Chapter 10

Hidden Markov Models

The Markov process forms the basis of many important models in bioinformatics, including Markov Models and Hidden Markov Models (HMMs) for sequence alignments, protein folding, protein-DNA binding, and phylogenetic trees. For sequence alignments, the BLOSUM, PAM, and other alignment matrices are constructed on the basis of a Markov process. With phylogenetic trees, it is possible to estimate evolutionary distances between several sequences and to visualize them with a tree. Lastly, Hidden Markov Models can learn, for example, the conserved amino acid characteristics of different protein families in order to model protein homology and predict protein family membership.

In this chapter, we will learn about the probability transition matrix and the role it plays in a Markov process to construct specific sequences. In addition, various models for building phylogenetic trees are explained in terms of the rate matrix and the probability transition matrix. Lastly, the basic ideas of the Hidden Markov Model are briefly explained and illustrated by examples[1].

10.1 Markov Process

A *stochastic process* is a sequence of events in which the outcome at any stage depends on some probability. A *Markov process* is a type of stochastic process with the following properties:

[1]This chapter is somewhat more technical in its notation (e.g. conditional probability). However, this is necessary to understand Markov processes.

1. The number of possible outcomes or states is finite.
2. The outcome at any stage depends only on the outcome of the previous stage.
3. The probabilities are constant over time.

Property number 2 is often referred to as *memorylessness*. In a Markov process, no memory of previous states is required to predict the next state in the process - only the current state determines the next state. We will introduce Markov processes with a couple of simple examples. A *Markov model* is a statistical model in which the system being modeled is assumed to be a Markov process.

10.2 Random sampling

Probabilistic Markov models used to predict and classify biological sequences make it possible to draw a random sample from a parameterized population. Fundamentally, these models sample from one or more probability distribution. Recall from Chapter 3 that a discrete distribution is a set of discrete values with certain probabilities that add up to one. The following two basic examples illustrate this point:

Example 1: Tossing a coin. When tossed high into the air, a fair coin X comes up Heads and Tails each with probability $1/2$. Thus, we define the probabilities of each *outcome* with $P(X = H) = 0.5$ and $P(X = T) = 0.5$ - where the letters H and T corresponding to the outcomes Heads and Tails, respectively. No matter how many outcomes, with such a random variable there always exists an outcome population as well as a sampling scheme which can properly model all the possible sequences of outcomes - which are called observed *events* (e.g. Press, et al., 1992).

To model the event of repeatedly tossing a coin into the air, we can use the `sample()` command to define the probabilities of each outcome and specify the number of tosses. The `noquote()` function just removes all quotes from the output.

```
> noquote(sample(c("H","T"),30,rep=TRUE,prob=c(0.5,0.5)))
 [1] T H H H H H H H T T H H H H H H T T T T H T H H T T H T H T
```

Thus, the sampled outcomes of Heads and Tails correspond to the event of a Markov process of repeatedly tossing a fair coin. In the above example, the `sample()` function randomly draws thirty times one of the outcomes

`c("H","T")` with replacement (`rep=TRUE`) and equal probabilities (`prob=c(0.5,0.5)`). The *event* that we observed was the set of outcomes `T H H H H H H H T T H H H H H H T T T T H T H H T T H T H T`

Example 2: Generating a sequence of nucleotides. Another example is that of a random variable X which has (letters corresponding to) nucleotides as its outcomes. The outcomes are $X = A$, $X = C$, $X = G$, and $X = T$. For example, let us define these outcomes for a specific family of DNA sequences with probabilities $P(X = A) = 0.1$, $P(X = G) = 0.4$, $P(X = C) = 0.4$, and $P(X = T) = 0.1$. A DNA sequence that is sampled from this discrete probability distribution represents a sequence event that is modeled by the Markov process. In R, let us `sample()` from this Markov process to produce a 30 base pair sequence (event):

```
> noquote(sample(c("A","G","C","T"),30,rep=TRUE,prob=c(0.1,0.4,0.4,0.1)))
 [1] G C C C C C A G A T C G G G G T G G G C G G G G G C C T C C
```

Of course, if you do this again, then the resulting sequence will differ due to the random nature of its generation.

10.3 Probability transition matrix

In order to build a Markov model that produces specific sequences, we will consider a certain type of random variable. In particular, we will consider a sequence $\{X_1, X_2, \cdots\}$ with values from a certain *state space* E. The latter is simply a set containing the possible values or states of a process. If, for instance, $X_n = i$, then the process is in state i at time n. Similarly, the expression $P(X_1 = i)$ denotes the probability that the process is in state i at time point 1. The event that the process changes its state from i to j (transition) between time point one and two corresponds to the event $(X_2 = j|X_1 = i)$, where the bar means "given that". The probability for this event to happen is denoted by $P(X_2 = j|X_1 = i)$. And in general, the probability of the transition from i to j between time point n and $n + 1$ is given by $P(X_{n+1} = j|X_n = i)$. Lastly, these probabilities can be collected in a *probability transition matrix* $\boldsymbol{P}$ with elements

$$p_{ij} = P(X_{n+1} = j|X_n = i).$$

We will assume that the transition probabilities are the same for all time points so that there is no time index needed on the left hand side. Given that the process X_n is in a certain state, the corresponding row of the transition matrix contains the distribution of X_{n+1}, implying that the sum of the probabilities over all possible states equals one. The probability transition matrix contains a (conditional) discrete probability distribution on each of its rows. For a Markov process it holds that the state at time point $n+1$ depends upon the state at time point n, but not on states at earlier time points.

Example 1: The probability transition matrix. Suppose X_n has two states: Y for a pyrimidine and R for a purine. A sequence can now be generated, as follows. If $X_n = Y$, then we throw with a fair die: If the outcome is lower than or equal to 5, then $X_{n+1} = Y$ and, otherwise, $X_{n+1} = R$. If $X_n = R$, then we throw with a fair coin: If the outcome equals Tail, then $X_{n+1} = Y$, and otherwise $X_{n+1} = R$. For this process the two by two probability transition matrix equals

		to	
		Y	R
from	Y	p_{YY}	p_{YR}
	R	p_{RY}	p_{RR}

where p_{RY} is the probability that the process changes from R to Y. This transition matrix can also be written as

$$\boldsymbol{P} = \begin{bmatrix} p_{YY} & p_{YR} \\ p_{RY} & p_{RR} \end{bmatrix} = \begin{bmatrix} P(X_1 = Y|X_0 = Y) & P(X_1 = R|X_0 = Y) \\ P(X_1 = Y|X_0 = R) & P(X_1 = R|X_0 = R) \end{bmatrix} = \begin{bmatrix} \frac{5}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

Any matrix probability transition matrix $\boldsymbol{P}$ can be visualized by a *transition graph*, where the transition probabilities are visualized by an arrow from state i to state j and the value of p_{ij}. For the current example the transition graph is given by Figure 10.1. The values Y and R of the process are written within the circles and the transition probabilities are written near the arrows.

To actually generate a sequence with values equal to Y and R according to the transition matrix, we define a `markov()` function and pass our state space `stateSpace`, probability transition matrix `P`, sample size `n`, and initial State `initState` at parameters:

```
> markov <- function(StateSpace,P,n, pi0=NULL, initState=NULL){
+  seq <- character(n)
+
```

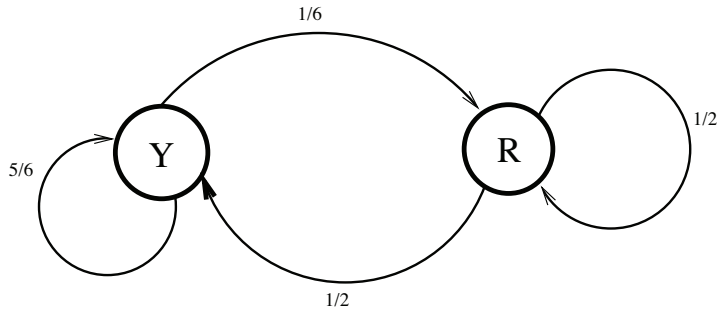

Figure 10.1: Graph of the probability transition matrix

```
+  if (is.null(pi0)) {
+    seq[1] <- initState
+  }
+  else {
+    seq[1] <- sample(StateSpace, 1, replace=TRUE, pi0)
+  }
+
+  for(k in 1:(n-1)) {
+    seq[k+1] <- sample(StateSpace, 1, replace=TRUE, P[seq[k],])
+  }
+
+  return(seq)
+ }
>
>  P <- matrix(c(1/6,5/6,0.5,0.5), 2, 2, byrow=TRUE)
>  rownames(P) <- colnames(P) <- stateSpace <- c("Y","R")
>  P
          Y         R
Y 0.1666667 0.8333333
R 0.5000000 0.5000000
>  noquote(markov(stateSpace,P,30,initState="Y"))
 [1] Y Y R Y R Y R R Y Y R Y R R R Y Y R Y R Y R R R Y R Y R Y R
```

