516 Final

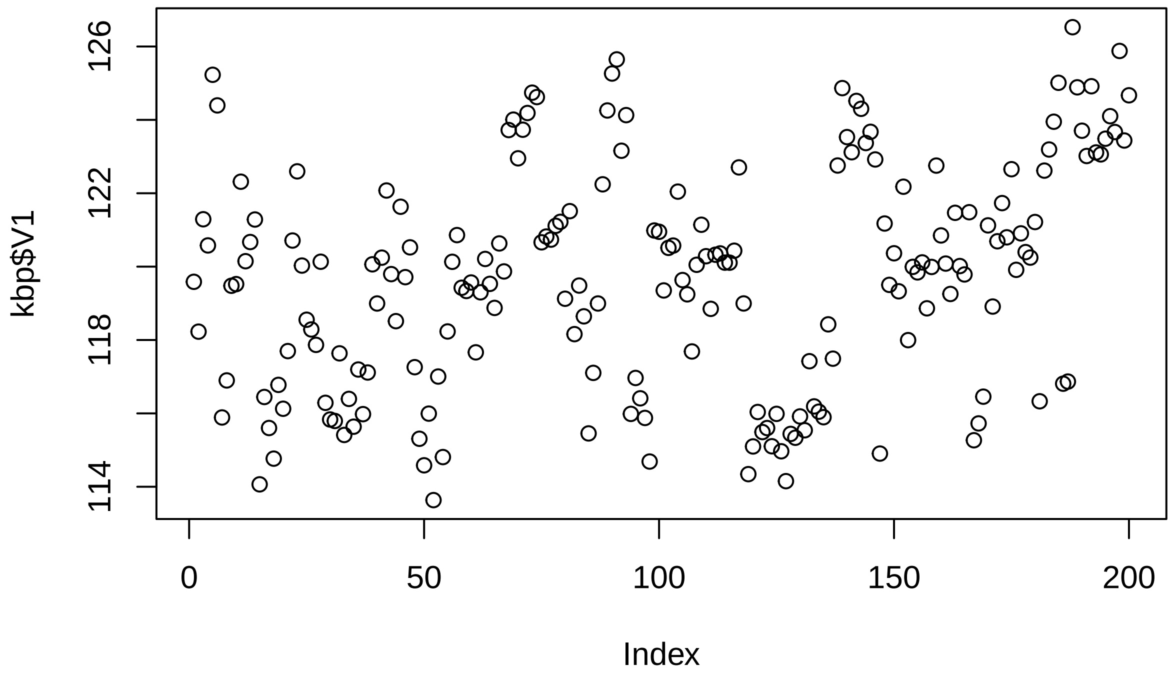
Dongyang Wang

12/11/2021

Intuition

1

```
#rm(list =ls())
kbp <- read.csv('SBP.csv', header = F)
#kbp
plot(kbp$V1)
```



```
v1 <- kbp$V1

sum1 = 0
sum2 = 0
sum3 = 0

attach(kbp)
level_122 <- sum(V1 > 122)
level_118 <- sum(V1 < 118)
level_bet <- length(V1) - level_118 - level_122
level_122
```

```
## [1] 49
```

```
level_118
```

```
## [1] 62
```

```
level_bet
```

```
## [1] 89
```

```
detach(kbp)

for(i in 1:200){
  if (v1[i] < 118){
    sum1 = sum1 +v1[i]
  }
  else if (v1[i] > 122){
    sum2 = sum2 +v1[i]
  }
  else {
    sum3 = sum3 +v1[i]
  }
}

c(sum1/level_118, sum3/level_bet, sum2/level_122)
```

```
## [1] 116.0365 120.0315 123.7530
```

```
# For simplicity, I will use 116, 120, 124.
```

By observation, there are approximately three levels separated by 2 thresholds, 122 and 118. No, the initial guesses of 1,2,3 do not seem to be appropriate. My guesses are 116.0365 120.0315 123.7530.

