Chapter 4

Monte Carlo Markov Chains

4.1 Stationary distributions in Markov Chains

Markov chains corresponds to a general model of dependence in a sequence of random elements $X_0, X_1, \ldots, X_n, \ldots$. The characteristic feature of the Markov model is the sequential dependence of an element X_n on the previous observations $X_0, \ldots, X_{n-1}$ only through the immediate predecessor X_{n-1}. In other words, the conditional density satisfies:

$$f(x_n|x_0, \ldots, x_{n-1}) = f(x_n|x_{n-1}) ,$$

for all n. Observe that joint densities can be represented in terms of conditional densities:

$$f(x_0, \ldots, x_n) = f(x_0) \prod_{i=1}^{n} \frac{f(x_0, \ldots, x_i)}{f(x_0, \ldots, x_{i-1})} = f(x_0) \prod_{i=1}^{n} f(x_i|x_0, \ldots, x_{i-1}) ,$$

which reduces in the Markov model to:

$$f(x_0, \ldots, x_n) = f(x_0) \prod_{i=1}^{n} f(x_i|x_{i-1}) .$$

Consequently, the joint distribution in the Markov setting is determined by the initial distribution $f(x_0)$ of the first observation and by the transition kernels $f(x_n|x_{n-1})$, $n = 1, 2, \ldots$.

In principle, the function form of the transition kernels $f(\cdot|\cdot)$ may depend on n. In the case where the same kernel is used for all n we say that the Markov process is *homogeneous*. Henceforth we will be considering only homogeneous Markov chains.

Of particular interest to us will be the marginal distributions of the elements X_n for different n's. Notice that the marginal distribution of the first element in the sequence is $f_0(x) = \mathbb{P}(X_0 = x)$. In order to obtain the marginal density of the second element notice that the joint density is given by

$$\mathbb{P}(X_1 = x, X_0 = y) = f(x|y) f_0(y) .$$

therefore, upon integration with respect to the values of X_0 we get that

$$f_1(x) = \mathbb{P}(X_1 = x) = \int f(x|y) f_0(y) dy .$$

In general, we get the recursive formula:

$$\mathbb{P}(X_n = x) = f_n(x) = \int f(x|y) f_{n-1}(y)dy \; . \tag{4.1}$$

Consider (4.1) as a functional equation. Under appropriate regularity conditions that will not be discussed here it can be shown that this equation has a unique solution, which we denote by $\pi(x)$. This solution is itself a density and it satisfies the relation

$$\pi(x) = \int f(x|y)\pi(y)dy \; . \tag{4.2}$$

Notice that this relation implies that if X_0 has π as its initial distribution then all subsequent elements of the sequence $X_1, X_2, \ldots$ share the same marginal density π. In such a case we say that the Markov process is *stationary.*

Stationary Markov processes in the world of Markov chains are the equivalent to identically distributed random variables (i.i.d.) in the world of independent random variables. In particular, one may expect properties like the law of large numbers to hold. Indeed, if $X_0, X_1, \ldots$ is a stationary Markov process then one can show (under an appropriate collection of regularity conditions, that typically come under the heading of ergodicity) that for any integrable function g:

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} g(X_i) = \int g(x)\pi(x)dx \; . \tag{4.3}$$

Another important limit theorem unique to Markov processes is the convergence of the marginal distributions to the stationary distribution. This theory identifies the fact that an iterative application of (4.1) produces a sequence of densities that converges to the stationary density π, regardless of the initial distribution f_0 used. Hence, if a long enough sequence is produced from the Markov process then the tail elements $X_m, X_{m+1}, \ldots$ of this sequence, when m is large, is essentially stationary. Consequently, if this convergence property holds then relation (4.3) formally holds even if the process was not initiated with the stationary distribution π, but with some other distribution.