In the function `markov()`, the actual sampling is performed by the `sample()` function. Note that we must pass in either an initial State `initState` or an initial probability distribution `Pi0` for the initial state. Then we sample one element at a time from the set containing Y and R according to the probabilities in row `seq[k]` of the matrix `P`. This makes the probabilities of the states dependent on the corresponding row of the transition matrix. We conveniently use the fact that R adds an element to the sequence; we do not have to declare its length beforehand. In this example, the sequence has a fixed start at State Y and thereafter the first row in the probability transition matrix. Note that without the return command the function does not return

any output.

Example 2: High GC-content. To model a high GC-content DNA sequence, we will define a new probability transition matrix `P` and a new state space consisting of the nucleotides `A`, `G`, `C`, and `T`:

from \ to	A	G	C	T
A	1/6	5/6	0	0
G	1/8	1/2	1/4	1/8
C	0	2/6	3/6	1/6
T	0	1/6	3/6	2/6

In addition, we will use an initial uniform probability distribution `pi0` for the starting state:

```
> P <- matrix(c(
+   1/6,5/6,0,0,
+   1/8,1/2,1/4,1/8,
+   0,2/6,3/6,1/6,
+   0,1/6,3/6,2/6),4,4,byrow=TRUE)
> rownames(P) <- colnames(P) <- StateSpace <- c("A","G","C","T")
> P
          A         G    C         T
A 0.1666667 0.8333333 0.00 0.0000000
G 0.1250000 0.5000000 0.25 0.1250000
C 0.0000000 0.3333333 0.50 0.1666667
T 0.0000000 0.1666667 0.50 0.3333333
>
> pi0 <- c(1/4,1/4,1/4,1/4)
> names(pi0) = StateSpace
> pi0
   A    G    C    T
0.25 0.25 0.25 0.25
>
> seq <- markov(StateSpace,P,1000, pi0=pi0)
> table(seq)
x
  A   C   G   T
 57 373 392 178
```

The `markov()` function starts with sampling just once from the initial uniform distribution `pi0` with equal probabilities to establish the start state. It conveniently uses the column and row names of the probability transition matrix for the sampling. The probabilities to go from `A` or `T` to `C` or `G` are large and as well as that to stay within `C` or `G`. From the frequency table it can be observed that the majority of residues are `C` or `G`.

Example 3: High phenylalanine content. In this example we wish to construct DNA sequences that translate into amino acid sequences with high phenylalanine (F) content. Recall that phenylalanine (F) is coded by the triplet codon `TTT` or `TTC`. Therefore, we will construct a DNA Markov model that emits `T`s with high probability:

from \ to	A	G	C	T
A	0.01	0.01	0.01	0.97
G	0.01	0.01	0.01	0.97
C	0.01	0.01	0.01	0.97
T	0.01	0.28	0.01	0.70

We use the function `getTrans()` of the `seqinr` package to translate the generated nucleotide triplets into amino acids.

```
> P <- matrix(c(.01,.01,.01,.97,
+               .01,.01,.01,.97,
+               .01,.01,.01,.97,
+               .01,.28,.01,0.70),4,4,byrow=T)
> rownames(P) <- colnames(P) <- StateSpace <- c("A","G","C","T")
> P
    A    G    C    T
A 0.01 0.01 0.01 0.97
G 0.01 0.01 0.01 0.97
C 0.01 0.01 0.01 0.97
T 0.01 0.28 0.01 0.70
>
> pi0 <- c(1/4,1/4,1/4,1/4)
> names(pi0) = StateSpace
> pi0
   A    G    C    T
0.25 0.25 0.25 0.25
>
> seq <- markov(StateSpace,P,30000, pi0=pi0)
> table(getTrans(seq))

   *    A    C    D    E    F    G    H    I    L    M    N    R    S    T    V
  27   22 2053   19    1 3835   23    1   61 1593   33    1    1   77    1 2161
   W    Y
  17   74
```

From the table it is clear that the phenylalanine (F) frequency is the largest among all the amino acids in the generated, very long amino acid sequence.

Example 4: Generating the probability transition matrix. To illustrate estimation of the probability transition matrix we proceed with the sequence `seq` produced by the previous example.

```
> nr <- count(tolower(seq),2)
> nr

   aa    ac    ag    at    ca    cc    cg    ct    ga    gc    gg    gt    ta
    3     3     2   283     1     2     2   283    76    60    74  6283   211
   tc    tg    tt
  223  6414 16079
> A <- matrix(NA,4,4)
> A[1,1] <- nr["aa"]; A[1,2]<-nr["ag"]; A[1,3]<-nr["ac"]; A[1,4]<-nr["at"]
> A[2,1] <- nr["ga"]; A[2,2]<-nr["gg"]; A[2,3]<-nr["gc"]; A[2,4]<-nr["gt"]
> A[3,1] <- nr["ca"]; A[3,2]<-nr["cg"]; A[3,3]<-nr["cc"]; A[3,4]<-nr["ct"]
> A[4,1] <- nr["ta"]; A[4,2]<-nr["tg"]; A[4,3]<-nr["tc"]; A[4,4]<-nr["tt"]
> rowsumA <- apply(A, 1, sum)
> Phat <- sweep(A, 1, rowsumA, FUN="/")
> rownames(Phat) <- colnames(Phat) <- c("a","g","c","t")
> round(Phat,3)
      a     g     c     t
a 0.010 0.007 0.010 0.973
g 0.012 0.011 0.009 0.968
c 0.003 0.007 0.007 0.983
t 0.009 0.280 0.010 0.701
```

The number of transitions are counted and divided by the row totals. The estimated transition probabilities are quite close to the true transition probabilities. The zero transition probabilities are exactly equal to the truth because these do not occur. This estimation procedure can also easily be applied to amino acid sequences.

10.4 Properties of the transition matrix

In the previous examples, the sequence was started at a certain state. Often, however, only the probabilities of the initial states are available. That is, we have a vector $\boldsymbol{\pi}_0$ with initial probabilities $\pi_{10} = P(X_0 = 1)$ and $\pi_{20} = P(X_0 = 2)$. Furthermore, if the transition matrix is defined as

$$\boldsymbol{P} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} = \begin{bmatrix} P(X_1 = 1|X_0 = 1) & P(X_1 = 2|X_0 = 1) \\ P(X_1 = 1|X_0 = 2) & P(X_1 = 2|X_0 = 2) \end{bmatrix},$$

then the probability that the process is in State 1 at time point 1 can be written as

$$P(X_1 = 1) = \pi_{10}p_{11} + \pi_{20}p_{21} = \boldsymbol{\pi}_0^T \boldsymbol{p}_1, \tag{10.1}$$

where $\boldsymbol{p}_1$ is the first column of $\boldsymbol{P}$, see Section 10.8. Note that the last equality holds by definition of matrix multiplication. In a similar manner, it can be shown that $P(X_1 = 2) = \boldsymbol{\pi}_0^T \boldsymbol{p}_2$, where $\boldsymbol{p}_2$ is column 2 of the

transition matrix $\boldsymbol{P} = (\boldsymbol{p}_1, \boldsymbol{p}_2)$. It can be concluded that $\boldsymbol{\pi}_0^T \boldsymbol{P} = \boldsymbol{\pi}_1^T$, where $\boldsymbol{\pi}_1^T = (P(X_1 = 1), P(X_1 = 2))$; the probability at time point 1 that the process is in State 1, State 2, respectively. This holds in general for all time points n, that is

$$\boldsymbol{\pi}_n^T \boldsymbol{P} = \boldsymbol{\pi}_{n+1}^T. \tag{10.2}$$

Thus to obtain the probabilities of the states at time point $n+1$, we can simply use matrix multiplication[2].

