Chapter 6

Data augmentation

One of the great strengths of Bayesian methods is the way all unknowns are represented as random variables. The cancer status of an asymptomatic woman attending screening is unknown, and hence random. The mean income of all expatriates in Singapore is unknown, and hence random. The secret weapon of Bayesian statistics is that if there is something that you would like to know, and if you knew it, it would make the likelihood function for the data you do know easy to calculate, you can introduce it into the analysis by treating it as a random variable. This trick is also called extending the conversation to include the additional variable. In notation, if you wish to calculate $p(D|\theta)$ and cannot, but you can calculate both $p(D|\theta, A)$ and $p(A|\theta)$, where $A$ is the additional “data” you would need to calculate the likelihood (possibly high dimensional), then you can recast the problem as finding

$$p(D|\theta) = \int_A p(D|\theta, A)p(A|\theta)\,dA.$$  

This turns the problem of calculating an intractable likelihood into solving an integral. Well, this is something you are going to be doing anyway if you are taking a Bayesian approach, because MCMC and similar routines are, essentially, tools to solve difficult integrals, although the integration problem is dressed up as a sampling one.

Data augmentation involves treating the unknown “data” as additional parameters to be estimated, i.e. involves switching from sampling $\theta$ to sampling $(\theta, A)$. You can then perform MCMC as usual, but if the interest is just on $\theta$, you can ignore the samples from $A$—integrating them out.

There are several technical problems you may encounter when doing data augmentation. If the augmented data $A$ are high dimensional, then you may have to propose values to a considerably larger number of parameters than you would had you known the information. This may increase the computational burden. It also makes it harder to assess convergence, since there may be too many additional, unwanted parameters to justify storing in memory. Another issue is that the augmented data are often correlated quite closely with each other and/or one or more parameters, so it can be difficult to design good proposal distributions.
Table 6.1: Data on citrus canker in Miami, Florida, from Cook et al (2008). The total number of hosts in this geographic area of Miami was 6056 and there were no diseased trees at the start of data collection.

<table>
<thead>
<tr>
<th>Day</th>
<th>30</th>
<th>60</th>
<th>90</th>
<th>120</th>
<th>150</th>
<th>180</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trees</td>
<td>4</td>
<td>14</td>
<td>23</td>
<td>40</td>
<td>71</td>
<td>122</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day</th>
<th>210</th>
<th>240</th>
<th>270</th>
<th>300</th>
<th>330</th>
<th>360</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trees</td>
<td>264</td>
<td>480</td>
<td>535</td>
<td>603</td>
<td>872</td>
<td>1124</td>
</tr>
</tbody>
</table>

Citrus canker

We illustrate data augmentation with an application to modelling citrus canker in Florida, a disease of citrus trees that leaves lesions on the fruit making them impossible to sell—except as juice—and which is therefore economically devastating to citrus growing areas. In the late 1990s, citrus canker was found to be spreading in Florida again, after having previously been eradicated, and a chief source of inoculum seemed to be trees in the gardens of private dwellings, which could not be controlled as effectively as on commercial orchards. Gottwald and colleagues collected data on several thousand such ‘backyard’ citrus trees, which were later analysed by Cook et al (2008). The cumulative number of diseased trees at 30d intervals are presented in table 6.1. Diseased trees were not removed (as this was a control population meant to allow the quantification of disease spread in an undisturbed population).

In the tutorial, I proposed using deterministic exponential or logistic growth models for the data, in which case an appropriate choice of error—representing biological variability, not observation error—must be selected. Note that if these really are variability and not measurement errors, then the model should be restarted from each observation to each next observation, rather than simply being run once from an initial condition at time 0. A more satisfying alternative is to fit a stochastic model, in which the variability is implicit to the model description. Here we shall focus on a stochastic SI model, which is one way to formulate a logistic growth for an infectious disease.

If there are currently \( n(t) \) trees infected out of \( N = 6056 \), then the SI model says that:

\[
p \{ n(t + h) = n(t) + 1 \mid n(t) \}, \beta = \beta' (N - n(t)) h + o(h),
\]

that is, \( \beta' (N - n(t)) \) is the instantaneous rate of infection. Since this rate does not vary between infection times, we can simulate “data” from this model for a given \( n(0) \) and \( \beta' \) by generating a series of Exp(1) variates and scaling them by a simple function of infections so far. It is convenient to rescale the parameter by the population size \( b = \beta N \).