The bottom line for Markov processes that posses the regularity conditions that assures convergence to the stationary distribution: An approximation to an integral with respect to the stationary distribution π can be obtained by generating the Markov sequence $X_1, X_2, \ldots, X_n$ and using the average $\sum_{i=m}^{n} g(X_i)/(n-m)$. The generation of the Markov process can be initiated with any starting value x_0. After a "burn down" period of length m, observations are assumed to approximately come from the stationary distribution in which (4.3) is applicable.

The generation of the Markov sequence is carried out with the aid of the kernel $f(x|y)$. Starting from the initial value x_0, one generates X_1 from the distribution $f(\cdot|x_0)$. Given the value of X_1, one generates X_2 from the distribution $f(\cdot|X_1)$, and so on.

4.2 MCMC algorithms

Markov Chain Monte Carlo (MCMC) exploits the properties of Markov chains that were discussed in the previous section in order to conduct simulations aimed at the approximation of integrals of a given distribution. Unlike regular simulations, in which independent samples are generated from the target distribution, in MCMC a Markov process is produced. The target distribution is the stationary distribution of the generated process. Hence, approximation (4.3) in exactly the same way means are used in ordinary simulations.

As an illustration, assume that we are interested in the approximation of the probability of an event A:

$$\mathbb{P}(A) = \int_A f(x)dx = \int I_A(x)f(x)dx \; .$$

Standard simulations will approximate this probability by the generation of independent observations $X_1, X_2, \ldots, X_n$, such that $X_i \sim f(x)$ and the computation of empirical frequency $\sum_{i=1}^n I_A(X_i)/n$ of the event $X_i \in A$ as an approximation of the given probability. The MCMC approach, on the other hand, will generate a Markov sequence with f as a stationary distribution and approximate the probability using $\sum_{i=m}^n I_A(X_i)/(n-m)$.

MCMC algorithms are designed in order to generate a Markov process with the target distribution as the stationary distribution. Many algorithms exist, each geared towards solving the given problem at different scenarios and contexts. The basic principle in all algorithms is the same: the construction of a transition kernel (or matrix in the case where the number of states is finite) in which (4.2) holds with π being the distribution we are trying to simulate from. This kernel is required to possess further ergodicity conditions that will ensure that the given solution is the only solution and that the marginal distributions converge to that solution so that (4.3) holds. After the initiation of the Markov process at some starting point, the kernel is applied in order to simulate sequentially a realization of the Markov chain. An approximation of the expectation of any function with respect to the target distribution can be obtained by the application of the function to the elements of the generated sequence and taking the average. Usually, only the tail elements of the sequence are used for the approximation since they are more likely to represent the target limit distribution.

Here we will consider an example that involves the application of a specific such algorithm: the *Gibbs algorithm*. The Gibbs algorithm is designed for the approximation of integrals of marginals of multi-dimensional distributions. The algorithm is useful when generation of samples from the conditional distribution of each component, given the other components, is relatively simple but an attempt to produce the marginal distribution directly is complicated, possibly due to the high dimensionality of the joint distribution. In the application that we will consider the distribution will be of high dimension. However, in order to explain the concept, let us consider a two-dimensional case.

Let $f(x, y)$ be some joint density and consider the marginal density $f_1(x) = \int f(x, y)dx$. We are interested in the simulation of samples. In order to do that we seek an algorithm that will produce a Markov sequence $x_0, x_1, x_2, \ldots$ with f_1 as its stationary distribution. Starting with some x_0 we will produce x_1 in two steps: first we will generate y_1 from the conditional distribution $y_1 \sim f(y|x = x_0)$ of the second margin, given the first one. After doing so we will use the conditional distribution of the first component, given the second one in order to generate an x value: $x_1 \sim f(x|y = y_1)$. Thus, we have produced the second element in the sequence that has x_0 as its first element. The third element x_2 is produced in the same way, with x_1 taking the role of x_0. Hence, one first produces y_2 from the conditional distribution, given x_1 and then x_2 from the conditional distribution, given y_2. All in all, $2n$ simulations from conditional distributions are used in order to produce a sequence of length $n+1$ (including x_0) of x elements.

