Genome Sequence Analysis assignment 1

303034908

21/01/2019

1. Write a program which reads a $K \times K$ transition matrix and a $1 \times K$ initial distribution from a file and outputs N elements of a Markov chain

```
create.chain <- function(ptrans, p0, N){
  S <- 1:(dim(ptrans)[1])
  chain <- rep(0, N)
  chain[1] <- sample(S, 1, prob = p0)
  for (i in 2:N){
    chain[i] <- sample(S, 1, prob = ptrans[chain[i - 1], ])
  }
  return(chain)
}

Markov.chain <- function(file, N){
  ## file in csv format
  ## N: lenght of markov chain
  ## ptrans: transition matrix
  ## p0: initial distribution
  input <- read.csv(file)
  ## some editing may be needed to input file
  ## correctly depending on type of file
  ptrans <- input[[1]]
  p0 <- input[[2]]
  chain <- create.chain(ptrans, p0, N)
  return(chain)
}
```

2. Write a program which reads a sequence of state indices (i.e. elements of {1, … , K}) and infers a maximum likelihood Markov chain transition matrix and initial distribution

```
find.ptrans <- function(state.indices){
  ## infer the maximum likelihood transition matrix and initial distribution
  ## state.indeces: sequence of state indices from 1 to K
  # initialise
  unique.indices <- unique(state.indices)
  status.prob <- rep(0, length = length(unique.indices))
  ptrans <- matrix(0, nrow=length(unique.indices), ncol=length(unique.indices))
  p0 <- numeric(length(unique.indices))
  # add to matrix by indexing over rows and columns
  for (i in 1:(length(state.indices)-1)){
    ptrans[state.indices[i], state.indices[i+1]] <- c(ptrans[state.indices[i],
                                                            state.indices[i+1]]+1)
  }
  sums <- apply(ptrans, 1, function(x) sum(x)) #sum over rows
  ptrans <- (ptrans/sums) #normalise

  # probability of each status over the sequence
  for (state in state.indices){
    status.prob[state] <- status.prob[state]+1
  }
  status.prob <- status.prob/length(state.indices)

  #initial distribution
  p0[state.indices[1]] <- 1

  return(list("ptrans" = ptrans, "init.dist" = p0  ,"status.prob" = status.dist))
}

find.ptrans(state.indices)
```

3. Implement a HMM by modifying your program from (1), adding an emitted variable at each point in the chain. Use it to simulate 115 emitted values from the following model:

$$S = 0,1, V = 1,2,3,4,5$$
$$A = \begin{pmatrix} 0.8 & 0.2 \\ 0.1 & 0.9 \end{pmatrix}$$

$$\mu^0 = \begin{pmatrix} 0.5, & 0.5 \end{pmatrix}$$

$$B = \begin{pmatrix} 0.2 & 0.5 & 0.2 & 0.1 & 0 \\ 0 & 0.1 & 0.4 & 0.4 & 0.1 \end{pmatrix}$$