We are unable to do straightforward MCMC for this problem because:

- the disease status of two trees in the population are not independent—knowing one is infected makes it more likely the other is infected, as the former may have infected the latter;
• the infection time of each tree is interval censored—we know the 30d window in which it occurred, ignoring the delay between infection and onset, or assuming it to be unknown but constant.

The combination of the two of these makes it impossible to derive the likelihood directly\(^1\). But, if we happened to know the exact times of infection, we could calculate the likelihood exactly. If the time of infection of the \(i\)th tree is \(t_i\), the likelihood under this hypothetical scenario would be

\[
p(t_1, t_2, t_3, \ldots, t_{1124}|b) \\
= p(t_2, t_3, \ldots, t_{1124}|t_1, b)p(t_1|b) \\
= p(t_3, \ldots, t_{1124}|t_1, b)p(t_2|t_1, b)p(t_1|b) \\
= \prod_{i=1}^{1124} p(t_i|t_{i-1}, \ldots, t_1, b) \\
= \prod_{i=1}^{1124} p(t_i|t_{i-1}, b)
\]

where, without loss of generality, we have ordered the individual indices by the infection order (so \(t_i < t_j\) for \(i < j\)), and we set \(t_0 = 0\) for the first new infection. (Actually, an additional term is required to account for no further infections between \(t_{1124}\) and 360. This takes the form of a survival function; see R code that follows for details.) Note, I will change the model slightly by starting with 0 infections and assuming the first infection has distribution \(U(0, 30)\) as an “act of god,” with subsequent infections following the stochastic SI model. The rationale for that is to make the approach easier to follow—it is not necessary for the approach used below.

Each term in this product is fairly easy to calculate. If the infection rate at time \(t\) is \(\lambda(t)\), the likelihood contribution from \(i(>1)\) is:

\[
p(t_i|t_{i-1}, b) = \lambda(t_i) \exp \left\{ -\int_{t_{i-1}}^{t_i} \lambda(t) \, dt \right\}
\]

(take a course in survival analysis or stochastic processes to see why). Since the infection rate does not change in between infection events, according to this model, the integral becomes the area of a rectangle with height \(b(i-1)(N-i+1)\) for the \(i\)th infection and length \(t_i - t_{i-1}\) and so the likelihood contribution simplifies to

\[
p(t_i|t_{i-1}, b) = b(i-1)(N-i+1) \exp \left\{ -b(i-1)(N-i+1)(t_i - t_{i-1}) \right\}
\]

So, if we take a continuous uniform prior on \((0, 100)\) for \(b\), the posterior given

\(^1\)This is not strictly true: the likelihood could be derived if you could work out the matrix exponential of a matrix with dimensions roughly 1000×1000. As far as I’m aware, current state of the art software to calculate matrix exponentials is limited to matrices 1% of that size.
the actual data $D$ is:

$$p(b|D) \propto p(D|b)p(b)$$

$$= \int \mathcal{T} p(D|t,b)p(t|b) \, dt$$

$$= \int \mathcal{T} p(t|b) \, dt$$

where $t$ is the unobserved vector of infection times and $\mathcal{T}$ the space of times that are consistent with the data. We can then run an MCMC sampler on the times and parameter $b$, as long as the initial set of times fall in $\mathcal{T}$ and we never propose values outside $\mathcal{T}$. This can be done in R as follows.

The following function calculates the log-posterior for a parameter value and combination of infection times:

```r
logposterior=function(pars,d=data)
{
  ninf = length(pars$t)
  no_s = d$N:1
  no_i = 0:(d$N-1)
  rates = (pars$b/d$N)*no_i*no_s
  deltat = pars$t[-1]-pars$t[-ninf]
  lastt = 360-pars$t[ninf]
  loglikelihood = sum(dexp(deltat,rates[2:ninf],log=TRUE)) +
                  pexp(lastt,rates[ninf+1])
  logprior = dunif(pars$b,0,100,log=TRUE)
  pars$logposterior = loglikelihood+logprior
  pars
}
```

Infection rates are calculated for all possible infections, but only (i) those rates corresponding to actual infections (via the `dexp` term) or (ii) the rate for the first infection that did not occur (the `pexp` term) enter the likelihood.

