5.3 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.

5.3.1 Intron 7 of chimpanzee α-fetoprotein gene V1

This example is based on a cool paper by Boys et al (2000, *J Roy Stat Soc ser C Appl Stat* 49:269–85) 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 5.2. As you will know, each base pair can be represented by one of the letters A, C, G and T.

Table 5.2: **Data for chimpanzee α-fetoprotein example.**

```
gtgaagagtc ttgcttctta aaaaagatga ttttcactcc cttttctttc tttttgtctc
attctaaaag ggagaaggtt gtttgacttg aattggttac agagtatgta aactaggtga
ttccttaaat ttgcagaatt ctcgatagca aaacttaaac catcttttgt tgatcctggc
tttcacttta gctatacccc tttttgtgaa acaaaggccc atctatttct tacttctaaa
aaaaccatgg gaacttctca gaaggcttct ccatagttac ttggaggacg ggaggaaact
aagttttaat gtatttattt tttcattcat ttattctttc atttgacaaa taaatatata
ttaaatactt tctatctgct agccactatg acagacactt gttgtaaaag cacaggctga
cctcgaggaa ttcacagtct gataggagag ataagacagt gacctccttg gagttaggga
ctgccttggt ttactgttat ctccatagca caatgcctgg cacatggaag gcattctata
atagtttgtt aaatgaacga atacaataaa aatagcagaa gtaactgtcc taccaggtaa
aaagctagcc atgccaaaga caagtgtgaa taaagtggtc tcggagaatt agaaaacaaa
atttaaaaaa cccagcaaca gtttcttgag tgtgctctag tcagtggtga cttaagcagt
gtggcattgg cttattttag ctaaccctag acttcctaga ttttacaaga gaggactgtt
gaccctcaga ttactctgtc tctgtgggcc ccatgacaca ccaaagagat taaaatccaa
ggagcttaaa aattactgct cttaggtact caaatggctt gataaccagc acgggactat
ggtttccaga aggctgaaag tgaagataaa atgctcattt cagcccactc cgatggcaat
tcagtgagat gcttgaagta agcaagaagc cgaggctgca ggggaggcct gagagtgaca
gtccctgagg agctggggaa gaagtgggag gaggcagcct ggcaggcgac tgtactactt
tacttgtttt tcttctagaa gtatcgcatc ctgggaaaac caacaagaag tattttgttt
ttctttgagc tcagttttcc cattttgaac ggacaatttt actgtttctc gttgtcattt
taaaaaagtt agtttttttca atttttggag ctcataacca cccttttcct ttaaagtgga
aacattaatt tcagcatgat atgtaagttg gattttgata gctgaataat gggttctaat
tatctttgct gagaatgtac agaattttca gtcccatgac aggtatatat gtaagctctg
cctctctctg gccacttagg tgcattgcca ttttattatc tatagactgc cctctgaagg
tcatagtcag tcactgcagt atgttctgat gatgattatc attcttacgg aatttctcat
ggagcagaaa gtttgctctc catgttatga taccagttgc aagtgttgtt taggggcaaa
tttgaatgct aatagaaata tatatagcaa catgcatcct attttatttg agcacatttc
cctcttattt gtaaaggttt tcaattaaaa taacataggg tttagcttac aactatggaa
agaagaaatg aacaaacagg taagtggaaa ggaatgagaa aaggcaaaag tggggagaag
gcactaaaac gggagacaag ttaaaatgtc ttctttcttc ttctttcctt cctccttccc
ccatccctct ctacttccct ttccccttcc ttctttcctg gccttttttc tttctttctt
tgttcccttc tccctccctc cccttttcttc cttttttctaa agctggcttt gagatccttt
attaaagaat aaatctttaa aacttatact ttattttccc tgttgcag
```

A moving average plot (figure 5.10) indicates regions of heterogeneous nucleotide frequencies, and a cusum plot (figure 5.11) suggests three possible regions, near the beginning, middle and end of this sequence, with the latter the most pronounced.