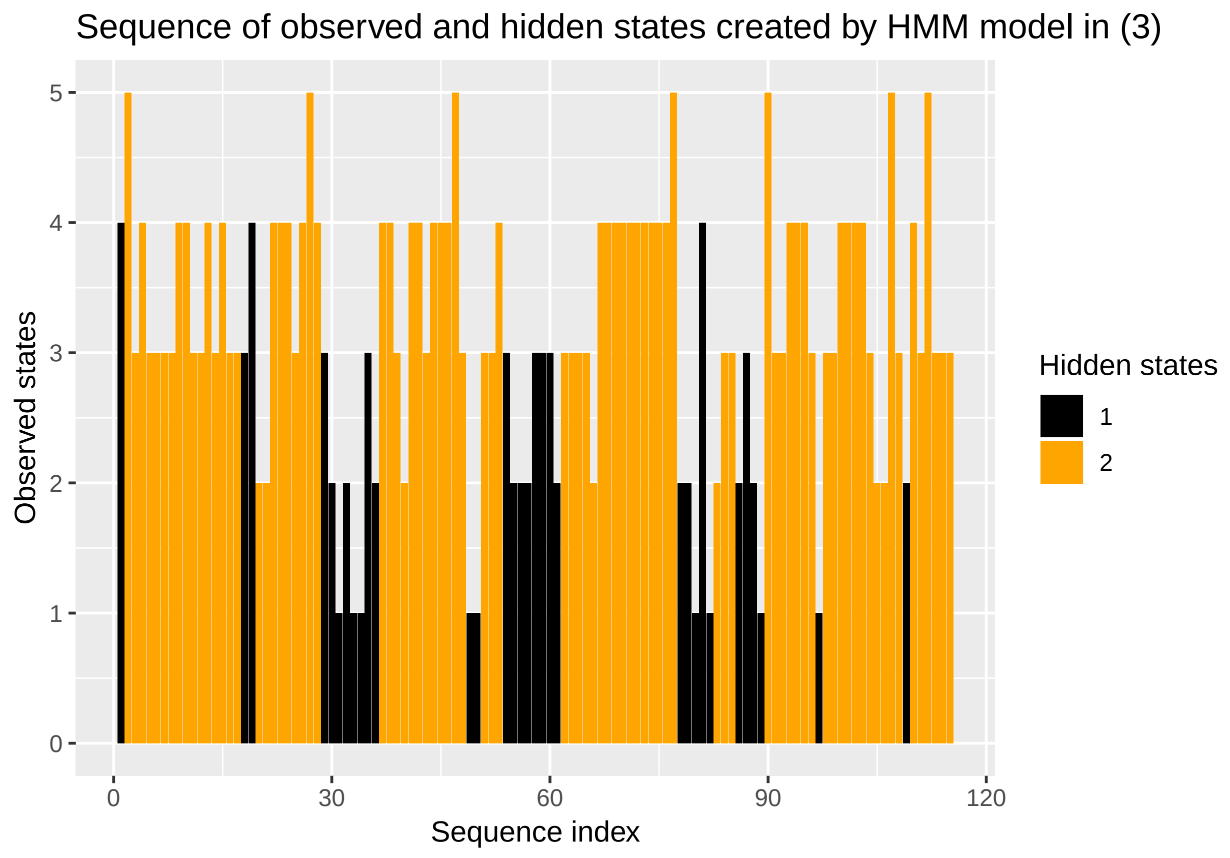
Plot the resulting sequences of hidden and emitted states on the same graph.

```
HMM = function(ptrans, p0, pemit, N){
  # simulate observed sequence from a HMM with transition matrix ptrans,
  # intial distribution p0, and emission probabilities pemit
  hidden = create.chain(ptrans, p0, N)
  if (is.matrix(pemit)){
    emit.states = colnames(pemit)
    observed = lapply(hidden, function(x){sample(emit.states, 1, prob = pemit[x, ])})
  } else if (is.function(pemit)){
    observed = lapply(hidden, pemit)
  }
  observed <- as.numeric(unlist(observed))
  return(list(hid = hidden, obs = observed))
}


h.states <- c(0,1)
e.space <- c(1,2,3,4,5)
p0 <- c(0.5,0.5)
ptrans <- matrix(c(0.8, 0.1, 0.2, 0.9), nrow=2, ncol=2)
colnames(ptrans) <- h.states
rownames(ptrans) <- h.states
p.emission <- matrix(c(0.2, 0, 0.5, 0.1, 0.2, 0.4, 0.1, 0.4, 0, 0.1), nrow=2, ncol=5)
colnames(p.emission) <- e.space
rownames(p.emission) <- h.states

Hmm1 <- HMM(ptrans, p0, p.emission, 115)
```

```r
print(Hmm1)
```

```
## $hid
##   [1] 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1
##  [36] 1 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 1 1 1 1 1 2 2 2 1 1 1 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2
## [106] 2 2 2 1 2 2 2 2 2 2
##
## $obs
##   [1] 4 5 3 4 3 3 3 3 4 4 3 3 4 3 4 3 3 3 4 2 2 4 4 4 3 4 5 4 3 2 1 2 1 1 3
##  [36] 2 4 4 3 2 4 4 3 4 4 4 5 3 1 1 3 3 4 3 2 2 2 3 3 3 2 3 3 3 3 2 4 4 4 4
##  [71] 4 4 4 4 4 4 5 2 2 1 4 1 2 3 3 2 3 2 1 5 3 3 4 4 4 3 1 3 3 4 4 4 4 3 2
## [106] 2 5 3 2 4 3 5 3 3 3
```

```r
Hmm1.df <- data.frame("hidden" = as.numeric(Hmm1[[1]]),
                      "observed" = as.numeric(Hmm1[[2]]))

p <- ggplot(Hmm1.df, aes(x= 1:dim(Hmm1.df)[1], y = observed, fill=as.factor(hidden))) +
  geom_bar(stat="identity") + labs(fill='Hidden states') +
  xlab("Sequence index") +  ylab("Observed states") +
  ggtitle("Sequence of observed and hidden states created by HMM model in (3)") +
  scale_fill_manual(values=c('black','orange'))

p
```



4. Implement the forward algorithm with scaling to calculate the likelihood of an emitted sequence given the model, reading the emitted sequence from a file.

Let $\lambda = (A, B, \pi)$ be a given model and let $O = (O_1, O_2, ..., O_T)$ be a series of observations. We want to find $P(O|\lambda)$. To do this we can implement the forward algorithm: for $t = 1, 2, ..., T$ and $i = 1, 2, ..., S$ (where S is the number of hidden states), define

$$\alpha_t(i) = P(O_1, O_1, ..., O_t, Z_t = q_i|\lambda)$$

$\alpha_t(i)$ is the probability of the partial observation sequence up to time t, where the underlying hidden state Z is in state q_i at time t.