The Metropolis-Hastings step is standard, and the only thing to note is how the consistency of the event times with the data is checked (lines 1 and 2).

```r
mh=function(current,old,data)
{
  reject=FALSE
  if(current$b<0)reject=TRUE
  if(current$b>100)reject=TRUE
  if(min(current$t-data$tmin)<0)reject=TRUE #1
  if(min(data$tmax-current$t)<0)reject=TRUE #2
  if(!reject)
  {
    current=logposterior(current)
    logaccprob=current$logposterior-old$logposterior
    lu=-rexp(1)
    if(lu>logaccprob)reject=TRUE
  }
}
```

Infection rates are calculated for all possible infections, but only (i) those rates corresponding to actual infections (via the `dexp` term) or (ii) the rate for the first infection that did not occur (the `pexp` term) enter the likelihood.

The Metropolis-Hastings step is standard, and the only thing to note is how the consistency of the event times with the data is checked (lines 1 and 2).
The data are set up as follows:

```r
data = list(t=seq(30,360,30),n=c(4,14,23,40,71,122,264,480,
535,603,872,1124),N=6056)
data$tmin = rep(0,data$n[1])
for(k in 2:12)
  data$tmin=c(data$tmin,rep(data$t[k-1],data$n[k]-data$n[k-1]))
data$tmax = data$tmin+30
```

The sampler itself is also rather standard. Note that to ameliorate the computational burden, I prefer not propose changes to each infection time at each iteration, and when I propose a change, I propose it uniformly between the last and next infection times. Traceplots are provided in figure 6.1.

```r
current=list(b=0.004,t=sort(runif(length(data$tmin),data$tmin,
data$tmax)))
current=logposterior(current)
MCMCiterations=10000
store=data.frame(b=rep(0,MCMCiterations),
t100=rep(0,MCMCiterations))
for(iteration in 1:MCMCiterations)
{
  if(iteration%%100==0)print(iteration)
  old=current
  current$b=rnorm(1,current$b,0.0005)
  current=mh(current,old,data)
  for(k in sample(1:length(data$tmin),100))
  {
    old=current
    mint=c(0,current$t[-length(current$t)])
    maxt=c(current$t[-1],360)
    current$t[k]=runif(1,mint[k],maxt[k])
    current=mh(current,old,data)
  }
  store[iteration,1]=current$b
  store[iteration,2]=current$t[100]
}
```

Although the traceplot for the unknown of interest, $b$, appears to have converged, the mixing of the one time point we choose to monitor is poor. It is not clear, then, whether the sample of $b$ really represents the marginal posterior. We might therefore try to improve the proposal distribution for the infection time. Instead of a proposal to change time $t_i$ that is $U(t_{i-1}, t_{i+1})$, we might propose from $N(t_i, \sigma)$ and then relabel the times to ensure $t_i < t_j$ for all $i < j$. With the following change
CHAPTER 6. DATA AUGMENTATION

Figure 6.1: Traceplots of infection parameter (left) and an arbitrary infection time (right) using data augmentation. Proposals used are uniform for infection times between last and next infection times.

```r
for(k in sample(1:length(data$tmin),100))
{
  old=current
  current$t[k]=rnorm(1,current$t[k],5)
  mint=c(0,current$t[-length(current$t)])
  maxt=c(current$t[-1],360)
  current$t=sort(current$t)
  current=mh(current,old,data)
}
```

mixing is much improved (see figure 6.2) and with more iterations we would be satisfied. This code could then act as the basis for an improved model of how infection rates change with time.

Data augmentation is particularly valuable when dealing with discrete space, continuous time stochastic processes, as the likelihood often becomes tractable once you condition on unobserved event times and natures. However, for large numbers of events, simulation based methods such as Approximate Bayesian Computation may be more feasible. Data augmentation is also valuable for discrete time stochastic processes, such as Hidden Markov Models, as in the following section.