2

If transition probabilities are all equal, three states will have approximately the same number of observations. if p_{ii} is smaller, we would observe a lot of shifts–ups and downs in the graph, i.e., changing states.

```
tpm <- matrix(c(rep(0,9)), nrow = 3, byrow = T)

for(i in 1:(length(v1)-1)){
  if (v1[i] < 118){
    if(v1[i+1] <118){
      tpm[1,1] = tpm[1,1] +1
    }
    else if (v1[i+1] > 122){
      tpm[1,3] = tpm[1,3] +1
    }
    else {
      tpm[1,2] = tpm[1,2] +1
    }
  }
  else if (v1[i] > 122){
    if(v1[i+1] <118){
      tpm[3,1] = tpm[3,1] +1
    }
    else if (v1[i+1] > 122){
      tpm[3,3] = tpm[3,3] +1
    }
    else {
      tpm[3,2] = tpm[3,2] +1
    }
  }
  else {
    if(v1[i+1] <118){
      tpm[2,1] = tpm[2,1] +1
    }
    else if (v1[i+1] > 122){
      tpm[2,3] = tpm[2,3] +1
    }
    else {
      tpm[2,2] = tpm[2,2] +1
    }
  }
}

#check correct: missing one entry as expected
sum(tpm)
```

```
## [1] 199
```

```
# After checking, only the third row needs one more entry. We deduct it from
# original to normalize it, making the row sum equal to 1.

tpm[1,] = tpm[1,]/level_118
tpm[2,] = tpm[2,]/level_bet
tpm[3,] = tpm[3,]/(level_122-1)

tpm
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.7419355 0.2096774 0.0483871
## [2,] 0.1235955 0.7528090 0.1235955
## [3,] 0.1041667 0.1666667 0.7291667
```

The guess for transition probabilities is the matrix listed above.

Likelihood evaluataion

1

Here, we know that Q is as defined above with the matrix tpm. Initial distribution is uniform. And emission probability is normal with my guesses for means as the means, and standard deviation of 1. With these information, we can code the following algorithm.

```
# backward

#initialize
beta = list()
input = c(0,0,0)
for (i in 1:200){
  beta[[i]] = input
}

backward_log <- function(v, Q, mu, y){
   for (i in 199:1){
     for (k in 1:3){
      maxi = max(Q[k,1] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +  beta[[i+1]][1]  ,
                 Q[k,2] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +  beta[[i+1]][2],
                 Q[k,3] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +  beta[[i+1]][3])
      #print(maxi)
      beta[[i]][k] = maxi + log (exp(Q[k,1] - log(2*pi)  - (y[i+1] - mu[k])^2/2  +
                                       beta[[i+1]][1] - maxi)  +
                                   exp(Q[k,2] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +
                                         beta[[i+1]][2] - maxi) +
                                   exp(Q[k,3]  - log(2*pi)  - (y[i+1] - mu[k])^2/2 +
                                         beta[[i+1]][3] - maxi))
     }
     #print(beta[[i]])
   }
   maximum <- max(v[1] - log(2*pi)  - (y[1] - mu[k])^2/2 + beta[[1]][1]  ,
                  v[2] - log(2*pi)  - (y[1] - mu[k])^2/2 +  beta[[1]][2] ,
                  v[3] - log(2*pi)  - (y[1] - mu[k])^2/2 +  beta[[1]][3])
   beta_0 =  maximum + log( exp(v[1] - log(2*pi)  - (y[1] - mu[k])^2/2 +
                                  beta[[1]][1] - maximum)
                            + exp(v[2] - log(2*pi)  - (y[1] - mu[k])^2/2 +
                                    beta[[1]][2] - maximum)
                            + exp(v[3] - log(2*pi)  - (y[1] - mu[k])^2/2 +
                                    beta[[1]][3] - maximum) )
   return(beta_0)
 }


# forward

#initialize
alpha = list()
input = c(0,0,0)
for (i in 1:200){
  alpha[[i]] = input
}

forward_log <- function(v, Q, mu, y){
  for (i in 1:3){
  alpha[[1]][i] = v[i] - log(2*pi)- (y[1] - mu[i])^2/2
}
  for (i in 1:199){
    for(k in 1:3){
      maxi = max(alpha[[i]][1] + Q[1,k] ,
                 alpha[[i]][2] + Q[2,k] ,
                 alpha[[i]][3]+ Q[3,k])
      alpha[[i+1]][k] = - log(2*pi)- (y[i+1] - mu[k])^2/2 +maxi
      + log(exp(alpha[[i]][1] + Q[1,k] - maxi)
            + exp(alpha[[i]][2] + Q[2,k] - maxi)
            + exp(alpha[[i]][3] + Q[3,k] - maxi) )
    }
    #print(alpha[[i]])
  }
  alpha_0 = alpha[[200]][1] + alpha[[200]][2] + alpha[[200]][3]
  return(alpha_0)
}
```