The forward algorithm can be computed recursively as follows:

1) Let $\alpha_0(i) = \mu_i^0 b_i(O_1)$ for $iin1, 2, ..., S$.
2) For $t = 1, 2, ..., T$ and i in $1, 2, ..., S$, compute

$$\alpha_t(i) = \left[\sum_{j=0}^{S} \alpha_{t-1}(j) a_{ji}\right] b_i(O_t)$$

3)

$$P(O|\lambda) = \sum_{i=0}^{S} \alpha_T(i)$$

For scaling, each $\alpha_t(i)$ value was divided by the sum of the $\alpha_t(i)$ at time t, and the sum was maintaned in a vector called c. The likelihood can be calculated by the product of the scaling elements in c or by the sum of the logs of these scaling elements.

```r
#emitted.sequence <- read.csv(file)
emitted.sequence <- Hmm1.df[1:115, 2]


alpha.recursion <- function(observed, p0, ptrans, pemit){
   ## probability of the partial observation sequence up to time t,
  ## where the underlying hidden state Z is in state i at time t
  K <- length(p0)
  L <- length(observed)
  alpha <- matrix(0, nrow=L, ncol=K)
  c <- numeric(L)
   #first alpha
  for (i in 1:K){
    alpha[1,i] <- pemit[i, observed[1]] * p0[i]
  }
    #scaling
  c[1] <- sum(alpha[1,])
  alpha[1,] <- alpha[1,]/sum(alpha[1,])
  for (t in 2:L){
    for (i in 1:K){
          #vectorised summation over js
      alpha[t,i] <- pemit[i, observed[t]] * sum(ptrans[,i]*alpha[(t-1),])
    }
       #scaling
    c[t] <- sum(alpha[t,])
    alpha[t,] <- alpha[t, ]/c[t]
  }
  return(list(alpha,c))
}
```

```r
alpha.list <- alpha.recursion(emitted.sequence, p0, ptrans, p.emission)
alpha <- alpha.list[[1]]
c <- alpha.list[[2]]

log.likelihood <- sum(log(c))
likelihood <- prod(c )
print(paste("Likelihood of given sequence =", likelihood))
```

```
## [1] "Likelihood of given sequence = 1.70326791002694e-69"
```

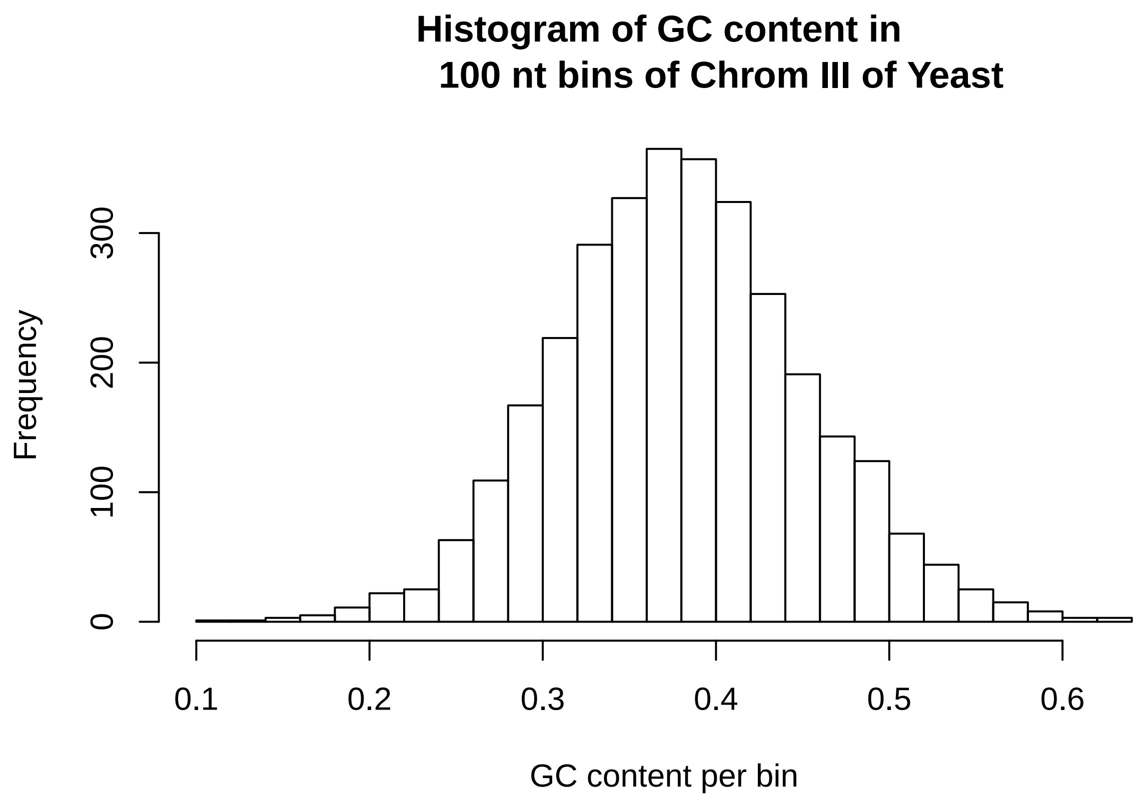
```r
print(paste("LogLikelihood of given sequence =", log.likelihood))
```

```
## [1] "LogLikelihood of given sequence = -158.345822710765"
```