6.1 Hidden Markov models

A Markov model is a model for sequentially linked data (for instance, linked over time), in which the past is irrelevant conditional on the present for predicting the future. Autoregressive models of order 1 are Markov, as are many models found in stochastic processes, such as the epidemic model considered in the last section. While Markov models can be very flexible tools for such data, sometimes we need
to use a more sophisticated representation of the underlying phenomenon. For instance, if the parameters governing the Markov model change over time—or whatever the linked-dimension of the data is—this should be built into the model, either via a deterministic relationship between time and the transition probabilities or other parameters, or by having a random component, such as switches between two different parameter regimes.

Hidden Markov models incorporate an unobserved state, which might, in a model of economic data, be bear and bull markets, which potentially changes from time point to time point, and which controls the parameter values for the data model. Examples include:

- for stock prices, having two underlying states allowing high and low fluctuation regimes;
- for epidemic disease outbreaks, having three underlying states allowing suitable, unsuitable and neutral conditions for propagation;
- for cancer detection, the underlying presence or absence, and stage if present, of an undiagnosed cancer.

As with the last section, if you knew the hidden state, the likelihood is usually tractable, and so the model can be fitted by extending the conversation to include the unobserved hidden state. This is perhaps best illustrated with an example.

**Intron 7 of chimpanzee α-fetoprotein gene V1**

This example is based on a cool paper by Boys et al (2000) in which they show how to use hidden Markov models to detect homogeneous segments in DNA
sequences, with a case study on intron 7 of the chimpanzee (*Pan troglodytes*) α-fetoprotein gene. This sequence is fairly short, with just 1968 base pairs or nucleotide, presented in table 6.2. As you will know, each base pair can be represented by one of the letters A, C, G and T.

A moving average plot (figure 6.3) indicates regions of heterogeneous nucleotide frequencies, and a cusum plot (figure 6.4) suggests three possible regions, near the beginning, middle and end of this sequence, with the latter the most pronounced.

We will fit a series of models of increasing complexity to these data until we arrive at the model considered by Boys et al. In all we will define \( Y_i \in \{1, 2, 3, 4\} \) to be the nucleotide at location \( i \) (with 1 representing A, 2, C, 3, G and 4, T).
The first model assumes independence between neighbouring nucleotides:

\[
Y_i = \begin{cases} 
1 & \text{with probability } p_1, \\
2 & \text{with probability } p_2, \\
3 & \text{with probability } p_3, \\
4 & \text{with probability } p_4.
\end{cases}
\]

Clearly, \( p_1 + p_2 + p_3 + p_4 = 1 \). A suitable prior distribution for these probabilities must therefore ensure they sum to unity. A popular prior for such constrained probabilities is the Dirichlet, which is a multidimensional generalisation of the beta. This has one parameter per probability (so, four in the
current example), with density, for our four probability model,  

$$P \left( \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \right) = \frac{\prod_{i=1}^{4} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{4} \alpha_i)} \prod_{i=1}^{4} p_i^{\alpha_i-1}.$$  

If $\alpha_i = 1$ for all $i$, then the Dirichlet is uniform over the simplex. Scatter plots of example three dimensional Dirichlet densities are plotted in figure 6.5, from which the flexibility of the distribution can be seen. For this model, we should have plenty of information from the data, and so a non-informative prior may be appropriate. The Dirichlet with hyperparameters $(1, 1, 1, 1)$ is uniform over permissible values of $p$ and so we shall use that.

Since the Dirichlet is conjugate to the multinomial (it is, trust me), and we’re assuming independence between locations and so the total number of each letter is a sufficient statistic and is multinomial, we don’t really need to do MCMC for this problem and could instead generate samples directly from the (Dirichlet) posterior. However, we’ll set it up as an MCMC sampler which we can adapt for the more complex models that follow. In all the models we consider in this section, it will be possible to sample directly from the distribution of each parameter, or subsets of parameters, conditional on the other parameters and data, i.e. we can use Gibbs sampling. This simplifies matters a great deal as it means we need not calculate the log posterior at any point, and accept all proposals automatically. This will be beneficial since the number of hidden states is large (around 2000) and the likelihood calculations will be onerous enough to want to avoid.