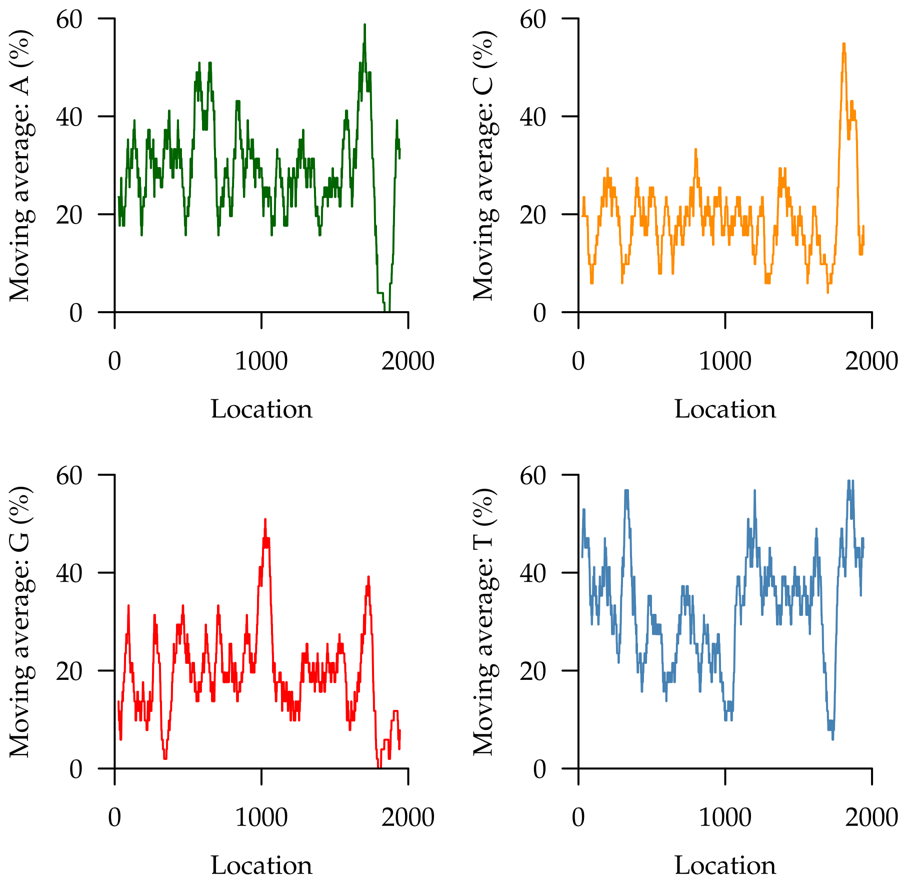


Figure 5.10: **Moving average plots of frequency of specific nucleotides in chimpanzee example**. Moving averages are of length 50.

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

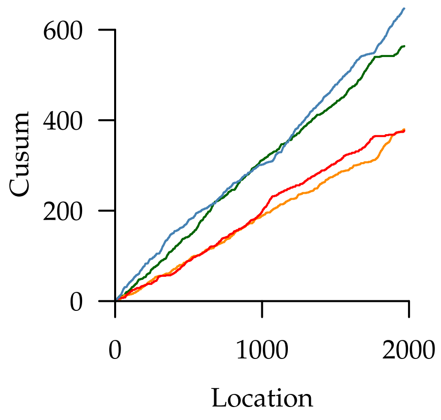


Figure 5.11: **Cusum plot of nucleotides in chimpanzee example**. Green: A; orange: C; red: G and blue: T.

nucleotides:

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

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,

$$\mathrm{p}\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \frac{\prod_{i=1}^4 \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^4 \alpha_i)} \prod_{i=1}^4 p_i^{\alpha_i - 1}.$$

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 5.12, 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