2

```
logQ = log(tpm)
logini_pdf = log(c(1/3,1/3,1/3))
#logv = log(c(116.0365,120.0315,123.7530))
mu =c(116.0365,120.0315,123.7530)
backward_log(logini_pdf, logQ, mu, v1)
```

```
## [1] -590.4762
```

```
forward_log(logini_pdf, logQ, mu, v1)
```

```
## [1] -1812.669
```

```
# change to a more concentrated tpm on the diagonal
tpm1 <- matrix(c(0.8,0.1,0.1,0.1,0.8,0.1,0.1,0.1,0.8), nrow = 3)

backward_log(logini_pdf, log(tpm1), mu, v1)
```

```
## [1] -586.8742
```

```
forward_log(logini_pdf, log(tpm1), mu, v1)
```

```
## [1] -1796.019
```

```
# change mean
mu1 <- c(118,120,122)
backward_log(logini_pdf, logQ, mu1, v1)
```

```
## [1] -768.5192
```

```
forward_log(logini_pdf, logQ, mu1, v1)
```

```
## [1] -2382.469
```

We have obtained the negative log likelihood, with backward log probability -590.4762 and forward log probability -1797.43. By adjusting the parameters, we see that a more concentrated tpm has higher likelihood. A less accurate mean will lead to lower likelihood.