Example 1: Matrix multiplication to compute probabilities. Suppose the following initial distribution and probability matrix

$$\boldsymbol{\pi}_0 = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \end{bmatrix}, \boldsymbol{P} = \begin{bmatrix} \frac{5}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix},$$

for State 1 and 2, respectively. Then $P(X_1 = 1)$ and $P(X_1 = 2)$ collected in $\boldsymbol{\pi}_1^T = (P(X_1 = 1), P(X_1 = 2))$ can be computed as follows.

$$\boldsymbol{\pi}_1^T = \boldsymbol{\pi}_0^T \boldsymbol{P} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} \frac{5}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \cdot \frac{5}{6} + \frac{1}{3} \cdot \frac{1}{2} & \frac{2}{3} \cdot \frac{1}{6} + \frac{1}{3} \cdot \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{13}{18} & \frac{5}{18} \end{bmatrix}$$

Using R its operator **%*%** for matrix multiplication, the product $\boldsymbol{\pi}_0^T \boldsymbol{P}$ can be computed as follows.

```
> P <- matrix(c(5/6,1/6,0.5,0.5),2,2,byrow=T)
> pi0 <- c(2/3,1/3)
> pi0 %*% P
          [,1]      [,2]
[1,] 0.7222222 0.2777778
```

Yet, another important property of the probability transition matrix deals with the probability of being in state 1 given that the process is in state 1 two time points before. In particular, it holds (see Section 10.8) that

$$P(X_2 = 1 | X_0 = 1) = p_{11}^2, \tag{10.3}$$

where the latter is element $(1, 1)$ of the matrix[3] $\boldsymbol{P}^2$. In general, we have that

$$P(X_n = j | X_0 = i) = p_{ij}^n,$$

[2]The transposition sign ^{T} simply transforms a column into a row.

[3]For a brief definition of matrix multiplication, see Pevsner (2003, p.56) or wikipedia using the search string "wiki matrix multiplication".

which is element i, j of $\boldsymbol{P}^n$.

Example 3: More matrix multiplication. Given the probability matrix of the previous example, the values $P(X_2 = j|X_0 = i)$ for all of i, j can be computed by matrix multiplication.

$$\boldsymbol{P}^2 = \begin{bmatrix} \frac{5}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{5}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} (\frac{5}{6})^2 + \frac{1}{6}\frac{1}{2} & \frac{5}{6}\frac{1}{6} + \frac{1}{6}\frac{1}{2} \\ \frac{1}{2}\frac{5}{6} + (\frac{1}{2})^2 & \frac{1}{2}\frac{1}{6} + (\frac{1}{2})^2 \end{bmatrix} = \begin{bmatrix} \frac{28}{36} & \frac{8}{36} \\ \frac{24}{36} & \frac{12}{36} \end{bmatrix}.$$

Obviously, such matrix multiplications can be accomplished much more convenient on a personal computer.

```
> P %*% P
          [,1]      [,2]
[1,] 0.7777778 0.2222222
[2,] 0.6666667 0.3333333
```

Larger powers of $\boldsymbol{P}$ can be computed more efficiently by methods given below.

10.5 Stationary distribution

A probability distribution $\boldsymbol{\pi}$ satisfying

$$\boldsymbol{\pi}^T = \boldsymbol{\pi}^T \boldsymbol{P}$$

is *stationary* because the transition matrix does not change the probabilities of the states of the process. Such a distribution usually exists, is unique, and plays an essential role in the long term behavior of the process. It sheds light on the question: What is the probability $P(X_n = 1|X_0 = 1) = p_{11}^n$, as n increases without bound. That is: What is the probability that the process is in State 1, given that it started in State 1, as time increases without bound? To answer such a question we need large powers of the probability transition matrix. To compute these we need the eigen-decomposition of the probability transition matrix

$$\boldsymbol{P} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1},$$

where $\boldsymbol{V}$ is the eigenvector matrix and $\boldsymbol{\Lambda}$ the diagonal matrix with eigenvalues. The latter are usually sorted in decreasing order so that the first

(left upper) is the largest. Now the third power of the probability transition matrix can be computed, as follows

$$\boldsymbol{P}^3 = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{\Lambda}\boldsymbol{\Lambda}\boldsymbol{V}^{-1} = \boldsymbol{V}\boldsymbol{\Lambda}^3\boldsymbol{V}^{-1}.$$

So that, indeed, in general

$$\boldsymbol{P}^n = \boldsymbol{V}\boldsymbol{\Lambda}^n\boldsymbol{V}^{-1}.$$

The latter is a computationally convenient expression because we only have to take the power of the eigenvalues in $\boldsymbol{\Lambda}$ and to multiply by the left and right eigenvector matrices. We say that an n x n probability transition matrix $\boldsymbol{P}$ is diagonalizable when $\boldsymbol{P}$ has an eigen-decomposition - which is if and only if $\boldsymbol{P}$ has n linearly independent eigenvectors.

In the long term the Markov process tends to a certain value (Brémaud, 1999, p.197) because a probability transition matrix has a unique largest eigenvalue equal to 1 with corresponding eigenvectors $\mathbf{1}$ and $\boldsymbol{\pi}$ (or rather normalized versions of these). It follows that, as n increases without bound, then $\boldsymbol{P}^n$ tends to $\mathbf{1}\boldsymbol{\pi}^T$. In other words, $P(X_n = j|X_0 = i) = p_{ij}^n$ tends to element (i, j) of $\mathbf{1}\boldsymbol{\pi}^T$, which is equal to element j of $\boldsymbol{\pi}$. For any initial distribution $\boldsymbol{\pi}_0$, it follows that $\boldsymbol{\pi}_0'\boldsymbol{P}^n$ tends to $\boldsymbol{\pi}^T$.

Example 1: The stationary distribution. To compute the eigen-decomposition of the probability transition matrix $\boldsymbol{P}$ as well as powers of it, we may use the function `eigen()`.

```
> P <- matrix(c(1/6,5/6,0.5,0.5),2,2,byrow=T)
> V <- eigen(P,symmetric = FALSE)
> V$values
[1]  1.0000000 -0.3333333
> V$vectors
           [,1]       [,2]
[1,] -0.7071068 -0.8574929
[2,] -0.7071068  0.5144958
```

The output of the function `eigen()` is assigned to the list `V` from which the eigenvalues and eigenvectors can be extracted and printed to the screen. Now we can compute $\boldsymbol{P}^{16}$; the probability transition matrix raised to the power sixteen.

```
> V$vec %*% diag(V$va)^(16) %*% solve(V$vec)
     [,1]  [,2]
[1,] 0.375 0.625
[2,] 0.375 0.625
```

So that the stationary distribution $\boldsymbol{\pi}^T$ equals $(0.375, 0.625)$.

Example 2: Diploid genomes. Suppose A is a dominant gene, a a recessive and that we start with a heterozygote aA. From the latter we obtain the initial state probability $\boldsymbol{\pi}^T = (0, 1, 0)$ for the events (AA, aA, aa). When we consider pure self-fertilization, then the offspring from AA is AA with probability $(1, 0, 0)$, that of aa is aa with probability $(0, 0, 1)$, and that of aA is (AA, aA, aa) with probability $1/4, 1/2, 1/4$, respectively. Hence, the probability transition matrix becomes

$$\boldsymbol{P} = \text{from} \begin{array}{c} \\ AA \\ aA \\ aa \end{array} \begin{array}{c} \text{to} \\ \begin{array}{ccc} AA & aA & aa \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1/4 & 1/2 & 1/4 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