For this model, then, we shall propose from the Dirichlet distribution and accept all proposals. We will take a Dirichlet$(1, 1, 1, 1)$ prior, i.e. uniform on the simplex. The code to do this is:
Figure 6.5: **Dirichlet distributions.** In all cases, \( p = (p_1, p_2, p_3) \) is Dirichlet(\( \alpha_1, \alpha_2, \alpha_3 \)). Top left: \( (\alpha_1, \alpha_2, \alpha_3) = (1, 1, 1) \). Top right: \( (\alpha_1, \alpha_2, \alpha_3) = (10, 10, 10) \). Bottom left: \( (\alpha_1, \alpha_2, \alpha_3) = (1, 10, 10) \). Bottom right: \( (\alpha_1, \alpha_2, \alpha_3) = (0.1, 0.1, 0.1) \).
source("data.txt") # read in x=c(A,G,T,C,A,etc)
xn=rep(0,length(x))
for(i in 1:length(x))
{
  if(x[i]==A)xn[i]=1
  if(x[i]==G)xn[i]=2
  if(x[i]==C)xn[i]=3
  if(x[i]==T)xn[i]=4
}
data=list(acgt=xn)
current=list(p=rep(0.25,4))
MCMCits=10000
storage=list(p=array(0,c(MCMCits,4)))
for(iteration in 1:MCMCits)
{
  if(iteration%%1000==0)print(iteration)
  A=sum(data$acgt==1)
  C=sum(data$acgt==2)
  G=sum(data$acgt==3)
  T=sum(data$acgt==4)
  current$p=rdirichlet(1,c(1+A,1+C,1+G,1+T))
  storage$p[iteration,]=current$p
}

Since this is, in actuality, a Monte Carlo sampler, we need not assess convergence. The estimated posterior distribution is presented marginally in figure 6.6.

![Figure 6.6: Model 1: no autoregressive structure](image)

Figure 6.6: **Model 1: no autoregressive structure.** Green: A; orange: C; red: G and blue: T. Note that I have cheated by offsetting the orange curve by +0.5%, and the red by −0.5%, to make them visually distinct.
6.1. HIDDEN MARKOV MODELS

Intron 7 of chimpanzee α-fetoprotein gene V2

In the second version of the analysis, we will use a Markov chain representation of the data, i.e. the probability of the \((i+1)\)th datum given the \(i\)th is taken to be \(p(Y_{i+1}|Y_i) = p_{Y_i,Y_{i+1}}\). There are thus 16 parameters to estimate, of which four groups of four must each sum to one, i.e. \(\sum_{k=1}^{4} p_{j,k} = 1\) for all \(j\). Again, therefore, we can exploit conjugacy by taking the prior to be the product of four independent Dirichlets, one for each starting nucleotide. The posterior is thus the product of four Dirichlets, the parameters of which can be derived by counting the number of each type of transition. We therefore set up a function to do this first:

```r
def howmanyACGTs(start, d=data) {
    right_start = (1:(d$n-1))[d$acgt[2:d$n]==start]
    if(start==0) right_start = (1:(d$n-1))
    A = sum(data$acgt[right_start]==1)
    C = sum(data$acgt[right_start]==2)
    G = sum(data$acgt[right_start]==3)
    T = sum(data$acgt[right_start]==4)
    return(c(A,C,G,T))
}
```

The rest of the code is a minor adaption of what came before.

```r
current=list(p=array(rep(0.25,16),c(4,4)))
data=list(acgt=xn,n=length(xn))
MCMCits=10000
storage=list(p=array(0,c(MCMCits,4,4)))
for(iteration in 1:MCMCits) {
    if(iteration%%1000==0) print(iteration)
    for(j in 1:4) {
        ACGT = howmanyACGTs(j)+rep(1,4)
        current$p[,j] = rdirichlet(1,ACGT)
    }
    storage$p[iteration,,] = current$p
}
```

Again, we need not assess convergence. The posteriors for each of the parameters are plotted in figure 6.7.

Intron 7 of chimpanzee α-fetoprotein gene V3

In this third draft of the analysis, we incorporate the hidden Markov model, though we do not, as yet, allow changes to the hidden states (henceforth, segments), and keep the initial conditions throughout the run. The initial conditions were selected to be roughly consistent with the findings of Boys et al.
CHAPTER 6. DATA AUGMENTATION

Figure 6.7: Model 2: simple Markov model. Transition probabilities, by starting nucleotide. Green: A as ending nucleotide; orange: C; red: G and blue: T.