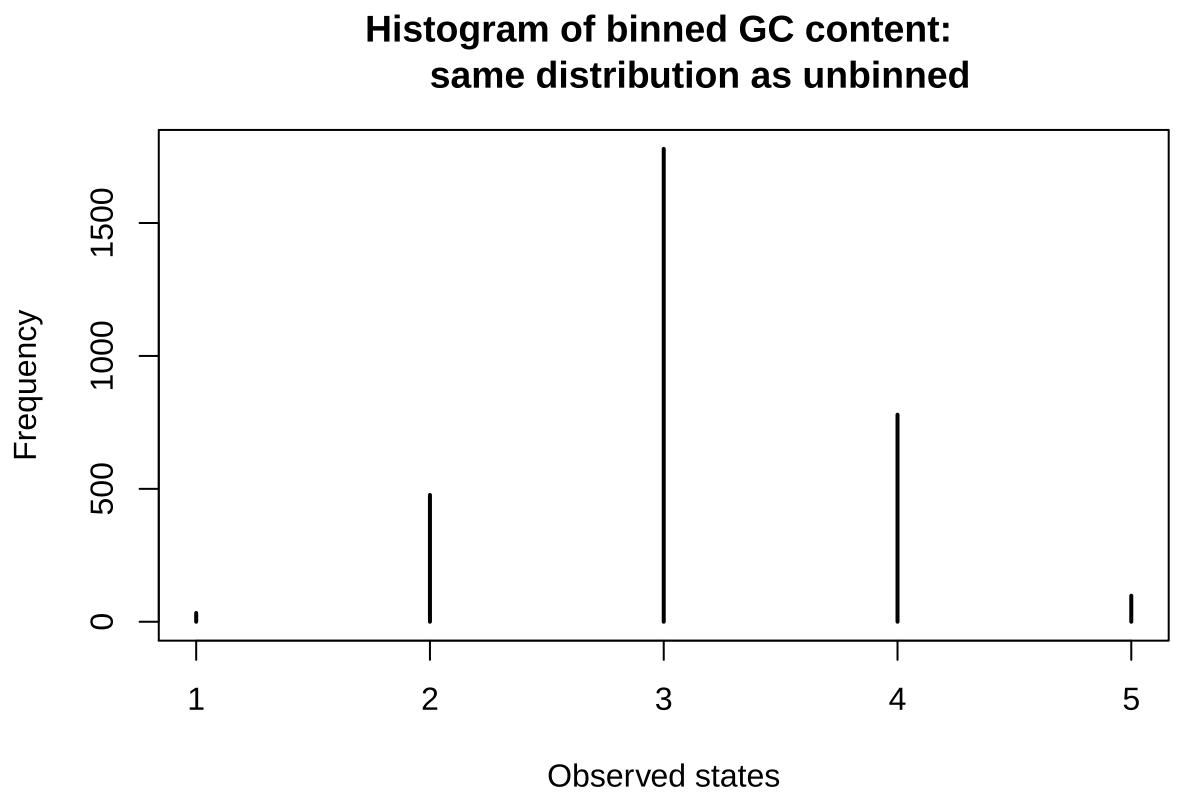
5. Download chromosome III of the Saccharomyces cerevisiae (yeast) genome sequence from Ensembl or NCBI. Calculate its GC content (the fraction of bases which are G or C) in 100 bp windows. Choose an appropriate binning scheme to represent the GC content of each window such that the resulting sequence corresponds to an emission sequence for the model in (3). Calculate the log likelihood of this GC sequence under the model in (3).

The GC content of each 100bp window follows a normal-looking distribution. Further bins of equal size were therefore chosen so as to maintain information about the distribution in the HMM.

```r
chrom3 <- read.fasta("Saccharomyces_cerevisiae.R64-1-1.dna.chromosome.III.fa")

attr(chrom3$III, "class") <- NULL
attr(chrom3$III, "name") <- NULL
attr(chrom3$III, "Annot") <- NULL

sequence <- chrom3$III
binned <- split(sequence, ceiling(seq_along(sequence)/100))

GC.content <- function(bin){
  G <- sum(grepl("g", bin))
  C <- sum(grepl("c", bin))
  content <- (G+C)/length(bin)
  return(content)
}

GC.bins <- lapply(binned, function(x) GC.content(x) )
GC.bins <- unlist(GC.bins)
par(mfrow=c(1,1))
rr <- hist(GC.bins, breaks=20, main="Histogram of GC content in
           100 nt bins of Chrom III of Yeast", xlab="GC content per bin")
```



```
GC.bins.df <- data.frame(GC.bins)
cuts <- apply(GC.bins.df, 2, cut,
              seq(from=min(GC.bins.df$GC.bins), to=max(GC.bins.df$GC.bins),
                  length.out = 6),  labels=1:5)

GCsequence <- as.numeric(cuts)
GCsequence <- GCsequence[!is.na(GCsequence)]

ff <- plot(table(GCsequence), main="Histogram of binned GC content:
       same distribution as unbinned", xlab="Observed states",
     ylab="Frequency")
```



```r
GC.prob <- alpha.recursion(GCsequence, p0, ptrans, p.emission)
alpha <- GC.prob[[1]]
c <- GC.prob[[2]]

log.likelihood <- sum(log(c))

print(paste("LogLikelihood of GC content sequence given the model =", log.likelihood))
```

```
## [1] "LogLikelihood of GC content sequence given the model = -3773.30951709517"
```