We can now compute the transition probability matrix after 5, 10, and 15 generations.

```
> P <- matrix(c(1,0,0, 1/4,1/2,1/4,0,0,1),3,3,byrow=TRUE) # Not an ergodic transition
    ↪ matrix!
> rownames(P) <- colnames(P) <- c("AA", "aA", "aa")
> P
     AA  aA   aa
AA 1.00 0.0 0.00
aA 0.25 0.5 0.25
aa 0.00 0.0 1.00
> V <- eigen(P,symmetric = FALSE)
> V$va
[1] 1.0 1.0 0.5
> V$vec
          [,1]      [,2] [,3]
[1,] 0.8944272 0.0000000    0
[2,] 0.4472136 0.4472136    1
[3,] 0.0000000 0.8944272    0
> V$vec %*% diag(V$va)^(5) %*% solve(V$vec)
         [,1]    [,2]     [,3]
[1,] 1.000000 0.00000 0.000000
[2,] 0.484375 0.03125 0.484375
[3,] 0.000000 0.00000 1.000000
> V$vec %*% diag(V$va)^(10) %*% solve(V$vec)
          [,1]         [,2]      [,3]
[1,] 1.0000000 0.0000000000 0.0000000
[2,] 0.4995117 0.0009765625 0.4995117
[3,] 0.0000000 0.0000000000 1.0000000
> V$vec %*% diag(V$va)^(15) %*% solve(V$vec)
          [,1]         [,2]      [,3]
[1,] 1.0000000 0.000000e+00 0.0000000
[2,] 0.4999847 3.051758e-05 0.4999847
[3,] 0.0000000 0.000000e+00 1.0000000
```

With an initial population of aA, the distribution of AA, aA, and aa combinations after 5, 10, and 15 generation can be read from the second row. We can see that as the generations increase the population is becoming more and more homozygotic. More precisely, using Equation 10.2 it can be shown that

$$\boldsymbol{\pi}_{n+1}^T = \left[\frac{1}{2} - \left(\frac{1}{2}\right)^{n+1}, \left(\frac{1}{2}\right)^n, \frac{1}{2} - \left(\frac{1}{2}\right)^{n+1}\right],$$

so that the distribution converges to $[1/2, 0, 1/2]$.

Note that this method of raising the transition probability matrix to a large power can easily be applied to determine the stationary distribution. A transition matrix is called `ergodic` if each state can reach every other state in one step. When a transition matrix is ergodic then there is only one unique stationary distribution and all the rows in the $\boldsymbol{P}^N$ matrix will converge to this one stationary distribution. However, if the transition matrix is not ergodic then there exist multiple stationary distributions that are dependent upon the initial population. Since our transition matrix $\boldsymbol{P}$ is not ergodic, the rows of $\boldsymbol{P}^N$ converge to three different stationary distributions that are dependent upon whether the initial population is AA, aA, or aa. Notice that since the transition matrix in Example 1 is ergodic, the two rows of $\boldsymbol{P}^N$ converge to the same unique stationary distribution.

Taking a transition matrix to a certain power is also used to construct the PAM substitution matrices from the PAM1 substitution matrix. For example, the PAM250 matrix is the PAM1 matrix to the 250th power (Pevsner, 2003, p.53). The theory behind powers of transition matrices is also used in the construction of various BLOSUM matrices (Pevsner, 2003, p.50-59; Deonier, et al. 2005, 187-190).

10.6 Phylogenetic distance

Phylogenetic trees are constructed on the basis of distances between DNA sequences. These distances are computed from substitution models which are defined by a matrix representing the rate of substitutions of one state to the other. The latter is usually expressed as a matrix $\boldsymbol{Q}$. The rates of staying in a state are given by a negative number on the diagonal of the substitution

matrix. The probability transition matrix $\boldsymbol{P}$ can be computed by matrix exponentiation $\boldsymbol{P} = \exp(\boldsymbol{Q})$. How to do this in practice will be illustrated by an example.

Example 1: From a rate matrix to a probability transition matrix. Suppose the rate matrix

$$\boldsymbol{Q} = \text{from} \begin{array}{c} \\ A \\ G \\ C \\ T \end{array} \begin{array}{c} \begin{array}{cccc} & & \textit{to} & \\ A & G & C & T \end{array} \\ \begin{bmatrix} -0.60 & 0.20 & 0.20 & 0.20 \\ 0.20 & -0.60 & 0.20 & 0.20 \\ 0.20 & 0.20 & -0.60 & 0.20 \\ 0.20 & 0.20 & 0.20 & -0.60 \end{bmatrix} \end{array}.$$

Thus, within a certain time period a proportion of 0.20 A mutates into G, 0.20 A into C, and 0.20 A into T. Consequently, a proportion of 0.60 of the residues goes back to A. Given this rate matrix, we can find the probability transition matrix $\boldsymbol{P} = \exp(\boldsymbol{Q})$ by using the function `expm(Q)` from the package `Matrix`.

```
> library(Matrix)
> Q <- 0.2 * Matrix(c(-3,1,1,1,1,-3,1,1,1,1,-3,1,1,1,1,-3),4)
> rownames(Q) <- colnames(Q) <- c("A","G","C","T")
> P <- as.matrix(expm(Q))
> round(P,2)
     A    G    C    T
A 0.59 0.14 0.14 0.14
G 0.14 0.59 0.14 0.14
C 0.14 0.14 0.59 0.14
T 0.14 0.14 0.14 0.59
```

Thus the probability that the state mutates from A to A is 0.59, from A to G is 0.14, etc.

Because all phylogenetic models are defined in terms of rate matrices, we shall concentrate on these. For instance, the rate matrix for the Jukes and Cantor (1969) (JC69) model can be written as

$$\boldsymbol{Q}_{JC69} = \begin{array}{c} \\ A \\ G \\ C \\ T \end{array} \begin{array}{c} \begin{array}{cccc} A & G & C & T \end{array} \\ \begin{bmatrix} \cdot & \alpha & \alpha & \alpha \\ \alpha & \cdot & \alpha & \alpha \\ \alpha & \alpha & \cdot & \alpha \\ \alpha & \alpha & \alpha & \cdot \end{bmatrix} \end{array}.$$

The sum of each row of a rate matrix equals zero, so that from this requirement the diagonal elements of the JC69 model are equal to -3α. Furthermore, the non-diagonal substitution rates of the JC69 model all have the same value α. That is, the mutate from i to j equals that from j to i, so that the rate matrix is symmetric. Also the probability that the sequence equals one of the nucleotides is $1/4$. This assumption, however, is unrealistic is many cases.

Transitions are substitutions of nucleotides within types of nucleotides, thus purine to purine or pyrmidine to pyrmidine ($A \leftrightarrow G$ or $C \leftrightarrow T$). Transversions are substitutions between nucleotide type ($A \leftrightarrow T$, $G \leftrightarrow T$,$A \leftrightarrow C$, and $C \leftrightarrow G$). In the JC69 model a transition is assumed to happen with equal probability as a transversion. That is, it does not account for the fact that transitions are more common that transversions. To cover this for more general type of models are proposed by Kimura (1980, 1981), which are commonly abbreviated by K80 and K81. In terms of the rate matrix these models can we written as

$$\boldsymbol{Q}_{K80} = \begin{bmatrix} \cdot & \alpha & \beta & \beta \\ \alpha & \cdot & \beta & \beta \\ \beta & \beta & \cdot & \alpha \\ \beta & \beta & \alpha & \cdot \end{bmatrix}, \qquad \boldsymbol{Q}_{K81} = \begin{bmatrix} \cdot & \alpha & \beta & \gamma \\ \alpha & \cdot & \gamma & \beta \\ \beta & \gamma & \cdot & \alpha \\ \gamma & \beta & \alpha & \cdot \end{bmatrix}$$

In the K80 model a change within type (transition) occurs at rate α and between type (transversion) at rate β. In the K81 model all changes occur at a different though symmetric rate; the rate of change $A \rightarrow G$ is α and equals that of $A \leftarrow G$. If α is large, then the amount of transitions is large; if both β and γ are very small, then the number of transversions is small.

A model is called "nested" if it is a special case of a more general model. For instance, the K80 model is nested in the K81 model because when we take $\gamma = \beta$, then we obtain the K80 model. Similarly, the JC69 model is nested in the K80 model because if we take $\beta = \alpha$, then we obtain the JC69 model.