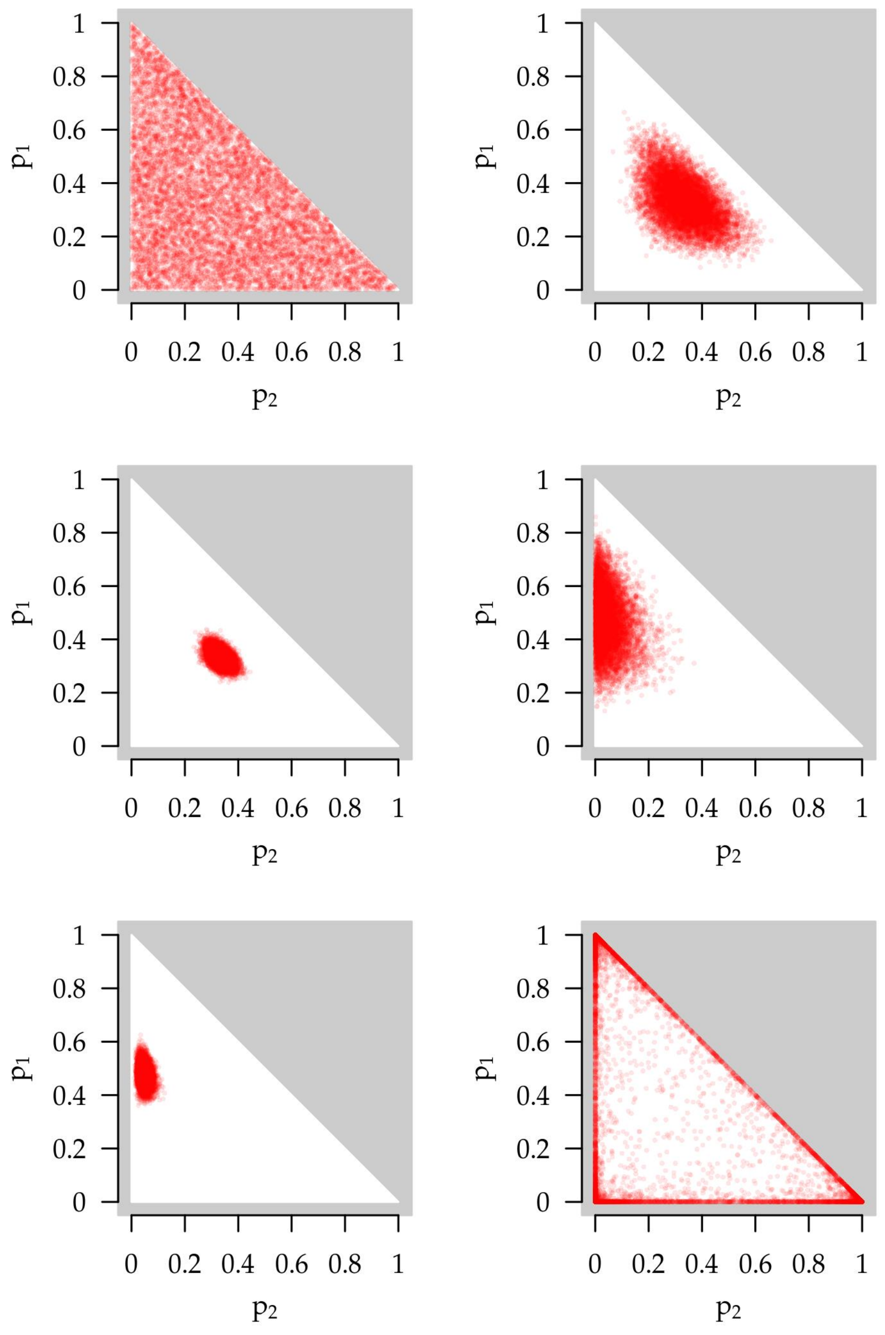


Figure 5.12: **Dirichlet distributions**. In all cases, $p = (p_1, p_2, p_3)$ is Dirichlet$(\alpha_1, \alpha_2, \alpha_3)$. Top left: $(\alpha_1, \alpha_2, \alpha_3) = (1, 1, 1)$. Top right: $(\alpha_1, \alpha_2, \alpha_3) = (10, 10, 10)$. Middle left: $(\alpha_1, \alpha_2, \alpha_3) = (100, 100, 100)$. Middle right: $(\alpha_1, \alpha_2, \alpha_3) = (1, 10, 10)$. Bottom left: $(\alpha_1, \alpha_2, \alpha_3) = (10, 100, 100)$. Bottom right: $(\alpha_1, \alpha_2, \alpha_3) = (0.1, 0.1, 0.1)$.