Clearly the sequence that is produced is Markovian. The distribution of x_n depends only on the value of y_n, which depends in turn only on the value of x_{n-1}. Consequently, the x_n element depend on the previous x elements only through the value of x_{n-1}.

The kernel of this Markov process is given in the form of a conditional of an element in the sequence, say x_1, given the value of its predecessor, say x_0. Notice, that the joint density of x_1 and y_1, conditional on the value of x_0 is given by $f(x_1|y = y_1)f(y_1|x = x_0)$. Consequently, the conditional density of x_1 given x_0, i.e. the kernel of the Markov sequence, is obtained by integration over the values of y_1:

$$k(x_1|x_0) = \int f(x_1|y = y_1)f(y_1|x = x_0)dy_1$$

Let us show that relation (4.2) holds for this kernel and for $\pi = f_1$. Indeed, since

$$f(x_1|y = y_1) = \frac{f(x_1, y_1)}{\int f(x, y_1)dx} , \quad f(y_1|x = x_0) = \frac{f(x_0, y_1)}{f_1(x_0)} ,$$

we get that

$$k(x_1|x_0)f_1(x_0) = \int \frac{f(x_1, y_1)f(x_0, y_1)}{\int f(x, y_1)dx}dy_1$$

and when we integrate with respect to x_0 we obtain, upon changing the order of integration, that:

$$\begin{aligned}\int k(x_1|x_0)f_1(x_0)dx_0 &= \iint \frac{f(x_1, y_1)f(x_0, y_1)}{\int f(x, y_1)dx}dy_1dx_0 \\ &= \int \frac{f(x_1, y_1)\int f(x_0, y_1)dx_0}{\int f(x, y_1)dx}dy_1 \\ &= \int f(x_1, y_1)dy_1 = f_1(x_1) ,\end{aligned}$$

establishing the claim that f_1 is the stationary distribution for the given kernel.

The construction of the Gibbs sampler can be generalized in several ways. First, it may be noted that the two components x and y can themselves be vectors. In which case sampling is carried out with respect to the conditional distribution of the vector y, given the entire vector x, and vice versa. Secondly, it follows from symmetry that the sequence $y_1, y_2, \ldots$ is a Markov process and has as its stationary distribution the marginal distribution of y. Thirdly, if the joint distribution has k components and one knows how to simulate a value from the conditional distribution of one component, given all the other, then one can generate in parallel k Markov processes by changing the value of each of the components, one at a time, and completing a Markov iteration after k steps. The resulting Markov processes has as their stationary distribution the marginal densities of the different components.

In the next section we demonstrate the application of the Gibbs sampler for the investigation of a specific genetic problem.

4.3 Identifying population of origin

Consider the following cluster analysis problem: A population is composed of K genetically distinct sub-populations. We are given a sample of N unlabeled individuals from that population. The task is to cluster these individuals into K clusters, which are genetically homogeneous within each cluster and genetically heterogeneous between clusters, based on the genotypic measurements taken over L loci. The approach we present here is due to Pritchard, Stephens, and Donnelly (Genetics 155:945-959, June 2000) and is based on Bayesian considerations. Their basic proposal is to place a prior distribution on the population origin of each individual and to compute the posterior distribution, given the genotype. Clustering can then be carried out based on these posterior probabilities, assigning each subject to the sub-population with highest posterior probability. The Gibbs sampler is applied in order to compute these posterior probabilities.

Let us consider the components of the problem. The observations are the genotypes. With each individual and each locus an unordered pair (X_1, X_2) of observed alleles is given. In favor of the convenience of the presentation we will act in the sequel as if the pair is ordered and separate the component X_1 from X_2. However, all functions that we will be using will be invariant with respect to the permutation of the two elements. Hence, in actuality, the outcome will be a function of the unordered pairs. The two `R` object `X1` and `X2` will be $N \times L$ matrices, with rows representing the individuals and columns representing the loci. The entries to the matrices will be zeros or ones, representing the absence or presence of the designated allele in each locus.