Some examples of models with even more parameters are the Hasegawa, Kishino, and Yano (1985) (HKY85) model and the General Time-Reversable Model (GTR) model

$$\boldsymbol{Q}_{HKY85} = \begin{bmatrix} \cdot & \alpha\pi_G & \beta\pi_C & \beta\pi_T \\ \alpha\pi_A & \cdot & \beta\pi_C & \beta\pi_T \\ \beta\pi_A & \beta\pi_G & \cdot & \alpha\pi_T \\ \beta\pi_A & \beta\pi_G & \alpha\pi_C & \cdot \end{bmatrix}, \quad \boldsymbol{Q}_{GTR} = \begin{bmatrix} \cdot & \alpha\pi_G & \beta\pi_C & \gamma\pi_T \\ \alpha\pi_A & \cdot & \delta\pi_C & \epsilon\pi_T \\ \beta\pi_A & \delta\pi_G & \cdot & \zeta\pi_T \\ \gamma\pi_A & \epsilon\pi_G & \zeta\pi_C & \cdot \end{bmatrix}$$

The distance between DNA sequences is defined on the basis of these models. From these distances the phylogenetic tree is computed by a neighbor-joining algorithm such that it has the smallest total branch length.

Example 2: The K81 model. To compute the rate matrix of the K81 model with $\alpha = 3/6$, $\beta = 2/6$, $\gamma = 1/6$ we may use the following.

```
> alpha <- 3/6; beta <- 2/6; gamma<- 1/6; Q <- matrix(data=NA,4,4)
> Q[1,2] <- Q[2,1] <- Q[3,4] <- Q[4,3] <- alpha
> Q[1,3] <- Q[3,1] <- Q[2,4] <- Q[4,2] <- beta
> Q[1,4] <- Q[4,1] <- Q[2,3] <- Q[3,2] <- gamma
> diag(Q) <- -(alpha + beta + gamma)
> Q
           [,1]       [,2]       [,3]       [,4]
[1,] -1.0000000  0.5000000  0.3333333  0.1666667
[2,]  0.5000000 -1.0000000  0.1666667  0.3333333
[3,]  0.3333333  0.1666667 -1.0000000  0.5000000
[4,]  0.1666667  0.3333333  0.5000000 -1.0000000
> Q <- Matrix(Q)
> P <- as.matrix(expm(Q))
> P
          [,1]      [,2]      [,3]      [,4]
[1,] 0.4550880 0.2288517 0.1767105 0.1393498
[2,] 0.2288517 0.4550880 0.1393498 0.1767105
[3,] 0.1767105 0.1393498 0.4550880 0.2288517
[4,] 0.1393498 0.1767105 0.2288517 0.4550880
```

By raising the power of the probability transition matrix to a sufficiently large number, it can be observed that the stationary distribution $\boldsymbol{\pi}^T = (0.25, 0.25, 0.25, 0.25)$.

Example 3: Stationarity distribution for the JC69 model. Let's take $\alpha = 1/5$ as in Example 1 and compute the rate matrix $\boldsymbol{Q}$ of the JC69 model, the corresponding probability transition matrix $\boldsymbol{P}$, and raise it to the power 50.

```
> library(Matrix)
> alpha <- 1/5;  Q <- matrix(rep(alpha,16),4,4)
> diag(Q) <- -3 * alpha
> Q <- Matrix(Q)
> P <- as.matrix(expm(Q))
> V <- eigen(P,symmetric = FALSE)
> V$vec %*% diag(V$va)^(50) %*% solve(V$vec)
     [,1] [,2] [,3] [,4]
[1,] 0.25 0.25 0.25 0.25
[2,] 0.25 0.25 0.25 0.25
[3,] 0.25 0.25 0.25 0.25
[4,] 0.25 0.25 0.25 0.25
```

Hence, the stationary distribution is $\boldsymbol{\pi}^T = (0.25, 0.25, 0.25, 0.25)$ (cf. Ewens & Grant, 2005, p. 477).

Example 4: JC69 distance between two sequences. In case of the JC69 model, the distance between sequences is a function of the proportion of different nucleotides. Namely,

$$d = -\frac{3}{4}\log(1 - 4p/3),$$

where p is the proportion of different nucleotides of the two sequences. The pairwise distances between DNA sequences can be computed by the function `dist.dna()` from the `ape` package.

```
> library(ape);library(seqinr)
> accnr <- paste("AJ5345",26:27,sep="")
> seqbin <- read.GenBank(accnr, species.names = TRUE, as.character = FALSE)
> dist.dna(seqbin, model = "JC69")
          AJ534526
AJ534527 0.1326839
```

Hence, the distance is 0.133. Over a total of 1143 nucleotides there are 139 differences, som that the proportion of different nucleotides $139/1143 = p$. Inserting this into the previous distance formula gives the distance. This can be verified as follows.

```
> seq <- read.GenBank(accnr, species.names = TRUE, as.character = TRUE)
> p <- sum(seq$AJ534526==seq$AJ534527)/1143
> d <- -log(1-4*p/3)*3/4
> d
[1] 0.1326839
```

Example 5: Phylogenetic tree of Cytb orthologs. To further illustrate distances between DNA sequences we shall download the Chamaea fasciata mitochondrial Cytb gene for cytochrome b for 10 species of warblers of the genus sylvia (Paradis, 2006). The function `paste()` is used to quickly define the accession numbers and `read.GenBank()` to actually download the sequences. The species names are extracted and attached to the sequences. We shall use the `dist.dna()` function with the `K80` model.

```
> library(ape);library(seqinr)
> accnr <- paste("AJ5345",26:35,sep="")
> seq <- read.GenBank(accnr)
> names(seq) <- attr(seq, "species")
> dist <- dist.dna(seq, model = "K80")
> plot(nj(dist))
```

Obviously, in this manner various trees can be computed and their plots compared.

When various different models are defined the question becomes apparent which of these fits best to the data relative to the number of parameters (symbols) of the model. When the models are estimated by maximum likelihood, then the Akaike information criterion (AIC = -2 · loglik + 2 · number of free parameters) can be used to select models. The best model is the one with the smallest AIC value.

Example 6: AIC calculations. The AIC of the JC69 and the TN93 models are computed.

```
> library(ape);library(seqinr);library(phangorn)
> phyDatseq <- as.phyDat(seq) # transform data into phyDat
> treeUPGMAJC69 <- unroot(upgma(dist.dna(seq, model = "JC69")))
> fit <- pml(treeUPGMAJC69,data=phyDatseq)
> fitUPGMAJC69 <- optim.pml(fit,TRUE)  #branch length optimization (works only for unrooted
    ↪  trees)
> AIC(fitUPGMAJC69)
[1] 9281.952
> treeUPGMATN93 <- unroot(upgma(dist.dna(seq, model = "TN93")))
> fit <- pml(treeUPGMATN93,data=phyDatseq)
> fitUPGMATN93 <- optim.pml(fit,TRUE)  #branch length optimization (works only for unrooted
    ↪  trees)
> AIC(fitUPGMATN93)
[1] 9281.952
```

The model have equal AIC so these are equally well.

By updating a simple model to a more complex one may yield a better fitting. Here we update a JC69 model to a GTR model and use the AIC or a bootstrap application to test (Shimodaira and Hasegawa, 1999) the models against one another.

```
> treeNJ <- NJ(dist.dna(seq, model = "JC69"))
> fit <- pml(treeNJ, data=phyDatseq)
> fitGTR <- update(fit,k=4,inv=0.2)
> fitGTR <- optim.pml(fitGTR, TRUE,TRUE,TRUE,TRUE,TRUE)
> AIC(fitGTR)
[1] 8207.742
> SH.test(fitGTR,fitUPGMAJC69,B=10000)
     Trees      ln L Diff ln L p-value
[1,]     1 -4074.871    0.0000  0.4988
[2,]     2 -4623.976  549.1047  0.0000
```

Note that the updated model has a much smaller AIC. The loglikelihood for the JC69 model is -4623.976 (df=17) and that of GTR is -4074.871 (df=29). The simpler model is rejected in favor of the more complex.

Example 7: Tree comparison. We may compare a neighbor joining (NJ) tree with an "unweighted pair group method with arithmetic mean" (UPGMA) tree (also known as an average linkage tree). A criterion for

choosing between these two is a parsimoniousness score. The score can be optimized by rearranging the trees.

```
> treeUPGMAGG95 <- unroot(upgma(dist.dna(seq, model = "GG95"))) #plot(treeNJ)
> treeUPGMAGG95 <- optim.parsimony(treeUPGMAGG95,phyDatseq)
> parsimony(treeUPGMAGG95,phyDatseq)
> treeNJGG95 <- NJ(dist.dna(seq, model = "GG95"))
> treeNJGG95 <- optim.parsimony(treeNJGG95,phyDatseq)
> parsimony(treeNJGG95,phyDatseq)
```

Both trees have score 570 and are equally parsimonious.