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:

```
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]=="c")xn[i]=2
  if(x[i]=="g")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 5.13.

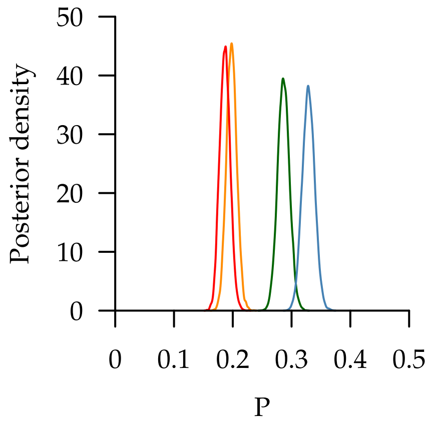


Figure 5.13: **Model 1: no autoregressive structure**. Green: A; orange: C; red: G and blue: T. Note that I have cheated by offsetting the orange curve by +0.5%, and the red by −0.5%, to make them visually distinct.

5.3.2 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 ith is taken to be $\text{p}(Y_{i+1}|Y_i) = p_{Y_i,Y_{i+1}}$. 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:

```
howmanyACGTs=function(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.

```
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
}
```

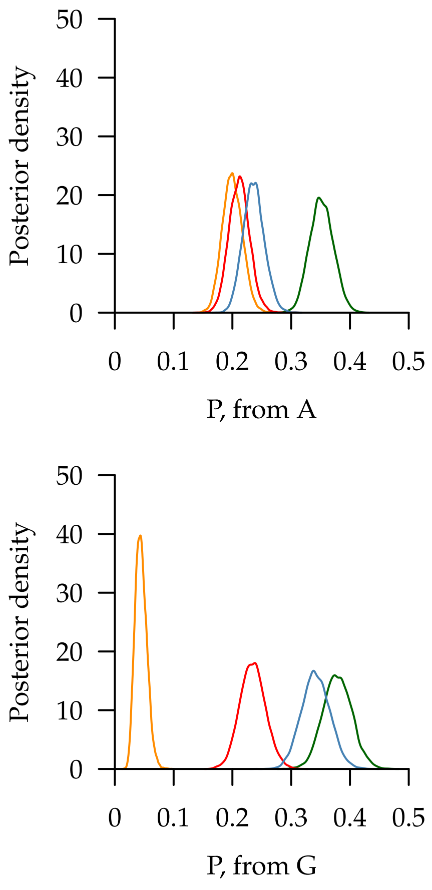
Again, we need not assess convergence. The posteriors for each of the parameters are plotted in figure 5.14.

5.3.3 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.

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:

```
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)
```



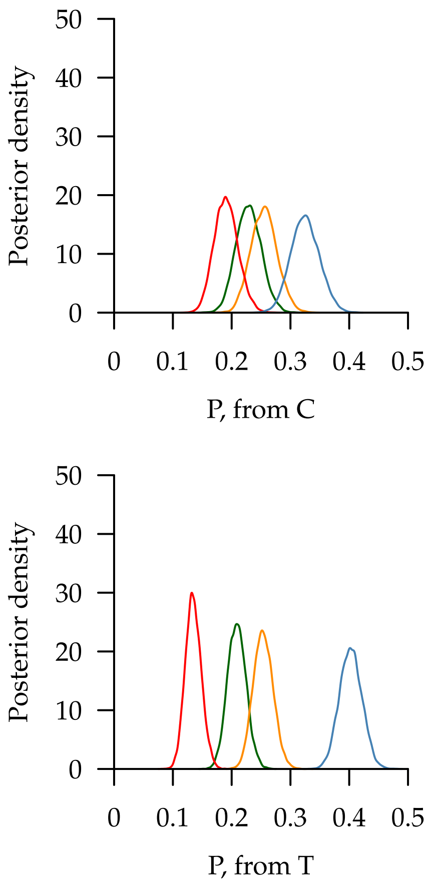


Figure 5.14: **Model 2: simple Markov model**. Transition probabilities, by starting nucleotide. Green: A as ending nucleotide; orange: C; red: G and blue: T.

```
  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 λ, 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 5.15. 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.

5.3.4 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 Gibb's 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 $\alpha = 100$ for the component relating to staying in the same segment and $\alpha = 1$ for the others.