The second component is a vector `Z` of length N representing the sub-population origin of each subject. The entries to this vector are the numbers $1, \ldots, K$. The entry for each subject is a-priory unknown and will be treated as a random quantity with the uniform distribution over the the K possible values. Our goal is to sample from the posterior distribution of these entries, given the observed genotypes, in order to compute the posterior probability for each subject.

The last component is the $K \times L$ matrix of the designated alleles sub-populations frequencies. The rows correspond correspond to sub-populations and the columns to loci. The entries are numbers between zero and one. These parameters are also unknown and taken to be random. Their a-priory distribution is beta. The Gibbs algorithm corresponds may be used to produce sample from their posterior distribution, given the genotype. Our main concern, however, will not be the distribution of `P` but, rather, the distribution of `Z`. Nonetheless, sampling `P` values will be part of the algorithm.

The Gibbs MCMC sampling will be initiated by setting starting values for the components of the vector `Z`. New values of the vector will be generated in two steps that for together a single iteration of the Gibbs sampler. In the first step the matrix `P` is generated from its conditional distribution, given the current sub-population identifier of each subject and given the genotypes. In the second step new identifiers will be generated from the conditional distribution of `Z`, given the values of `P` from the first step and given the genotypes. These two steps will be reiterated, each time with the updated values of identifiers `Z` in order to produce a Markovian sequence of vectors of identifiers with marginal distribution that converges to the posterior distribution.

Let us build the R code and explain the details of the steps at the same time with the aid of a small numerical example. Let us consider an artificial example that involves 10 subjects, 5 loci and 2 sub-populations:

```
> N <- 10 # sample size
> L <- 5 # number of loci
> K <- 2 # number of populations
```

We generate the genotypes of the subjects by simulating allele frequencies and sub-population identifiers and, based on these numbers, simulate the genotypes for each locus and each subject:

```
> P.true <- matrix(rbeta(K*L,2,2),ncol=L,nrow=K)
> Z.true <- sample(1:K,N,rep=TRUE)
> X1 <- matrix(rbinom(N*L,1,P.true[Z.true,]),nrow=N,ncol=L)
> X2 <- matrix(rbinom(N*L,1,P.true[Z.true,]),nrow=N,ncol=L)
```

Let us examine the results. The actual allele probabilities that were used for the two sub-populations and the actual identifiers are:

```
> P.true
           [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.09332287 0.3782789 0.3382755 0.4889488 0.3234987
[2,] 0.20860665 0.7138192 0.2734588 0.1592900 0.6474052
> Z.true
 [1] 1 1 2 2 1 1 1 2 2 2
```

The resulted in the genotypes:

```
> X1
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    0    1    0    0
 [2,]    0    1    1    1    0
 [3,]    0    1    0    0    0
 [4,]    0    0    0    0    1
 [5,]    0    0    0    0    0
 [6,]    0    0    0    0    1
 [7,]    0    0    0    1    1
 [8,]    1    1    0    1    1
 [9,]    0    1    0    0    0
[10,]    0    1    1    0    1
> X2
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    0    0    0    0
 [2,]    0    1    0    1    0
 [3,]    1    1    0    0    0
 [4,]    1    1    0    0    0
 [5,]    0    0    1    0    1
 [6,]    0    0    1    1    0
 [7,]    0    0    1    1    0
 [8,]    0    1    0    0    1
 [9,]    0    0    0    0    1
[10,]    0    1    0    0    1
```

These genotypes are the information the statistician gets to observe in order to conduct the cluster analysis.