For this version, we will allow two hidden segments, but I would like the flexibility to rerun it with more, so the number of hidden segments will be a variable in what follows. We will let $\lambda_{i,j}$ be the probability of moving from segment $i$ to segment $j$ on neighbouring points on the sequence. Again, we get a multinomial distribution for the total number of each transition type, and so can use a Dirichlet for the prior to get a Dirichlet posterior. We will therefore need to be able to count the number of switches from one segment to another, and the following function does this:

```r
howmanysegmentswitches=function(startsegment,d=data,pars=current)
{
  right_start=which(pars$segment[1:(d$n-1)]==startsegment)
  counts=rep(0,pars$nsegments)
  for(k in 1:pars$nsegments)
```
counts[k] = sum(pars$segment[right_start+1] == k)
return(counts)
}

We also have to amend the previous function so that it only counts nucleotides when in a specified segment.

howmanyACGTs = function(start, segment, d = data)
{
  right_start = (1:(d$n-1))[(d$acgt[2:d$n] == start) &
  (current$segment[1:(d$n-1)] == segment)]
  A = sum(data$acgt[right_start] == 1)
  C = sum(data$acgt[right_start] == 2)
  G = sum(data$acgt[right_start] == 3)
  T = sum(data$acgt[right_start] == 4)
  return(c(A, C, G, T))
}

We’ll start by setting all segments to be 1, except the first 100 and a stretch of length 150 near the end, which will be segment 2. We will propose changes to both $p$ and $\lambda$, but one set at a time.

nsegments = 2
current = list(nsegments = nsegments,
  p = array(rep(0.25, 16), c(4, 4, nsegments)),
  segment = rep(1, data$n),
  lambda = array(rep(1/nsegments, nsegments^2),
    c(nsegments, nsegments))
)

current$segment[1:100] = rep(2, 100)
current$segment[1751:1900] = rep(2, 150)
MCMCits = 10000
storage = list(p = array(0, c(MCMCits, 4, 4, nsegments)),
  lambda = array(0, c(MCMCits, nsegments, nsegments)))
for(iteration in 1:MCMCits)
{
  if(iteration%%1000 == 0) print(iteration)
  for(start in 1:4)
  {
    for(segment in 1:current$nsegments)
    {
      ACGT = howmanyACGTs(start, segment) + rep(1, 4)
      current$p[, start, segment] = rdirichlet(1, ACGT)
    }
  }
  for(segment in 1:current$nsegments)
  {
    ss = howmanysegmentswitches(segment)
    current$lambda[, segment] = rdirichlet(1, 1+ss)
storage$p[iteration,,]=current$p
storage$lambda[iteration,,]=current$lambda
}

The kernel density plots of the posterior for the 32 parameters are plotted in figure 6.8. There are some ostensibly significant differences between the two segments, but this finding is, of course, not robust since we made up the segments’ locations. However, it provides evidence that we may be able to estimate the segments in the next phase.

Figure 6.8: Model 3: hidden Markov model, without updates to hidden states. Transition probabilities, by starting nucleotide. Green: A as ending nucleotide; orange: C; red: G and blue: T. Solid lines: segment type 1; dashed: type 2.
Intron 7 of chimpanzee α-fetoprotein gene V4

We now relax the assumption that the segment types are known and specified initially. One way to propose changes to them is one at a time, which might also be from the full conditional distribution (i.e. a Gibbs step), but as argued by Boys et al, this is likely to lead to poor mixing. Boys et al propose a ‘backwards scan’ algorithm to propose a change to all segments in one step, by first:

- Calculating the marginal probability for each segment using the previous segment from the last iteration;
- Simulating the final segment from this;
- Simulating each preceding segment conditional on the following (just simulated) segment.