Parameters estimation

EM

The updated backward and forward algorithms are as follows to return all an and bn.

```
# first rewrite the backward and forward likelihood algos to return all at,bt

# backward

#initialize
beta = list()
input = c(0,0,0)
for (i in 1:200){
  beta[[i]] = input
}
```

```
backward <- function(v, Q, mu, y){
   for (i in 199:1){
     for (k in 1:3){
      maxi = max(Q[k,1] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +  beta[[i+1]][1],
                 Q[k,2] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +  beta[[i+1]][2],
                 Q[k,3] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +  beta[[i+1]][3])
      #print(maxi)
      beta[[i]][k] = maxi + log (exp(Q[k,1] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +
                                  beta[[i+1]][1] - maxi)
                              + exp(Q[k,2] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +
                                    beta[[i+1]][2] - maxi)
                              + exp(Q[k,3] - log(2*pi)  - (y[i+1] - mu[k])^2/2 +
                                    beta[[i+1]][3] - maxi))
     }
     #print(beta[[i]])
   }
   maximum <- max(v[1] - log(2*pi)  - (y[1] - mu[k])^2/2 + beta[[1]][1]  ,
                  v[2] - log(2*pi)  - (y[1] - mu[k])^2/2 +  beta[[1]][2] ,
                  v[3] - log(2*pi)  - (y[1] - mu[k])^2/2 +  beta[[1]][3])
   beta_0 =  maximum + log(
     exp(v[1] - log(2*pi)  - (y[1] - mu[k])^2/2 + beta[[1]][1] - maximum)  +
     exp(v[2] - log(2*pi)  - (y[1] - mu[k])^2/2 + beta[[1]][2] - maximum) +
     exp(v[3] - log(2*pi)  - (y[1] - mu[k])^2/2 + beta[[1]][3] - maximum)
     )
   return(beta)
 }


# forward

#initialize
alpha = list()
input = c(0,0,0)
for (i in 1:200){
  alpha[[i]] = input
}

forward <- function(v, Q, mu, y){
  for (i in 1:3){
  alpha[[1]][i] = v[i] - log(2*pi)- (y[1] - mu[i])^2/2
}
  for (i in 1:199){
    for(k in 1:3){
      maxi = max(alpha[[i]][1] + Q[1,k] ,
                 alpha[[i]][2] + Q[2,k] ,
                 alpha[[i]][3]+ Q[3,k])
      alpha[[i+1]][k] = - log(2*pi)- (y[i+1] - mu[k])^2/2 +maxi +
      log(exp(alpha[[i]][1] + Q[1,k] - maxi) +
           exp(alpha[[i]][2] + Q[2,k] - maxi)  +
           exp(alpha[[i]][3] + Q[3,k] - maxi) )
    }
    #print(alpha[[i]])
  }
  alpha_0 = alpha[[200]][1] + alpha[[200]][2] + alpha[[200]][3]
  return(alpha)
}

#example
#backward(logini_pdf, logQ, mu, v1)
#forward(logini_pdf, logQ, mu, v1)
```

The setup of the algorithm is as follows.

```
# For simplicity, I will run this algorithm 100 times,
# and we will observe from graph that likelihood converges.

# set up gamma
gamma = list()
input = c(0,0,0)
for (i in 1:200){
  gamma[[i]] = input
}

# set up gt
gt = list()
input = c(0,0,0)
for (i in 1:200){
  gt[[i]] = list(input, input, input)
}

# store MLE
mle_vector <- c()
```

The algorithm is as follows.

```
# Get started
em <- function(v, Q, mu, y){
  for(i in 1:3){
    # use forward & backward likelihood to calculate gamma
    alpha = forward(v, Q, mu, y)
    beta = backward(v, Q, mu, y)
    #print(head(alpha))
    # gamma
    for (j in 1:200){
      for (k in 1:3){
        gamma[[j]][k] = alpha[[j]][k] + beta[[j]][k] -
            log(exp(alpha[[j]][1] + beta[[j]][1]) +
                exp(alpha[[j]][2] + beta[[j]][2]) +
                exp(alpha[[j]][3] + beta[[j]][3]))
      }
    }
    #print(head(gamma))
    # update initial prob
    v[1] = gamma[[1]][1]
    v[2] = gamma[[1]][2]
    v[3] = gamma[[1]][3]
    # gt
    for ( q in 2:200){
```

```
    denominator1 = 0
    summation = 0
    #for (ii in 1:3){
      #for (jj in 1:3){
        #maxi =  max(maxi, beta[[q]][jj] - log(2*pi)  -
          #(y[q] - mu[jj])^2/2 + Q[ii,jj] + alpha[[q-1]][ii])
      #}
    #}
    #for (ii in 1:3){
      #for (jj in 1:3){
        #helper = exp(beta[[q]][jj] - log(2*pi)  -
          #(y[q] - mu[jj])^2/2 + Q[ii,jj] + alpha[[q-1]][ii] - maxi)
        #denominator1 = denominator1 + helper
      #}
    #}
    #denominator1 = maxi + log(denominator1)
    for (ii in 1:3){
      for (jj in 1:3){
        summation = summation + exp(beta[[q]][jj]) * exp(- log(2*pi)  -
          (y[q] - mu[jj])^2/2)  * exp(Q[ii,jj]) * exp(alpha[[q-1]][ii])
      }
    }
    #print(summation)
    denominator1 = log(summation)
    #print(denominator1)
    for (w in 1:3){
      for (e in 1:3){
        numerator1 = beta[[q]][e] - log(2*pi)  - (y[q] - mu[e])^2/2 + Q[w,e] + alpha[[q-1]][w]
        gt[[q]][[w]][e] = numerator1 - denominator1
      }
    }
  }
  #print(head(gt))
  # update tpm
  for (j in 1:3){
    for (k in 1:3){
      numerator2 = 0
      denominator2 = 0
      for (kk in 2:200){
        numerator2 = numerator2 + exp(gt[[kk]][[j]][k])
      }
      for (jk in 1:199){
        denominator2 = denominator2 +  exp(gamma[[jk]][j])
      }
      Q[j,k] = log(numerator2) - log(denominator2)
  }
  }
  #print(Q)
  # update e
  for (j in 1:3){
    numerator3 = 0
    denominator3 = 0
    for (k in 1:200){
        numerator3 = numerator3 + gamma[[k]][j]*y[k]
        denominator3 = denominator3 + gamma[[k]][j]
      }
      mu[j] = numerator3/denominator3
    }
    print(mu)
    # calculate MLE for comparison
    mle_vector[i] = backward_log(v, Q, mu, y)
  }

  return(mle_vector)
}

em(logini_pdf, logQ, mu, v1)
```

```
## [1] 122.4209 119.1956 117.1805
## [1] 116.7098 120.6695 122.6148
## [1] 122.6064 118.0016 116.9277
```

```
## [1] -690.2262 -645.1368 -739.7698
```