Let us carry out one iteration of the Gibbs sampler. First we need some auxiliary objects and the initial value of Z

```
> log.lik <- matrix(nrow=N,ncol=K)
> I <- diag(K)
> Z0 <- sample(1:K,N,rep=TRUE) # initiating the sampler
> Z0
 [1] 1 2 1 2 1 1 1 1 1 1
```

The first step involves the simulation of the matrix P of allele frequencies in each locus and each population, given the observed genotypes and given the population identifiers of each individual. The prior distribution assumes that all entries are independent of each other and the marginal distribution is beta(1,1) (i.e., uniform). This assumption may be appropriate if all loci are in linkage disequilibrium and the sub-populations are genetically unrelated. In order to obtain the posterior distribution it is further assumed the subjects are independent and that all loci are in Hardy-Weinberg equilibrium. As a result the total count of the designated allele at each locus and for each sub-population has a binomial distribution. Therefore, the posterior distribution of the entries is also beta with the α parameter modified to one plus the total count of the designated allele and the β parameter is modified to one plus the total count of the other allele. The resulting matrix P corresponds to the simulation of independent entries from these modified beta distributions. Let us implement this in R:

```
> A.count <- I[,Z0]%*%(X1+X2)
> a.count <- I[,Z0]%*%(2-X1-X2)
> P <- matrix(rbeta(K*L,1+A.count,1+a.count),ncol=L,nrow=K)
```

Observe that the first line produces a matrix of dimension $K \times L$ with the total counts of the designated allele and the second line produces a similar matrix for the other allele. In the last line $K \cdot L$ independent beta random variables are generated, each with its own α and β values. The simulated values are then place in a matrix of the appropriate dimension.

The second step corresponds to the simulation of population identifiers, given the allele frequencies and given the observed genotypes. Notice that the a-priory distribution of the identifiers assumes independence between subjects and the uniform distribution over the K sub-population. The posterior distribution corresponds also to independent sampling from discrete distributions on the numbers $1, \ldots, K$, but each individual may have its unique distribution, depending on his/her observed genotypes across all loci.

Fix attention to a single individual. Assuming the Hardy-Weinberg equilibrium and linkage disequilibrium, the joint probability of the observed genotypes and belonging to a specific sub-population is a product of the probabilities of the observed alleles across loci, where the probabilities in the product are the probabilities associated with the given sub-population. The resulting vector of length K

is proportional to the posterior distribution of the population identifiers, given the genotypes, for the that subject. Let us implement this step in R:

```
> for(k in 1:K)
+ {
+     P1 <- sweep(X1,2,P[k,],"*")+sweep(1-X1,2,1-P[k,],"*")
+     P2 <- sweep(X2,2,P[k,],"*")+sweep(1-X2,2,1-P[k,],"*")
+     log.lik[,k] <- rowSums(log(P1*P2))
+ }
> s <- rowMeans(log.lik)
> log.lik <- sweep(log.lik,1,s)
> Z1 <- apply(exp(log.lik),1,function(x) sample(1:K,1,rep=TRUE,prob=x))
> Z1
 [1] 1 2 2 2 1 1 2 1 1 2
```

Notice that the entries of the matrices P1 and P2 contain only one of the two probabilities that are in the sum, depending on whether the designated allele is present or not. The entries of the $N \times K$ matrix `log.lik` correspond to the logarithm of the product of probabilities across all loci. After the formation of the `log.lik` matrix and before exponentiating it we center its values in order to enhance numerical stability. Notice that the resulting exponentiated rows of the matrix do not sum to one. However, they are proportional to the required probability vectors. This is just as good, since the function `sample`, which is applied in order to generate for each row a sub-population identifier, can be applied with the argument `prob` set to a non-negative vector that is proportional to the target sampling distribution.