```
nsegments=2
current=list(nsegments=nsegments,
```

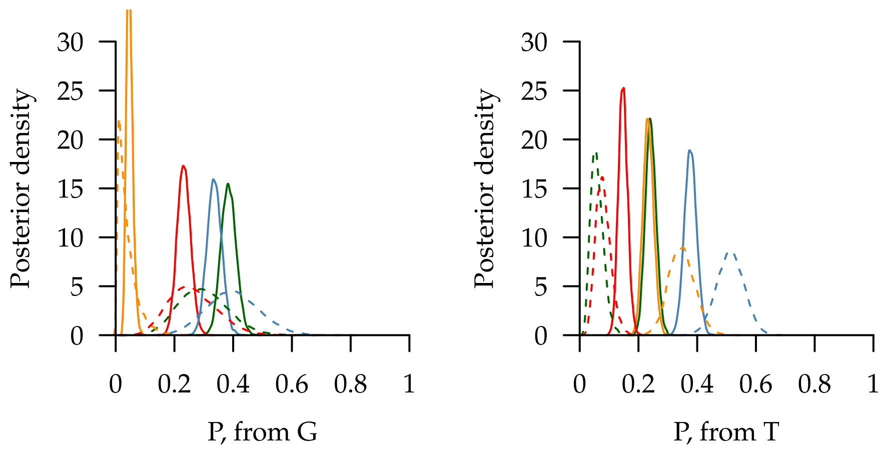


Figure 5.15: **Model 3: hidden Markov model, without updates to hidden states**. Transition probabilities, by starting nucleotide. Green: A as ending nucleotide; orange: C; red: G and blue: T. Solid lines: segment type 1; dashed: type 2.

```
             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=howmanysegmentswitches(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 5.16 (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 5.17.

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 5.18.

5.3.5 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. Running the routine, we observe what initially appears to be poor mixing (figure 5.19), but which on further investigation is an example of the label switching problem: we can arbitrarily switch the labels on the segment types ($1, 2, 3 \to 3, 2, 1$ for instance) and switch the transition probabilities accordingly, and regain the same likelihood. Boys et al suggest ordering each draw

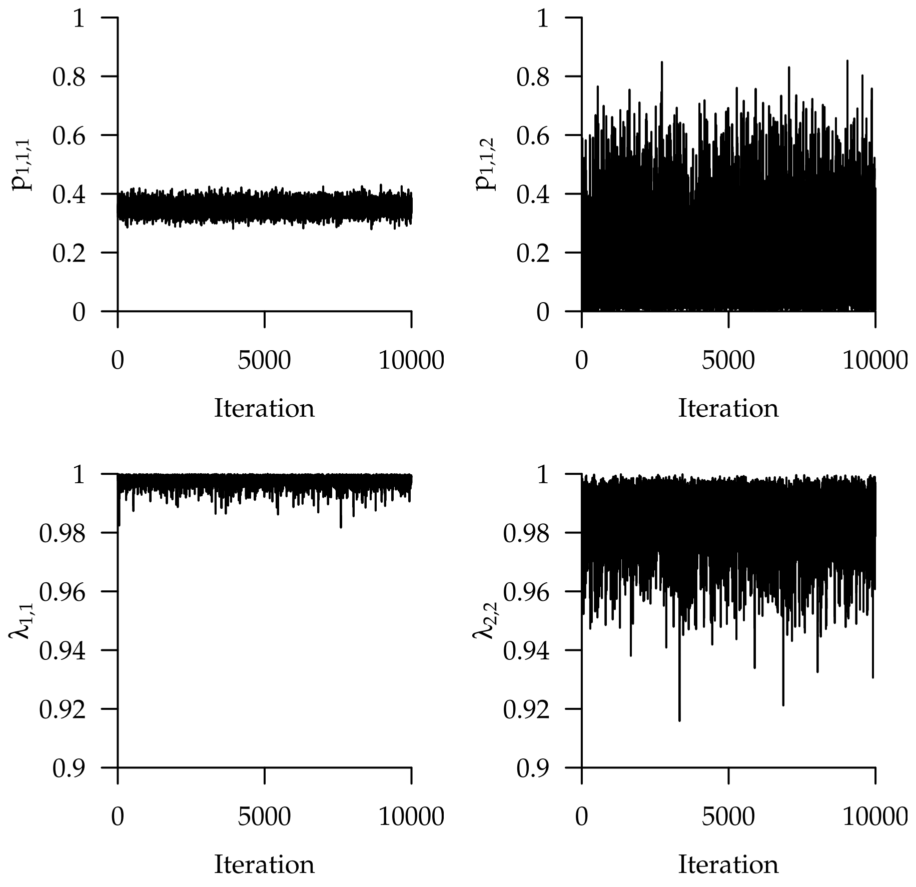


Figure 5.16: **Traceplot for Model 4: hidden Markov model, with updates to hidden states**.