Since the second log likelihood has been highest, and there is decreasing trend afterwards, we can conclude that the 2nd update is optimal. So we report the mu to be updated to 116.7098, 120.6695, and 122.6148.

Hidden state inference

Viterbi

```
#initialize
di = list()
input = c(0,0,0)
for (i in 1:200){
  di[[i]] = input
}

fi = list()
input = c(0,0,0)
for (i in 1:200){
  fi[[i]] = input
}

xn = list()
for (i in 1:200){
  xn[[i]] = c(0)
}

viterbi <- function(v, Q, mu, y){
  for(k in 1:3){
    di[[1]][k] = v[k] - log(2*pi)- (y[1] - mu[k])^2/2
    #print(di[[1]])
  }
  for (i in 2:200){
    for (j in 1:3){
      maximum = max(di[[i-1]][1]+ Q[1,j], di[[i-1]][2]+ Q[2,j], di[[i-1]][3]+ Q[3,j])
      di[[i]][j] = - log(2*pi)- (y[i] - mu[j])^2/2 + maximum
      if (di[[i-1]][1] + Q[1,j] == maximum){
        fi[[i]][j] = 1
      }
      else if (di[[i-1]][2] + Q[2,j] == maximum){
        fi[[i]][j] = 2
      }
      else{
        fi[[i]][j] = 3
      }
    }
  }
  #print(di)
  maximum1 = max(di[[200]][1],di[[200]][2],di[[200]][3])
  if (di[[200]][1] == maximum1){
        xn[[200]] =1
      }
      else if (di[[200]][2] == maximum1){
        xn[[200]] = 2
      }
      else{
        xn[[200]] = 3
      }
  for (t in 199:1){
      maximum2 = max(di[[t]][1],di[[t]][2],di[[t]][3])
      if (di[[t]][1] == maximum2){
        xn[[t]] =1
      }
      else if (di[[t]][2] == maximum2){
        xn[[t]] = 2
      }
      else{
        xn[[t]] = 3
      }
  }
  return(xn)
}

predict_state <- viterbi(logini_pdf, logQ, mu, v1)
states_pred <- unlist(predict_state)
states_pred
```