Example 8: Using PHYML. A program called PHYML (Guindon & Gascuel, 2003) can be downloaded from `http://atgc.lirmm.fr/phyml/` and run by the R function `phymltest()`, if the executable is available at the same directory. We first write the sequences to the appropriate directory. The output from the program is written to the object called `out` for which the functions `plot(out)` and `summary(out)` can be used to extract more detailed information.

```
> setwd("/share/home/wim/bin")
> write.dna(seq,"seq.txt", format ="interleaved")
> out <-phymltest("seq.txt",format = "interleaved", execname ="phyml_3.0.1_linux32")
> print(out)
          nb.free.para    loglik      AIC
JC69                 1 -4605.966 9213.931
JC69+I               2 -4425.602 8855.203
JC69+G               2 -4421.304 8846.608
JC69+I+G             3 -4421.000 8848.001
K80                  2 -4423.727 8851.455
K80+I                3 -4230.539 8467.079
K80+G                3 -4224.457 8454.915
K80+I+G              4 -4223.136 8454.272
F81                  4 -4514.331 9036.662
F81+I                5 -4309.600 8629.199
F81+G                5 -4304.530 8619.060
F81+I+G              6 -4303.760 8619.519
F84                  5 -4351.164 8712.328
F84+I                6 -4112.006 8236.012
F84+G                6 -4106.568 8225.135
F84+I+G              7 -4105.500 8225.001
HKY85                5 -4333.086 8676.171
HKY85+I              6 -4102.262 8216.524
HKY85+G              6 -4097.401 8206.802
HKY85+I+G            7 -4096.624 8207.248
TN93                 6 -4323.291 8658.581
TN93+I               7 -4097.099 8208.198
TN93+G               7 -4091.461 8196.922
TN93+I+G             8 -4090.790 8197.580
GTR                  9 -4293.398 8604.795
GTR+I               10 -4084.522 8189.043
GTR+G               10 -4079.010 8178.020
GTR+I+G             11 -4078.149 8178.299
```

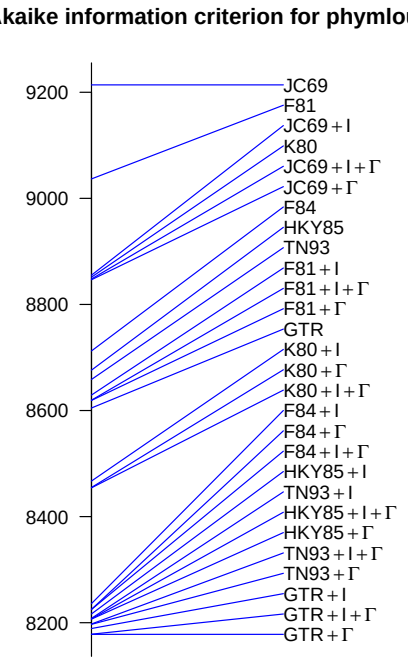

Figure 10.2: Evaluation of models by AIC .

The notation "+I" and "+G" indicates the presence of invariant sites and/or a gamma distribution of substitution rates. It can be seen that the smallest AIC corresponds to model 27 called `GTR+G`. To plot it, we have to read the trees, and, next, to extract the 27th, see Figure 10.3.

```
> tr <- read.tree("seq.txt_phyml_tree.txt")
> plot(tr[[27]])
> add.scale.bar(length=0.01)
```

In case similar sequences have slightly different lengths, these have to be aligned by programs such as `clustalx` or `clustalw` before these can be used.

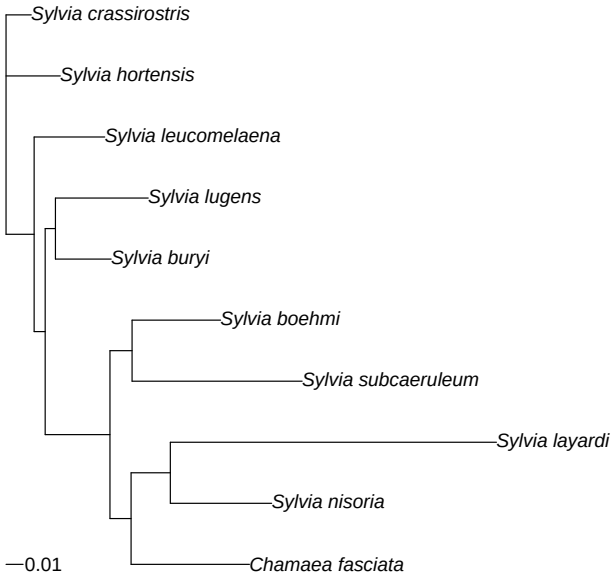

Figure 10.3: Tree according to GTR model.

10.7 Hidden Markov Models

A *Hidden Markov model (HMM)* is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved, hidden states. Accordingly, in an HMM there are two probability transition matrices. There is an *emission matrix* $\boldsymbol{E}$ with probabilities of observed, emitted symbols and a *transition matrix* $\boldsymbol{A}$ with transition probabilities between the unobserved, hidden states. The generation of an observable sequence requires two separate steps. First, there is a transition via a Markov process of a hidden state and then given the new hidden state value there is an emission of an observable outcome. We shall illustrate this by the classical example of the occasionally dishonest casino (Durbin et. al., 1998, p.18). The number

of hidden states and how they are connected (via transitions) defines the *topology* of the HMM. For example, even though the dishonest casino and GC-isochore HMMs, that are defined below, model very different problems, they in fact share the same topology.

Example 1: Occasionally dishonest casino. A casino uses a fair die most of the time. However, occasionally it switches to a loaded, unfair die. The die state with respect to fairness is hidden for the observer. The observer can only observe the values of the die and not its hidden state with respect to its fairness. It is convenient to denote fair by F and loaded by L. The transition probabilities matrix A of the hidden states is defined as:

$$\boldsymbol{A} = \begin{bmatrix} P(D_i = F|D_{i-1} = F) & P(D_i = L|D_{i-1} = F) \\ P(D_i = F|D_{i-1} = L) & P(D_i = L|D_{i-1} = L) \end{bmatrix} = \begin{bmatrix} 0.95 & 0.05 \\ 0.10 & 0.90 \end{bmatrix}$$

Thus, the probability is 0.95 that the die is fair at time point i, given that it is fair at time point $i-1$. The probability that it will switch from fair to loaded is 0.05. The probability that it will switch from loaded to fair is 0.10 and that it stays loaded is 0.90. With this emission matrix we can generate a sequence of hidden states, where the values F and L indicate whether the die is fair or loaded, respectively. Given the fairness of the die, we define the probability emission matrix E:

$$\begin{aligned} \boldsymbol{E} &= \begin{bmatrix} P(O_i = 1|D_i = F) & P(O_i = 2|D_i = F) & P(O_i = 3|D_i = F) & \cdots \\ P(O_i = 1|D_i = L) & P(O_i = 2|D_i = L) & P(O_i = 3|D_i = L) & \cdots \end{bmatrix} \\ &= \begin{bmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/10 & 1/10 & 1/10 & 1/10 & 1/10 & 1/2 \end{bmatrix}. \end{aligned} \tag{10.4}$$

Thus given that the die is fair, the probability of any outcome equals $1/6$. However, when the die is loaded, the probability of outcome 6 equals $1/2$ and that of any other outcome equals $1/10$.