The function `gibbs.sample` wraps the line codes into a single function that carries out an iteration of the Gibbs sampler. The input is the current value of the vectors of identifiers Z, the number of sub-populations and the observed genotypes and the output is that updated vector of identifiers:

```
> gibbs.sample <- function(Z,K,X1,X2)
+ {
+     N <- nrow(X1)
+     L <- ncol(X1)
+     log.lik <- matrix(nrow=N,ncol=K)
+     I <- diag(K)
+     A.count <- I[,Z]%*%(X1+X2)
+     a.count <- I[,Z]%*%(2-X1-X2)
+     P <- matrix(rbeta(K*L,1+A.count,1+a.count),ncol=L,nrow=K)
+     for(k in 1:K)
+     {
+         P1 <- sweep(X1,2,P[k,],"*")+sweep(1-X1,2,1-P[k,],"*")
+         P2 <- sweep(X2,2,P[k,],"*")+sweep(1-X2,2,1-P[k,],"*")
+         log.lik[,k] <- rowSums(log(P1*P2))
+     }
+     s <- rowMeans(log.lik)
+     log.lik <- sweep(log.lik,1,s)
+     Z <- apply(exp(log.lik),1,function(x) sample(1:K,1,rep=TRUE,prob=x))
+     return(Z)
+ }
```

Let us turn back to the clustering problem. Recall that we get as input the genotypes of N subjects over L loci and goal is to cluster these subjects into K clusters. The clustering algorithm applies Bayesian formulation in order to compute subject-specific prior distributions over the K clusters. The individual is assigned to the cluster with highest posterior probability. The Gibbs sampler is used in order to simulate from the posterior distribution. These simulated values, in turn, produce an approximation of the posterior probabilities.

The function `gibbs.cluster` may be applied in order to implement the clustering algorithm. It takes as input a an initial assignment of the subjects to clusters, the number of clusters and the observed genotypes and produces as output an $N \times K$ matrix of the estimated subject specific posterior distribution of cluster identifiers. One may control the number of iterations of the Gibbs sampler and the number of initial iterations that will be ignored in the approximation of the distributions. The defaults for these numbers are set to 100 and 10, respectively.

```
> gibbs.cluster <- function(Z,K,X1,X2,n.iter=100,burn.down=10)
+ {
+    Z.dist <- matrix(0,nrow=N,ncol=K)
+    for(i in 1:burn.down) Z <- gibbs.sample(Z,K,X1,X2)
+    for(i in (burn.down+1):n.iter)
+    {
+        Z <- gibbs.sample(Z,K,X1,X2)
+        for(k in 1:K) Z.dist[,k] <- Z.dist[,k] + (Z==k)
+    }
+    return(Z.dist/(n.iter-burn.down))
+ }
```

Let us demonstrate the application of the function in a simulated example. Assume that a sample of size of 100 is taken from a population that is composed of 5 genetically unrelated sub-populations. These samples are genotyped over 500 loci:

```
> N <- 100 # sample size
> L <- 500 # number of loci
> K <- 5 # number of populations
> P.true <- matrix(rbeta(K*L,1,1),ncol=L,nrow=K)
> Z.true <- sample(1:K,N,rep=TRUE)
> X1 <- matrix(rbinom(N*L,1,P.true[Z.true,]),nrow=N,ncol=L)
> X2 <- matrix(rbinom(N*L,1,P.true[Z.true,]),nrow=N,ncol=L)
```

The applied statistician gets as input the genotypes and applies the clustering algorithm in order to assign each subject into a cluster. The result of the analysis are stores in the vector `Z.est`:

```
> Z0 <- sample(1:K,N,rep=TRUE)
> Z.dist <- gibbs.cluster(Z0,K,X1,X2)
> Z.est <- apply(Z.dist,1,which.max)
```

In order to assess the successfulness of the applied statistician we may compare its assignment to the actual (unobserved) assignment:

```
> table(Z.true,Z.est)
      Z.est
Z.true  1  2  3  4  5
     1  0 33  0  0  0
     2 18  0  0  0  0
     3  0  0  0  0 17
     4  0  0  0 16  0
     5  0  0 16  0  0
```

Notice that in this example the assignment was carried out with no errors. All subjects that belong to the same cluster where assigned by the applied statistician to the same cluster. Indeed, labeling of the clusters, as far as the algorithm goes, is completely random, hence it is not surprising that the original labels and assigned labels do not match.