6. Extend your programs in (3) and (4) to implement Baum-Welch estimation of model parameters A, B and $\mu 0$, given an emitted sequence. Using the encoded GC sequence for the yeast genomic sequence produced in (5), estimate new parameters for the HMM specified in (3). Calculate the log likelihood of the data under this new model

The Baum-Welch algorithm is used to re-estimate the model parametrs given an emitted sequence. This algorithm can be broken down into 4 steps: the alpha recursion/forward algorithm, the beta recursion (backword algorithm), the gamma recursion, and the re-estimatino of parameters. The algorithm is then run for a maximum of 100 iterations, aiming for a change in log likelihood between iterations of less than 0.01.

The log likelihood with using the previous parameters reaches around -3700. After re-estimation, it equals around -3250, suggesting a significant improvement. A trace of the log likelihoods over time is shown in the figure below.

```r
beta.recursion <- function(c, observed, p0, ptrans, pemit){
  K <- length(p0)
  L <- length(observed)
  beta <- matrix(0, nrow=L, ncol=K)
  for (j in 1:K){
    beta[L,j] <- 1
  }
  beta[L,] <- beta[L,]/c[L]
  for (t in (L-1):1){
    for (i in 1:K){
      beta[t,i] <- sum(ptrans[i,]*pemit[, observed[t+1]]*beta[t+1,])
    }
    beta[t,] <- beta[t,]/c[t]
  }
  return(beta)
}


gamma.recursion <- function(alpha, beta, observed, p0, ptrans, pemit){
  K <- length(p0)
  L <- length(observed)
  gamma.i.j <- matrix(0, nrow=K, ncol=K)
  gamma <- matrix(0, nrow=L, ncol=K)
  for (t in 1:(L-1)){
    for (i in 1:K){
      gamma.i.j[i,] <- gamma.i.j[i,] + alpha[t,i]*ptrans[i,]*pemit[,observed[t+1]]*beta[(t+1),]
      gamma[t,i] <- sum(alpha[t,i]*ptrans[i,]*pemit[,observed[t+1]]*beta[(t+1),])
    }
  }
  #special case for gamma[T,i]
  for (i in 1:K){
    gamma[L,i] <- alpha[L,i]
  }
  return(list(gamma, gamma.i.j))
}


re.estimate <- function(gamma, gamma.i.j,  observed, p0, ptrans, pemit){
  K <- length(p0)
  L <- length(observed)
  # re-estimate ??
  for (i in 1:K){
    p0[i] <- gamma[1,i]
  }
  #re-estimate ptrans
  denom <- colSums(gamma)
  ptrans <- gamma.i.j/denom
  #re-estimate pemit
  for (i in 1:K){
    denom <- 0
    for (t in 1:L){
      denom <- denom+gamma[t,i]
    }
    for (j in observed){
      numer <- 0
      for (t in 1:L){
        if (observed[t] == j){
          numer <- numer+gamma[t,i]
        }
      }
      pemit[i,j] <- numer/denom
    }
  }
  return(list(p0,ptrans, pemit))
}

baum.welch <- function(maxIters,threshold,  observed, p0, ptrans, pemit){
  L <- length(observed)
  maxIters <- maxIters
  iters <- 0
  oldLogProb <- -Inf
  logProb <- 0
  logdiff <- Inf
  logProbs <- list()
  while (iters < maxIters & logdiff > threshold){
    alpha.list <- alpha.recursion(observed, p0, ptrans, pemit)
    alpha <- alpha.list[[1]]
    c <- alpha.list[[2]]
    beta <- beta.recursion(c, observed, p0, ptrans, pemit)
    gamma.list <- gamma.recursion(alpha, beta, observed, p0, ptrans, pemit)
    gamma <- gamma.list[[1]]
    gamma.i.j. <- gamma.list[[2]]
    params <- re.estimate(gamma, gamma.i.j., observed, p0, ptrans, pemit)
    p0 <- params[[1]]
    ptrans <- params[[2]]
    pemit <- params[[3]]
    logProb <- sum(log(c))
    #print(paste("logProb = ", logProb))
    #print(paste("oldLogProb = ", oldLogProb))
    logdiff <- (logProb - oldLogProb)
    #print(paste("logdiff = ",logdiff ))
    oldLogProb <- logProb
    #logProb <- -logProb
    iters <- iters +1
    #print(paste("iters = ",iters))
    logProbs[[iters]] <- oldLogProb
  }
  logProbs <- unlist(logProbs)
  return(list("p0" = p0, "ptrans" = ptrans, "pemit" = pemit, "c" = c, "logProbs" = logProbs))
}


p.emission.new <- matrix(c(0.2, 0.01, 0.5, 0.09, 0.2, 0.4, 0.09, 0.4, 0.01, 0.1), nrow=2, ncol=5)
params <- baum.welch(100, 0.01, GCsequence, p0, ptrans, p.emission.new)


p0 <- as.matrix(params$p0)
ptrans <- as.matrix(params$ptrans)
pemit <- as.matrix(params$pemit)
c2 <- params$c
logProbs <- params$logProbs

new.Log.likelihoood <- sum(log(c2));new.Log.likelihoood
```