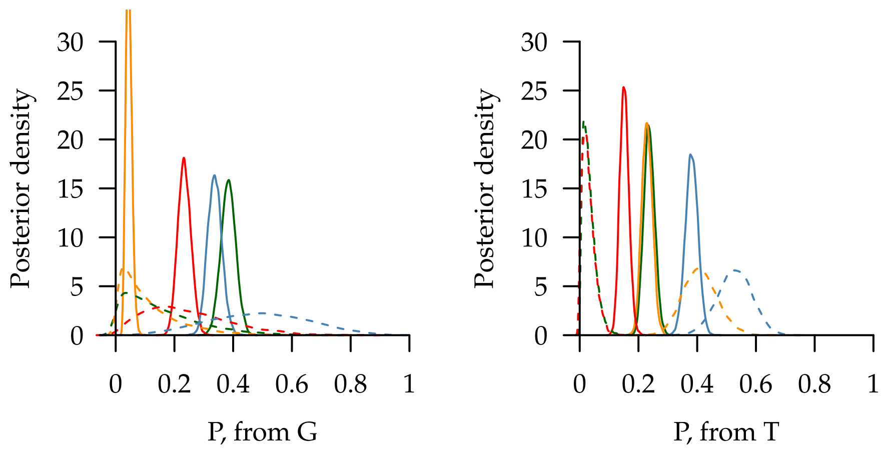


Figure 5.17: **Model 4: hidden Markov model, with updates to hidden states**. Transition probabilities, by starting nucleotide. Green: A as ending nucleotide; orange: C; red: G and blue: T. Solid lines: segment type 1; dashed: type 2.

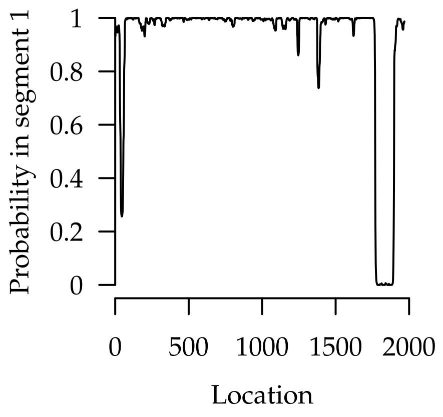


Figure 5.18: **Posterior for segment type, model 4.**

by the Fröbenious norm, i.e.

$$\sum_{i=1}^{4}\sum_{j=1}^{4} p_{i,j}^2$$

and then assigning segment labels in order (so segment 1 has the lowest Fröbenious norm, and so on). This partially, though not fully, resolves the problem (figure 5.20). If we were doing this for real, then, we would, as our next steps in the analysis, wish to rerun the routine, reordering the labels for each segment using either the Fröbenious norm or some other heuristic, and then use the posterior samples to assess the goodness-of-fit, for instance by simulating cusum charts or predicting the occurence of triplets in the data (rather than just pairs).