This very clever algorithm thus allows an entire sweep to be simulated and accepted with probability 1, and yields rather good mixing. It is coded as follows. As in Boys, we assign an informative prior distribution to the λ parameters to prevent excessive jumping from segment to segment, in particular, we set α = 100 for the component relating to staying in the same segment and α = 1 for the others.

```r
nsegments=2
current=list(nsegments=nsegments,
    p=array(rep(0.25,16),c(4,4,nsegments)),
    segment=rep(1,data$n),
    lambda=array(rep(1/nsegments,nsegments^2),
    c(nsegments,nsegments))
)
current$segment[1:100]=rep(2,100)
current$segment[1751:1900]=rep(2,150)
MCMCits=10000
storage=list(p=array(0,c(MCMCits,4,4,nsegments)),
    lambda=array(0,c(MCMCits,nsegments,nsegments)))
for(iteration in 1:MCMCits)
{
    if(iteration%%100==0)print(iteration)
    for(start in 1:4)
    {
        for(segment in 1:current$nsegments)
        {
            ACGT=howmanyACGTs(start,segment)+rep(1,4)
            current$p[,start,segment]=rdirichlet(1,ACGT)
        }
    }
    for(segment in 1:current$nsegments)
    {
        ss=howmanyswitches(segment)
        alpha=rep(1,current$nsegments);alpha[segment]=100
    }
}
```
current$lambda[, segment] = rdirichlet(1, alpha+ss)
"
## forward scan
probsegments = array(0, c(data$n, current$nsegments))
probsegments[1,] = rep(1/current$nsegments, current$nsegments)
for (k in 2: data$n)
{
  pro = t(current$p[data$acgt[k],
          data$acgt[k-1],] * (current$lambda[,]
          %*% probsegments[k-1,]))
  pro = pro / sum(pro)
  probsegments[k,] = pro
}
## backward scan
current$segment[data$n] = sample(1: current$nsegments, 1, prob = probsegments[data$n,])
for (k in (data$n-1): 1)
{
  pro = current$lambda[current$segment[k+1],] * probsegments[k,]
  current$segment[k] = sample(1: current$nsegments, 1, prob = pro / sum(pro))
}
storage$p[iteration,,] = current$p
storage$lambda[iteration,,] = current$lambda
"

This is no longer a Monte Carlo sample and so we should assess convergence. Traceplots for the probability of moving from A to an A in the two segments, and for staying in each segment, are plotted in figure 6.9 (traceplots for the other parameters are similar). As the routine ran, I had it output the current segment configuration, which appeared to have converged. Parameter estimates are presented in figure 6.10.

It would be good to be able to visualise switches between segments by obtaining the marginal distribution of the segment at each location. To do this, I add the following lines:

## At the beginning:
store_segments = array(0, c(data$n, nsegments))
## At the end of each iteration
for (k in 1: current$nsegments)
  store_segments[, k] = store_segments[, k] + current$segment == k

We can then plot the posterior probability each location belongs to each segment. This is plotted in figure 6.11.
6.1. HIDDEN MARKOV MODELS

Figure 6.9: Traceplot for Model 4: hidden Markov model, with updates to hidden states.

Intron 7 of chimpanzee α-fetoprotein gene V5

We now allow a third segment type. The code requires only `nsegments=2` being changed to `nsegments=3` and setting different initial conditions. The mixing is fine (figure 6.12), and we would proceed to analyse the output by extending the previous graphs to plot the information from the third segment type (left as a challenge to the enthusiastic reader).

Activities

Consider the mining accidents data once more. Fit a model in which the number of accidents per year is $D_t \sim Po(\lambda_t)$ and where $\lambda_t$ is allowed to change from year to year. Two models for $\lambda_t$ you might try are:

- $\log(\lambda_t) \sim N(\log(\lambda_{t-1}), \sigma^2)$ where $\sigma$ controls the amount of variation from
Figure 6.10: **Model 4: hidden Markov model, with updates to hidden states.** Transition probabilities, by starting nucleotide. Green: A as ending nucleotide; orange: C; red: G and blue: T. Solid lines: segment type 1; dashed: type 2.
Figure 6.11: **Posterior for segment type, model 4.**

year to year; or

- \( \lambda_t = \phi_{S_t} \) where \( S_t \) is a discrete space, discrete time Markov chain, as in the chimpanzee DNA example in the notes.

References


Figure 6.12: Traceplot for Model 5: three segment hidden Markov model.