Table 1: Initial probabilities

0
1

Table 2: Transition matrix

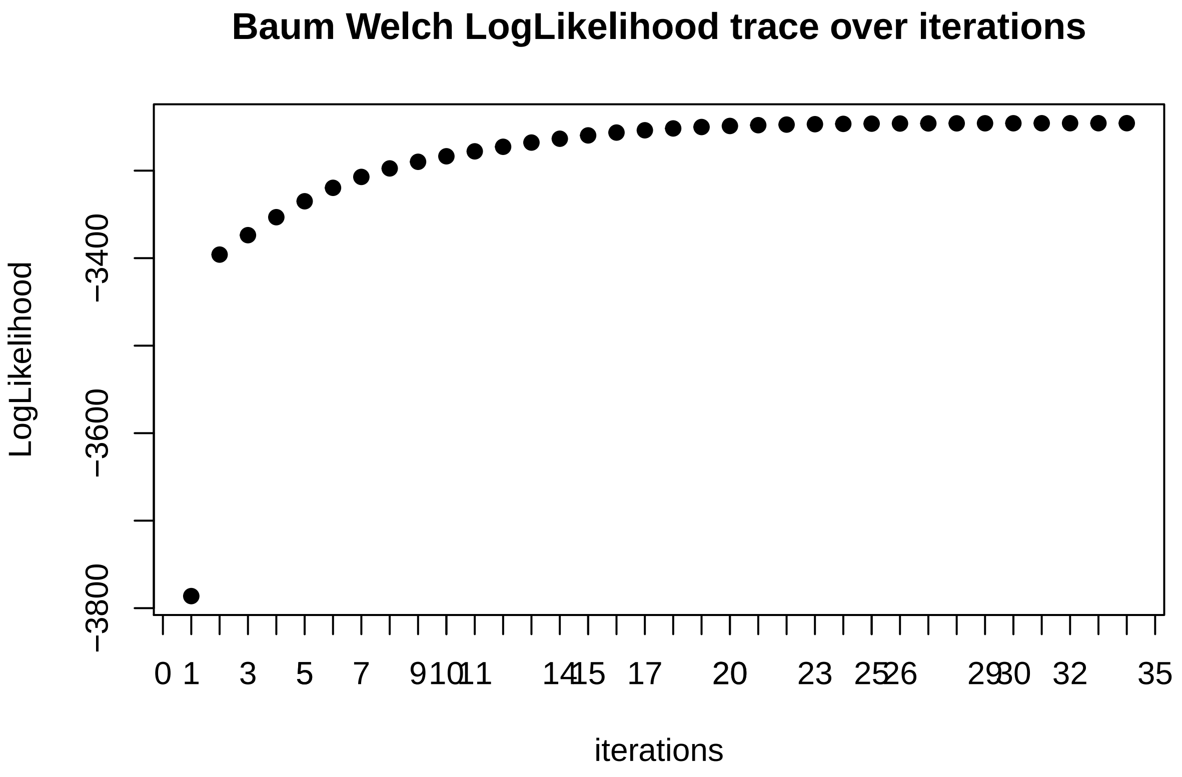
0.9667309	0.0332691
0.0775667	0.9213810

```
## [1] -3245.756
```

```r
print(paste("New log likelihood with updated model parameters = ", new.Log.likelihoood))
```

```
## [1] "New log likelihood with updated model parameters =  -3245.75564624135"
```

```r
plot(logProbs, main="Baum Welch LogLikelihood trace over iterations",
     xlab="iterations", ylab="LogLikelihood", pch=19)
axis(1, at=1:length(logProbs), labels=as.character(1:length(logProbs)))
```



```r
kable(p0, caption="Initial probabilities", format='latex')
```

```r
kable(ptrans, caption="Transition matrix", format='latex')
```

```r
kable(pemit, caption="Emission matrix", format='latex')
```

7. Infer the most likely sequence of hidden states under the model with your new parameters. Plot some of the output along with the corresponding GC content, and demonstrate the significance of your inference in relation to annotation in this and other genome sequences. Are there any extensions or adaptations you could make to your model which might make it more useful for this purpose?

Table 3: Emission matrix

0.0125790	0.2005871	0.6728787	0.1139524	0.0000028
0.0053972	0.0342661	0.3031798	0.5540408	0.1031162

The Viterbi algorithm is used to infer the most likely sequence of hidden states to describe the observed sequence given a model, also written as $q* = argmax_q p(q|O, \lambda)$, where $q*$ is the most likely hidden state, λ is the model and O are the observations.

The Viterbi algorithm "grows" the optimal path $q*$ gradually. At time t it will keep track of all the optimal paths ending at each of the S different states. At time $t + 1$ it will update these S optimal paths.

In more practical ways, the Viterbi algorithm uses the same function as the alpha.recursion, although finding the *max* instead of the *sum* of $\alpha_t(i)$. Finally, the optimal path is given by finding the state index at the highest value at each time point.