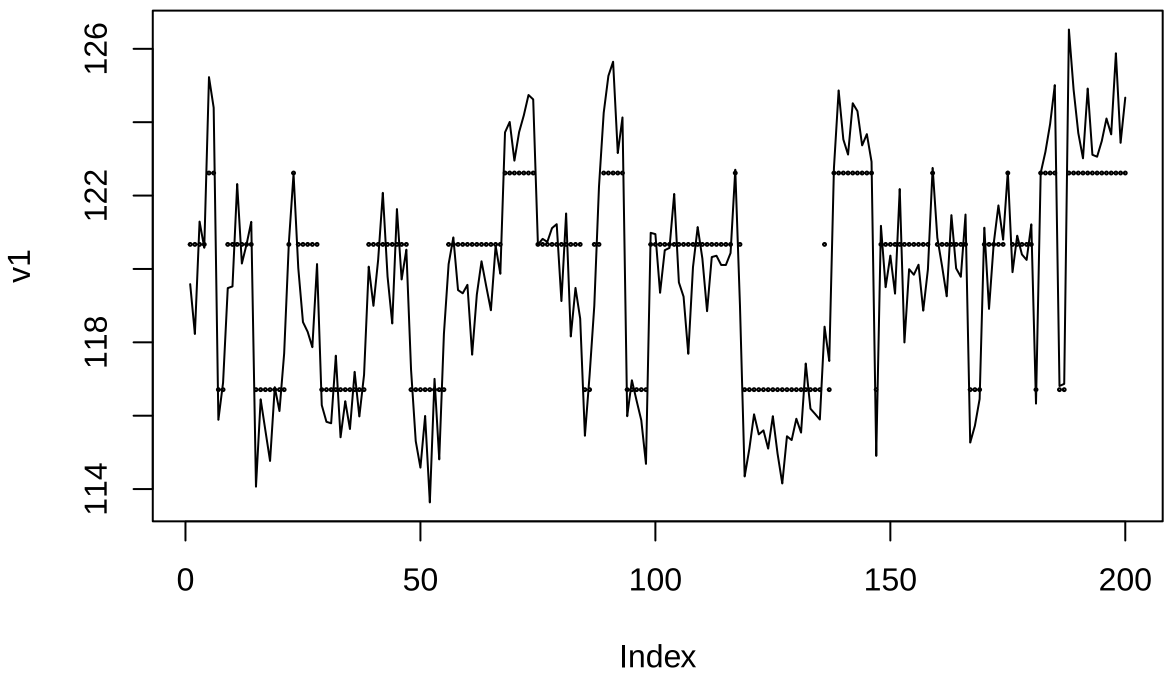
```
##   [1] 2 2 2 2 3 3 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 2 3 2 2 2 2 2 1 1 1 1 1 1 1 1 1
##  [38] 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3
##  [75] 2 2 2 2 2 2 2 2 2 2 1 1 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 3 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 3 3 3 3 3 3 3 3 3 1 2
## [149] 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 3 2 2 2 2 2 1 3 3 3 3
## [186] 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3
```

Estimation of hidden states & Plots

With the result from EM Algorithm we can obtain the updated mu's for emission probability. Also from the Viterbi Algorithm we have the predicted states. Therefore, we can draw a graph with the data and their predicted means. But not that this update does not seem to fit perfectly with the data, not even as good as out guess, which I attached below the first graph.