To create the occasionally dishonest casino HMM we must define the transition and emission matrices according to the probabilities above. In the `hmm()` function below, we first sample the hidden states from the Markov chain to establish the die type to toss, then the outcome of the die toss is sampled according to the die type (hidden state).

```
> hmm <- function(A,E,n,initState){
+     observationSet <- colnames(E)
+     hiddenSet <- rownames(E)
```

```
+     x <- sample(observationSet,1,replace=T,E[initState,])
+     h <- markov(hiddenSet,A,n,initState=initState)
+     for(k in 1:(n-1)) {x[k+1] <- sample(observationSet,1,replace=T,E[h[k],])}
+     out <- cbind(x,h)
+     return(out)
+ }
>
> E <- matrix(c(rep(1/6,6),rep(1/10,5),1/2),2,6,byrow=T) #emission matrix
> A <- matrix(c(0.95,0.05,0.1,0.9),2,2,byrow=TRUE)       #transition matrix
>
> colnames(E) <- EmissionSpace <- c("1","2","3","4","5","6")
> rownames(E) <- StateSpace <- c("F","L")
> E
          1         2         3         4         5         6
F 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
L 0.1000000 0.1000000 0.1000000 0.1000000 0.1000000 0.5000000
>
> rownames(A) <- colnames(A) <- StateSpace
> A
     F    L
F 0.95 0.05
L 0.10 0.90
>
> statePath <- hmm(A,E,100,c("F"))
> dimnames(statePath) <- list(rep("",100),c("observation","hidden state"))
> noquote(t(statePath))

observation  3 1 3 5 3 5 3 6 4 6 3 6 2 1 1 2 6 1 4 5 4 3 3 1 2 4 6 5 6 2 1 5 2
hidden state F F F F F F F F F F F F F F F F F F F F F F F F L L F L L F F

observation  1 5 6 6 3 3 5 2 2 2 5 1 3 5 4 3 1 5 6 1 2 6 6 6 2 1 4 6 3 5 2 4 3
hidden state F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F

observation  1 5 1 6 6 3 6 6 3 6 6 3 3 3 6 1 6 2 6 6 6 6 4 2 5 2 3 4 1 4 5 6 4
hidden state F F L L L L L L L L L L L L L L L L L F F F F F F F F F L L

observation  6
hidden state F
```

In certain applications in bioinformatics, it is of most importance to estimate the values of the hidden states given observable data. First, the *Baum-Welch* algorithm is used to estimate the transition and emission matrices from the observable data. The Baum-Welch algorithm is a greedy *Expectation Maximization (EM)* method that finds the *Maximum Likelihood Estimate (MLE)* of the model parameters. Remember from Chapter 4 that the MLE is the set of model parameters that maximizes the likelihood of observing the data, which in this case is the observed sequence emitted by our HMM. After inferring the MLE of the model parameters, the *Viterbi* algorithm is often used to discover the most probable state path (over the hidden states) that is best supported by the model parameters and training sequence. Like the Needleman-Wunsch and Smith-Waterman algorithms in

Chapter 9, the Baum-Welch and Viterbi algorithms also use *dynamic programming* methods in their implementations.

Our Viterbi algorithm below builds up a matrix $v(i, l)$, where i runs from one to the number of observations and l from one to the number of states. The initial values are $v(1, 1) = F$, and $v(1, l) = 0$ for all l. Then the values for $v(i, l)$ are recursively defined by

$$v(i, l) = e(l, x(i)) \cdot \max_k \{v(i - 1, k)a(k, l)\}$$

For each row of the matrix the maximum is taken as the best predictor of the hidden state.

Example 2: Predict die type with the Viterbi algorithm. In this example, we will skip the Baum-Welch EM step to find the maximum likelihood estimate of the transition matrices. Instead, we will just use the same transition matrices that were used to generate the die tossing sequence. Next, to predict the hidden states with the transition matrices and emitted sequence, we program the `viterbi()` function with the arbitrary precision `mpfrArray()` function from the `Rmpfr` package in order to avoid double-float underflow errors. Then we will use the `viterbi()` function to find the hidden states of the state path that generated the observed, emitted sequence above.

```
> viterbi <- function(A,E,seq,pi0) {
+     # v <- matrix(NA, nr=length(seq), nc=dim(A)[2])
+     v <- mpfrArray(NA, dim=c(length(seq),dim(A)[2]), precBits = 100)
+     colnames(v) = rownames(E)
+     v[1,] <- pi0
+     for(i in 2:length(seq)) {
+         for (l in 1:dim(A)[1]) {v[i,l] <- E[l,seq[i]] * max(v[(i-1),] * A[,l])} }
+     return(v)
+ }
>
> vit <- viterbi(A,E,statePath[,1],c(1,0))
> vitRowMax <- unlist(apply(vit, 1, function(x) names(x)[which.max(x)]))
>
> statePath <- cbind(statePath,vitRowMax)
> dimnames(statePath) <- list(rep("",100),c("observation","hidden state","predicted state")
    ↪ )
> noquote(t(statePath))

observation     3 1 3 5 3 5 3 6 4 6 3 6 2 1 1 2 6 1 4 5 4 3 3 1 2 4 6 5 6 2 1 5
hidden state    F F F F F F F F F F F F F F F F F F F F F F F F L L F L L F
predicted state F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F

observation     2 1 5 6 6 3 3 5 2 2 2 5 1 3 5 4 3 1 5 6 1 2 6 6 6 2 1 4 6 3 5 2
hidden state    F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
predicted state F F F F F F F F F F F F F F F F F F F F F F L F F F F F F F
```

```
observation     4 3 1 5 1 6 6 3 6 6 3 6 6 3 3 3 6 1 6 2 6 6 6 6 4 2 5 2 3 4 1 4
hidden state    F F F F L L L L L L L L L L L L L L L L L L L F F F F F F F F F
predicted state F F F F F F F F F L L L L L L L L L L L L L L L L F F F F F

observation     5 6 4 6
hidden state    F L L F
predicted state F F F F
>
> 1 - (sum(statePath[,2] == statePath[,3]) / nrow(statePath))
[1] 0.16
```

The misclassification rate is 0.16 - which is quite large given the fact that we used the true transition and emission matrix. An important observation is that after a transition to a new hidden state, it often takes a few values for the prediction to change. This is caused by the recursive nature of the algorithm.

Example 3: GC-isochores. The nucleotide content of the chromosomes of warm-blooded vertebrates is not uniform. Instead, there exist large ($>$ 200KB), alternating GC-rich and GC-poor regions along the individual chromosomes. These large GC-rich and GC-poor regions are often called GC-isochores and AT-isochores, respectively. In addition, the telomeres at the ends of the chromosomes are very often GC-isochores - which helps to keep the ends of the double-stranded DNA from unraveling. We will construct an HMM to first produce alternating isochores along a chromosome and then to predict their placements. Note that we will intentionally shrink the scale in order to save computation time. First, we will define the transition probabilities matrix A of the hidden states as:

$$\boldsymbol{A} = \begin{bmatrix} P(D_i = GC|D_{i-1} = GC) & P(D_i = AT|D_{i-1} = GC) \\ P(D_i = GC|D_{i-1} = AT) & P(D_i = AT|D_{i-1} = AT) \end{bmatrix} = \begin{bmatrix} 0.990 & 0.010 \\ 0.005 & 0.995 \end{bmatrix}$$

Then we will define the probability emission matrix E as:

$$\begin{aligned} \boldsymbol{E} &= \begin{bmatrix} P(A|D_i = GC) & P(G|D_i = GC) & P(C|D_i = GC) & P(T|D_i = GC) \\ P(A|D_i = AT) & P(G|D_i = AT) & P(C|D_i = AT) & P(T|D_i = AT) \end{bmatrix} \\ &= \begin{bmatrix} 0.15 & 0.35 & 0.35 & 0.15 \\ 0.35 & 0.15 & 0.15 & 0.35 \end{bmatrix}. \end{aligned} \tag{10.5}$$

where GC and AT refer to GC-isochores and AT-isochores, respectively. We will now define the A and E matrices in R and then use our `hmm()` function to generate alternating isochores according to our transition and emission probabilities:

```
> E <- matrix(c(0.15,0.35,0.35,0.15,
+               0.35,0.15,0.15,0.35),2,4,byrow=T) #emission matrix
> A <- matrix(c(0.99,0.01,0.005,0.995),2,2,byrow=TRUE)       #transition matrix
>
> colnames(E) <- EmissionSpace <- c("A","G","C","T")
> rownames(E) <- StateSpace <- c("GC","AT")
> E
      A    G    C    T
GC 0.15 0.35 0.35 0.15
AT 0.35 0.15 0.15 0.35
> rownames(A) <- colnames(A) <- StateSpace
> A
      GC    AT
GC 0.990 0.010
AT 0.005 0.995
>
> statePath <- hmm(A,E,1000,c("AT"))
> dimnames(statePath) <- list(rep("",1000),c("observation","hidden state"))
>
> statePathFactor = factor(statePath[,2])
> plot(x=1:length(statePathFactor),    # 1 to n
+      y=as.numeric(statePathFactor),  # 1 or 2 based on factor
+      type = "l",                     # line plot
+      yaxt = "n",                     # do not plot y axis
+      xlab = "Chromosome Location",   # x-axis label
+      ylab = "GC vs AT Isochores",    # y-axis label
+      col = "magenta",                # color
+      lwd = 2,                        # line width = 2
+      cex.lab=1.5                     # make axis labels big
+      )
> axis(2, at=c(1,2), labels=c("AT","GC"), cex.axis=1.5) # plot y-axis with special labels
```

The resulting plot of the generated isochores can be seen in Figure 10.4. Due to the transition probabilities, we see that the AT-isochores are generally longer than the GC-isochores.