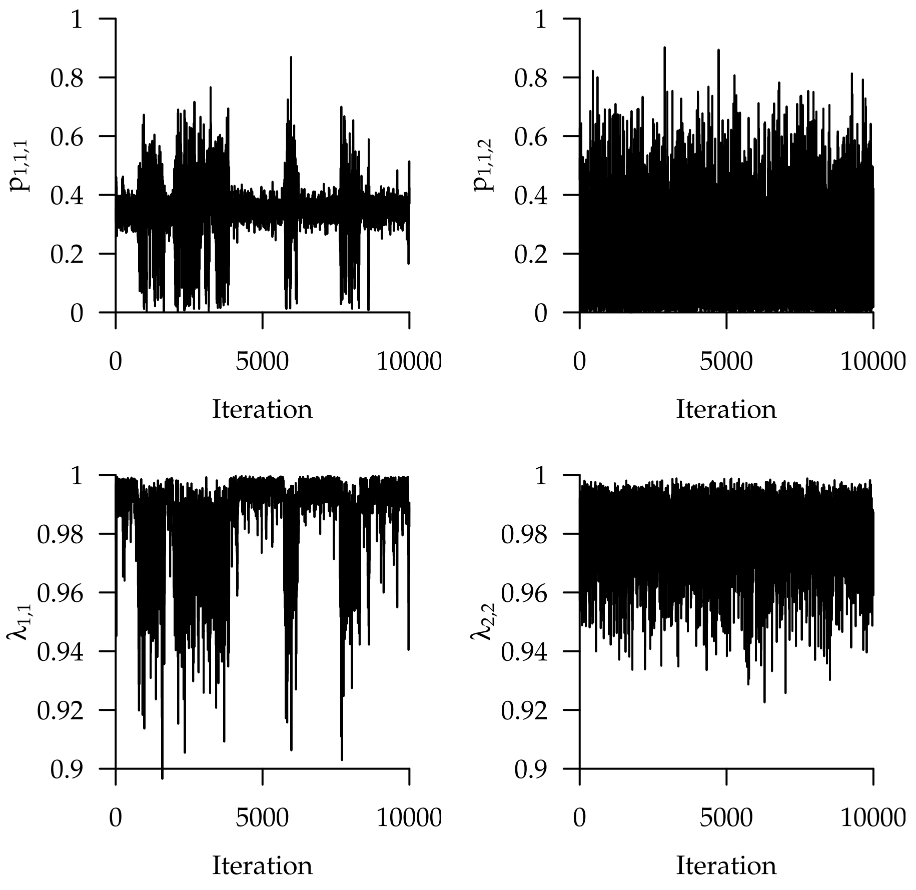


Figure 5.19: **Traceplot for Model 5: three segment hidden Markov model.**

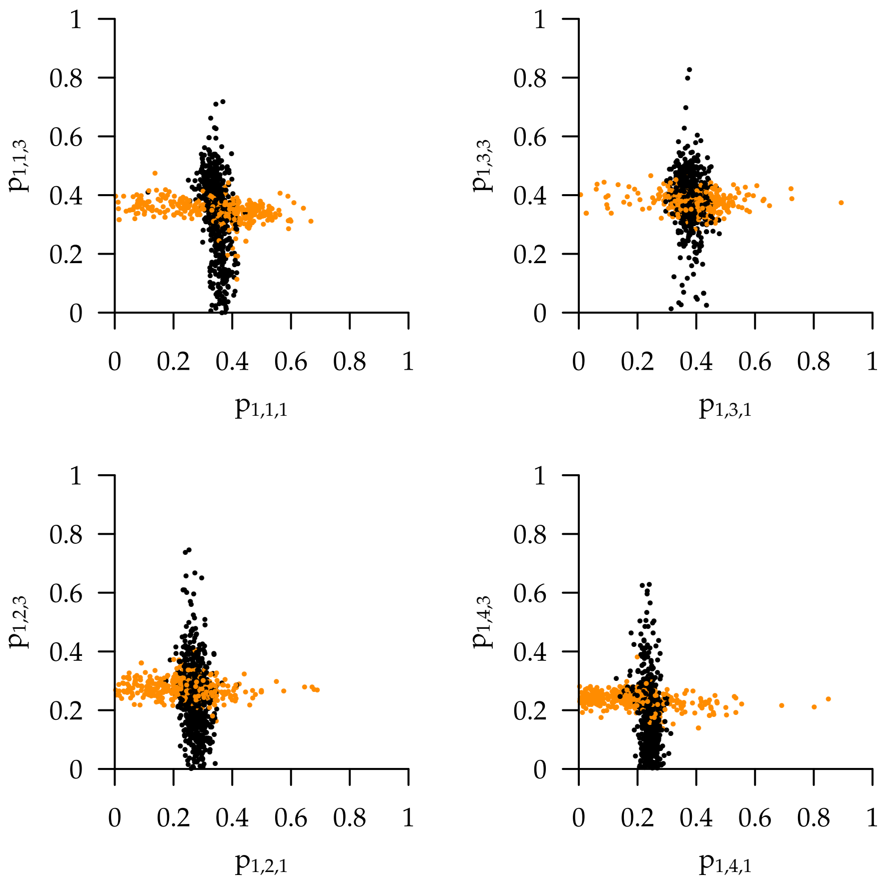


Figure 5.20: **Samples from posterior for segments 1 and 3, labelled post-hoc by Fröbenious norm.** Black is the smaller and orange the larger norm.