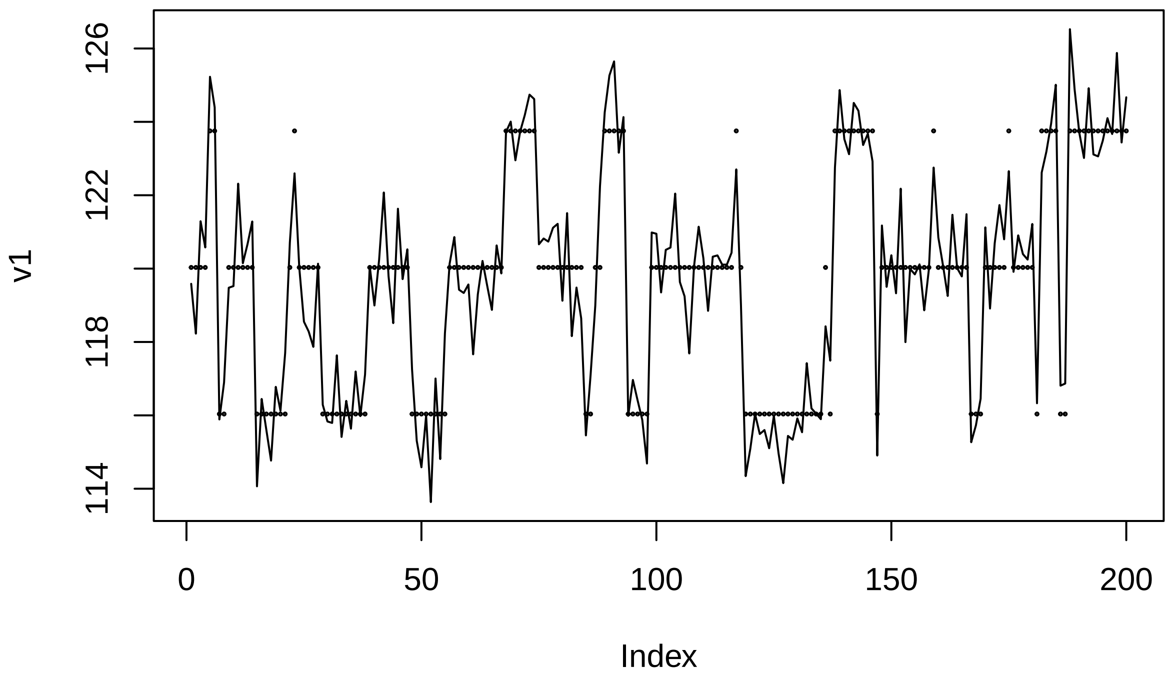
```
means = c()
predicted <- c(116.7098, 120.6695, 122.6148)
for (i in 1:200){
  if (states_pred[i] == 1){
    means[i] = predicted[1]
  }
  else if (states_pred[i] == 2){
    means[i] = predicted[2]
  }
  else {
    means[i] = predicted[3]
  }
}

plot(v1, type = "l", lty = 1);points(means, cex =0.2)
```



```
predicted <- c(116.0365, 120.0315, 123.7530)
for (i in 1:200){
  if (states_pred[i] == 1){
    means[i] = predicted[1]
  }
  else if (states_pred[i] == 2){
    means[i] = predicted[2]
  }
  else {
    means[i] = predicted[3]
  }
}

plot(v1, type = "l", lty = 1);points(means, cex =0.2)
```



Forecasting

1

From Ch.16-2, we can observe that the formula for calculating the forecasting probability is $P(Y_{n+h} = y_{n+h}|y) = \frac{\sum_i \{\sum_j e(y_{n+h|j}) p^{(h)}(j|i)\} a_n(i)}{\sum_k a_n(k)}$. Here, we simply apply $e(y_{n+h|j})$ with our normal distribution, which in the logrithm case, is $-log(2\pi) - \frac{(y_[i]_{n+h} - \mu_j)^2}{2}$.

2

The algorithm should aim at calculating a confidence interval for the data. To do this, since I have already had an expression for the pdf, I would start with the previous observation's number, i.e., the observation on day n. After this, since I know that the tpm concentrates on the diagonal, I would say predictions about the future are dependent on today's state. Therefore, I would start with $y_{n+h} = y_n|y$, and move to both directions for an interval. I would say, for each direction, I increase/decrease at a rate of 0.001 and add up all probabilities until I got a 0.95. Then the confidence interval can be obtained by reporting numbers from two ends. Admittedly, it is possible that the maximum or minimum are reached, so we can add an if-else statement that lets the interval expand only on the other end.