Below is a plot of the full binned GC content sequence coloured by the hidden state found by the Viterbi algorithm. As shown, most of the observed elements have a hidden state of 1. These have relatively lower GC content. Instead, sometimes, islands of higher GC content with hidden state 2 arise. The frequency of each hidden state is shown also below.

Discussion

GC content has been shown to be higher at around protein coding genes and promotor regions. This is partly because CpG islands, (C followed by a G in the 5' to 3' direction) can undergo methylation such that the expression of the genes linked to these CpG islands can be epigenetically controlled. Methylation status can also be inherited, and is often linked sex determination.

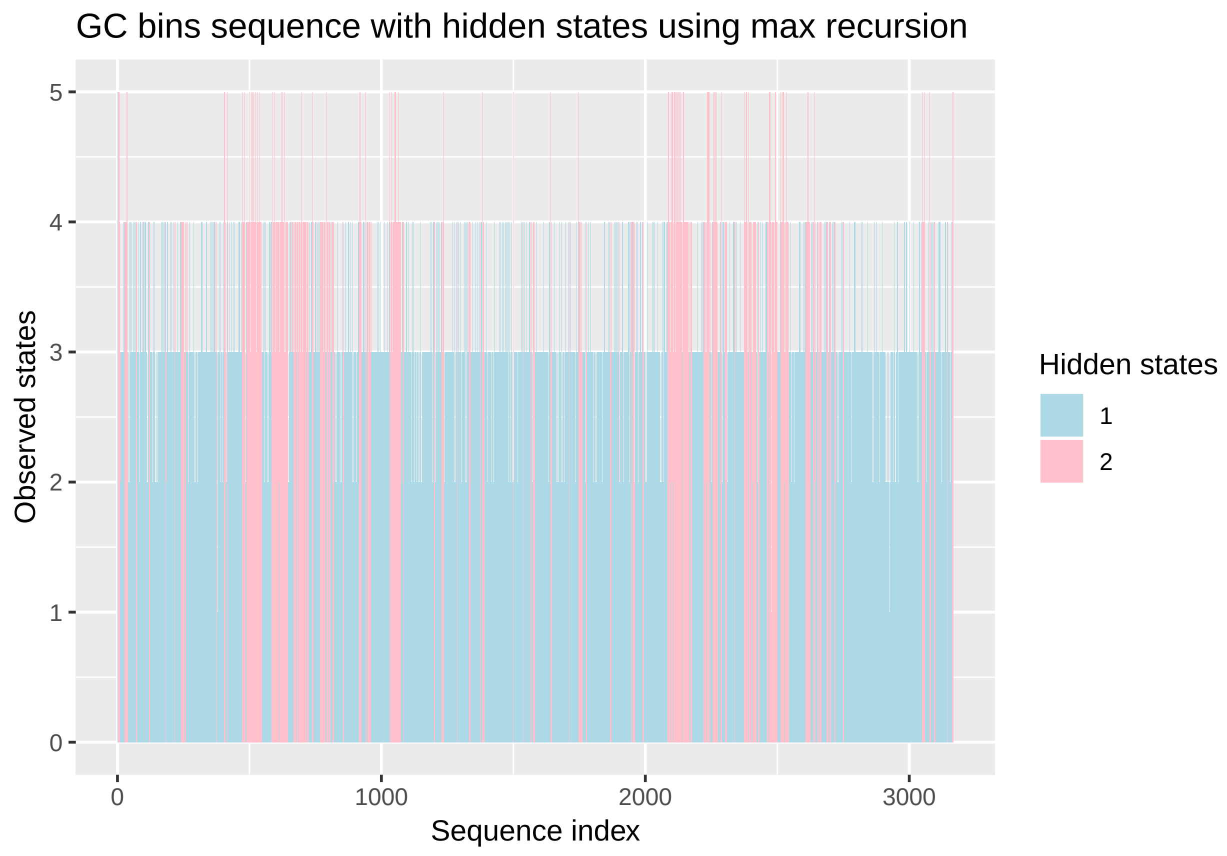
A more species and chromosome specific approach to GC content by Bradman et al., 1999 showed that chromosome III of Yeast has the most pronouced regional variation in GC content. The most likely reason proposed is that chromosome III contains the mating-type loci MAT and HML and HMR which require very regionally dependent control of expression possibly achieved by methylation of these loci.

A traditional definition of CpG islands is a "region of at least 200 bp, with the proportion of Gs or Cs, referred to as GC content, greater than 50%, and observed to expected CpG ratio (O/E) greater than 0.6." (by Gardiner-Garden and Frommer, 1987). However, this definition of the thresholds is somewhat arbitrary. Instead, as the underlying structure of the genome (CpG islands vs baseline) are sequentially correlated, hidden Markov models (HMMs) can be considered as a better method for identifying CpG islands.

