009; Yi, Steyvers, and Lee, 2010). Note that particle filtering does not refer to a particular algorithm that rather a general approach – there are many different methods to implement particle filters.

A particularly attractive feature of particle filters is that no memory of previous states is needed. All the information needed for the filtering task is contained in a fixed set of particles leading to a constant memory demand over time. However, when applying particle filters to high-dimensional problems, a very large number of particles might be needed to achieve good performance. This can in some cases lead to very slow performance, especially when overly simplistic particle filters are used.

6.3.1 Sampling Importance Resampling (SIR)

One of the most basic particle filters is Sampling Importance Resampling (SIR). It is based on the idea of importance sampling where a distribution is approximated by a set of samples weighted by the density of the target function such that the dense regions of the posterior are more heavily weighted than the tails. In the SIR filter, we always work with a fixed set of M particles. To introduce notation, let x_t^i represent the hypothesis about the latent state of the ith particle at time t. Across particles, we can approximate the distribution of the filtered estimate at time t by the set of particles $\{x_t^i\}$. If a point estimate is needed for the

latent state, we can take the mode or mean of this distribution. The SIR particle filter as described below is desirable for its simplicity. It only requires that one can sample from the state transition distribution $p(x_t|x_{t-1})$ and can evaluate the likelihood $p(y_t|x_t)$.

We initialize the particle filter by sampling each particle from the prior distribution $p(x_0)$. At each iteration, we then evolve the set of particles from the last generation. First, we sample for each particle from the last generation a candidate hypothesis for the *current* latent state. Therefore, each particle leads to a projection of the hidden state to the current time step. We then calculate the likelihood of the current observation y_t under each particle. This likelihood serves as the *importance weight*. We then normalize the importance weights across particles. In the second step, we evolve the set of particles from the last generation by sampling, with replacement, from the candidate hypotheses weighted by the importance weights. Therefore, candidate hypotheses that make the current observations likely lead to higher importance weights and are more likely to be sampled. This resampling continues until M new particles are generated. The procedure can be summarized as follows:

1. At time $t = 0$, initialize each particle $x_0^i \sim p(x_0)$ where $i = 1, \ldots, M$
2. Repeat

   $t = t + 1$

   For i=1:M

   Generate a proposal $x^{*i}_t \sim p(x_t|x^i_{t-1})$

   Calculate a weight $w_t^i = p(y_t|x^{*i}_t)$

   End

   Normalize weights, $w_t^i = w_t^i / \sum_{i'=1,\ldots,M} w_t^{i'}$

   Sample M new particles x_t^i with replacement from $\{x^{*i}_t\}$ with weights w_t^i
3. Until $t = T$

Note that in the resampling step, the same proposal can be sampled multiple times. Also, a particular proposal might not be sampled at all. Therefore, each particle can lead to a varying number of "offspring" depending on the proximity of particle to the region in state space that best explains the current observation – bad particles die out and are taken over by particles that do (at least for the current time step) explain the data.

In this particular version of SIR, the proposal distribution equals the state transition distribution. This version of SIR is also known as the *bootstrap* or *condensation* algorithm. It is important to realize that this choice of proposal distribution can lead to problems with high-dimensional cases. This is because in the first step of the SIR approach, we sample proposals from the state transition distribution *without* taking the current observation into account. The current observation is taken into account only in the second step, by weighting each particle by the likelihood. Therefore, we are blindly proposing how the current hidden state evolves from the last state and then correct only *afterwards* for any unlikely proposals.