Example 4: Predict isochores with the Viterbi algorithm. Now we will use our `viterbi()` function to try to predict the placement of the isochores from the emitted sequence that was generated by our HMM:

```
> vit <- viterbi(A,E,statePath[,1],c(0,1))
> vitRowMax <- unlist(apply(vit, 1, function(x) names(x)[which.max(x)]))
> vitRowMaxFactor = factor(vitRowMax)
> plot(x=1:length(vitRowMaxFactor),    # 1 to n
+      y=as.numeric(vitRowMaxFactor),  # 1 or 2 based on factor
+      type = "l",                     # line plot
+      yaxt = "n",                     # do not plot y-axis
+      xlab = "Chromosome Location",   # x-axis label
+      ylab = "GC vs AT Isochores",    # y-axis label
+      col = "magenta",                # line color
+      lwd = 2,                        # line width
+      cex.lab=1.5                     # make axis labels big
+      )
> axis(2, at=c(1,2), labels=c("AT","GC"), cex.axis=1.5) # plot y-axis with special labels
>
> statePath <- cbind(statePath,vitRowMax)
```

```
> dimnames(statePath) <- list(rep("",1000),c("observation","hidden state","predicted state"
    ↪ ))
>
> 1 - (sum(statePath[,2] == statePath[,3]) / nrow(statePath))
[1] 0.035
```

The misclassification rate is only 0.035 and the resulting plot of the recovered isochores can be seen in Figure 10.5. Although not perfect, the `viterbi()` function is able to properly recover the locations of many of the isochores.

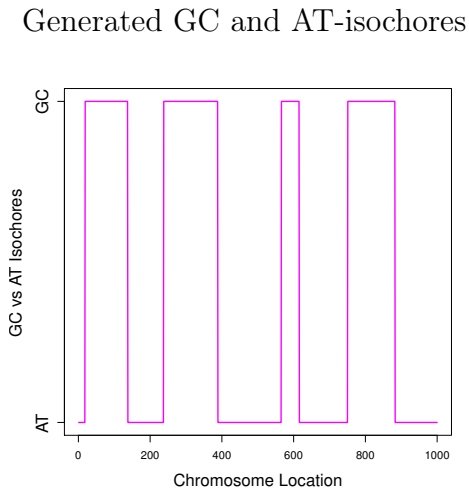

Figure 10.4: Plot of chromosomal GC and AT-isochores sampled from an HMM.

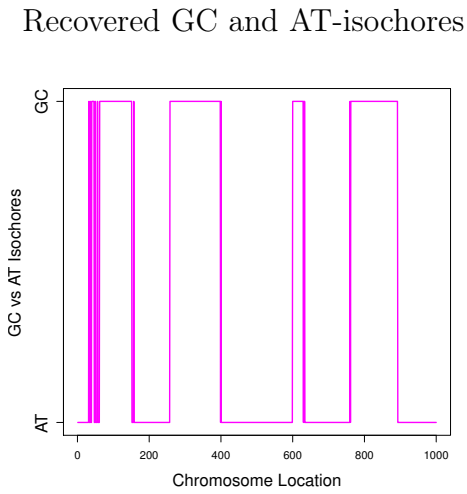

Figure 10.5: Plot of recovered chromosomal GC and AT-isochores using the sampled chromosomal sequence with the Viterbi method.

10.8 Appendix

The probability that the process is in State 1 at time point 1 can be computed as follows.

$$\begin{aligned} P(X_1 = 1) &= P(X_1 = 1, X_0 = 1) + P(X_1 = 1, X_0 = 2) \\ &= P(X_1 = 1 | X_0 = 1) \cdot P(X_0 = 1) + P(X_1 = 1 | X_0 = 2) \cdot P(X_0 = 2) \\ &= \pi_{10} p_{11} + \pi_{20} p_{21} \\ &= \boldsymbol{\pi}_0^T \boldsymbol{p}_1, \end{aligned}$$

where $\boldsymbol{p}_1$ is the first column of $\boldsymbol{P}$.
In particular, it holds that

$$\begin{aligned}
P(X_2 = 1|X_0 = 1) &= P(X_2 = 1, X_1 = 1|X_0 = 1) + P(X_2 = 1, X_1 = 2|X_0 = 1) \\
&= \sum_{k=1}^{2} P(X_2 = 1, X_1 = k|X_0 = 1) \\
&= \sum_{k=1}^{2} P(X_2 = 1|X_1 = k, X_0 = 1) \cdot P(X_1 = k|X_0 = 1) \\
&= \sum_{k=1}^{2} P(X_2 = 1|X_1 = k) \cdot P(X_1 = k|X_0 = 1) \\
&= p_{11}p_{11} + p_{21}p_{12} \\
&= \text{row 1 of } \boldsymbol{P} \text{ times column 1 of } \boldsymbol{P} = \boldsymbol{P}^2_{11},
\end{aligned}$$

where the latter is element $(1, 1)$ of the matrix $\boldsymbol{P}^2 = \boldsymbol{P} \cdot \boldsymbol{P}$.

10.9 Overview and concluding remarks

The probability transition matrix is extensively explained and illustrated because it is a cornerstone to many ideas in bioinformatics. A thorough treatment of phylogenetics is given by Paradis (2006) and of Hidden Markov Models by Durbin et. al (2005).

10.10 Exercises

1. **Transition graph.** Visualize by a transition graph the following transition matrices. For the process with four states take the names of the nucleotides in the order A, G, T, and C.

$$\begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} \frac{1}{4} & \frac{2}{4} & 0 & \frac{1}{4} \\ \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \\ 0 & \frac{2}{7} & \frac{5}{7} & 0 \\ \frac{1}{8} & \frac{1}{8} & \frac{2}{8} & \frac{4}{8} \end{bmatrix}, \begin{bmatrix} \frac{1}{4} & \frac{3}{4} & 0 & 0 \\ \frac{1}{6} & \frac{5}{6} & 0 & 0 \\ 0 & 0 & \frac{5}{7} & \frac{2}{7} \\ 0 & 0 & \frac{3}{8} & \frac{5}{8} \end{bmatrix}.$$

2. **Computing probabilities.** Given the states 0 and 1 and the following initial distribution and probability matrix

$$\boldsymbol{\pi}_0 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix}, \boldsymbol{P} = \begin{bmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

   (a) Compute $P(X_1 = 0)$.

   (b) Compute $P(X_1 = 1)$.

   (c) Compute $P(X_2 = 0|X_0 = 0)$.

   (d) Compute $P(X_2 = 1|X_0 = 0)$.

3. **Programming GTR.** Use $\pi_A = 0.15$, $\pi_G = 0.35$, $\pi_C = 0.35$, $\pi_T = 0.15$, $\alpha = 4$, $\beta = 0.5$, $\gamma = 0.4$, $\delta = 0.3$, $\epsilon = 0.2$, and $\zeta = 4$.

   (a) Program the rate matrix in such a manner that it is simple to adapt for other values of the parameters.

   (b) Is the transversion rate larger or smaller then the transition rate?

   (c) Compute the corresponding the probability transition matrix.

   (d) Try to argue whether you expect a large frequency of transversions or translations.

   (e) Generate a sequence of 99 nucleotide residues according to the markov model.

4. **Distance according to JC69.**

   (a) Down load the sequences `AJ534526` and `AJ534527`. Hint: Use `as.character = TRUE` in the `read.GenBank()` function.

   (b) Compute the proportion of different nucleotides.

   (c) Use this proportion to verify the distances between these sequences according to the JC69 model.