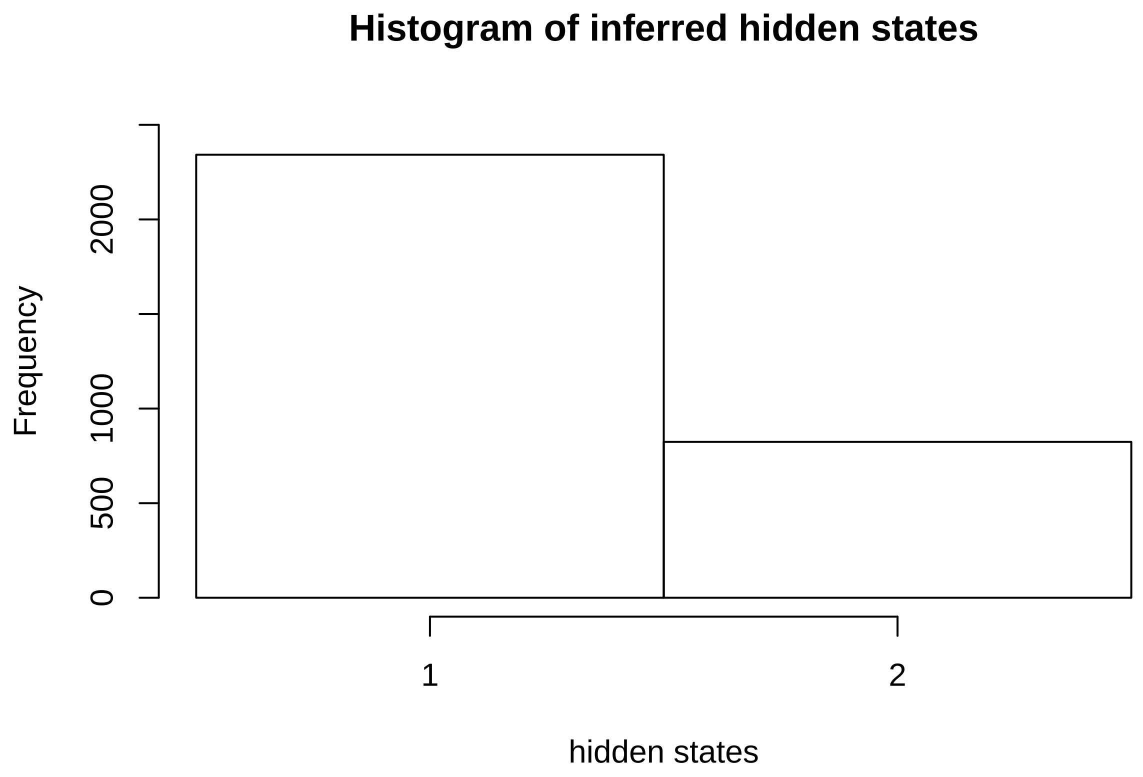
With this in mind, we can now re-interpret the plot below. The GC bins coloured in blue (hidden state 1) are those classified as having a baseline GC content given the previous bin. Instead, the GC bins coloured in pink (hidden state 2), suggest the presence of CpG islands in the chromosome.

This model can, moreoever, be used to determine CpG islands in different species. A challenge when doing this is to decide the most likely parameters to input in the Baum Welch so as to not get stuck at local maxima. This can be done by setting initial probabilities as 0, such that even during the Baum-Welch iterations they remain 0. Secondly, one could be to alter the size of the GC bins, and see if the results show improvement with increased resolution or not.

```r
max.recursion <- function(observed, p0, ptrans, pemit){
  K <- length(p0)
  L <- length(observed)
  q.star <- matrix(0, nrow=L, ncol=K)
  Vit <- matrix(0, nrow=L, ncol=K)
  c <- numeric(L)
  for (i in 1:K){
    Vit[1,i] <- p0[1]*pemit[observed[1]]
    q.star[1,i] <- i
  }
  c[1] <- sum(Vit[1,])
  Vit[1,] <- Vit[1,]/sum(Vit[1,])
  for (t in 2:L){
    for (i in 1:K){
      Vit[t, i] <- pemit[i, observed[t]] * max(ptrans[,i]*Vit[t-1,])
    }
    c[t] <- sum(Vit[t,])
    Vit[t,] <- Vit[t,]/sum(Vit[t,])
  }
  hidden.states <- apply(Vit, 1, which.max)
  return(list(Vit, hidden.states, c))
}

Max.list <- max.recursion(GCsequence, p0.new, ptrans.new, pemit.new)
hidden.states.1 <- Max.list[[2]]
Max.df <- data.frame("hidden" = hidden.states.1,
                     "observed" = GCsequence)
p1 <- ggplot(Max.df, aes(x= 1:dim(Max.df)[1], y = observed , fill=as.factor(hidden))) +
  geom_bar(stat="identity") + labs(fill='Hidden states') + xlab("Sequence index") +
  ylab("Observed states") + scale_fill_manual(values=c('lightblue','pink')) +
  ggtitle("GC bins sequence with hidden states using max recursion")
p1
```



```
h <- hist(hidden.states.1, breaks=2,  xaxt='n', ylim=c(0,2500),
     xlab="hidden states", main="Histogram of inferred hidden states")
axis(1, at=c(1.25,1.75), label=c("1", "2"))
```


Histogram of inferred hidden states
Frequency
0
500
1000
2000
1
2
hidden states