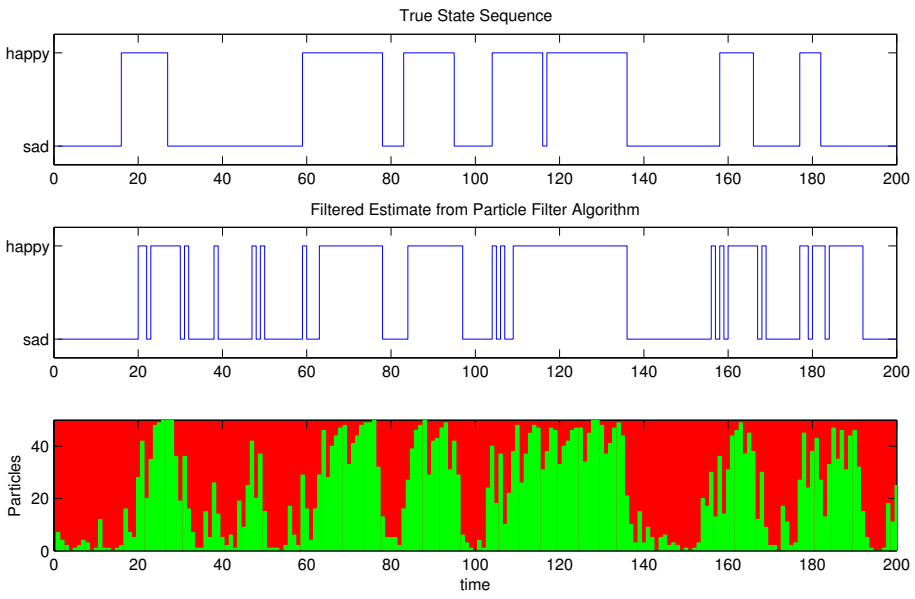

Figure 6.4: Predictions from the Sequential Importance Resampling (SIR) particle filter with $M = 50$ particles. The bottom panel shows the distribution over particles at each time step. For the middle panel, we have taken the mode of the distribution as the point estimate for the hidden state.

In some cases, this leads to a situation where none of the candidate proposals come near the regions in the state space that can adequately explain the current observation. In other versions of SIR particle filters, more complex proposal distributions are used that do take the current observation into account (in the proposal generation). Also, some authors have proposed methods such as *unscented Kalman filters* to deal with this problem.

Figure 6.4 shows the results of the SIR filter applied to the example data. The number of particles, M was set to 50. The bottom panel shows the distribution over particles for each iteration. The middle panel shows the mode of the particles to create a point estimate for the latent state at each time t. Importantly, remember that in the filtering task, a filtering estimate at time t implies that all data up to and including time t has been observed.

6.3.2 Direct Simulation

The method of direct simulation is a slight improvement over the simple SIR approach. Similar to the SIR approach, there are two steps, one to sample proposals for the current hidden state and one to assess the likelihood of the data given each proposal. However, we reject any proposed particles if they lead to unlikely predicted distributions for the observed data. Therefore, we are simultaneously weighting the "prior" distribution from the state transitions and the "evidence" from the observed data. Here is a summary of the procedure:

1. At time $t = 0$, initialize each particle $x_0^i \sim p(x_0)$ where $i = 1, \ldots, M$
2. Repeat

   $t = t + 1$

   Set the particle count m=0

   Repeat

   Sample a random particle $i \sim \text{Uniform}(1, \ldots, M)$

   Sample a proposal $x^* \sim p(x_t|x_{t-1}^i)$

   Sample $u \sim \text{Uniform}(0, 1)$

   If $u < p(y_t|x^*)$, accept the proposal, set $m = m + 1$, and set $x_t^m = x^*$

   Until $m = M$
3. Until $t = T$

Exercises

1. Write a Matlab program that implements the SIR particle filter and the direct simulation method. Apply each particle filter with $M = 50$ particles to the emotional states dataset. Use the particle filters to estimate the hidden state at each time t up to $t = 200$. You can assume a uniform prior for $p(x_0)$.
2. Look up the paper "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking" by Arulampalam et al. (2002) and carefully read about the example problem on page 183. Implement the dynamic system given by Equations 79-83 to create an artificial dataset. Apply the SIR and direct simulation particle filter to this data.