516-Final

Chris Chen

12/13/2021

Preliminaries

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(expm)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## Attaching package: 'expm'
```

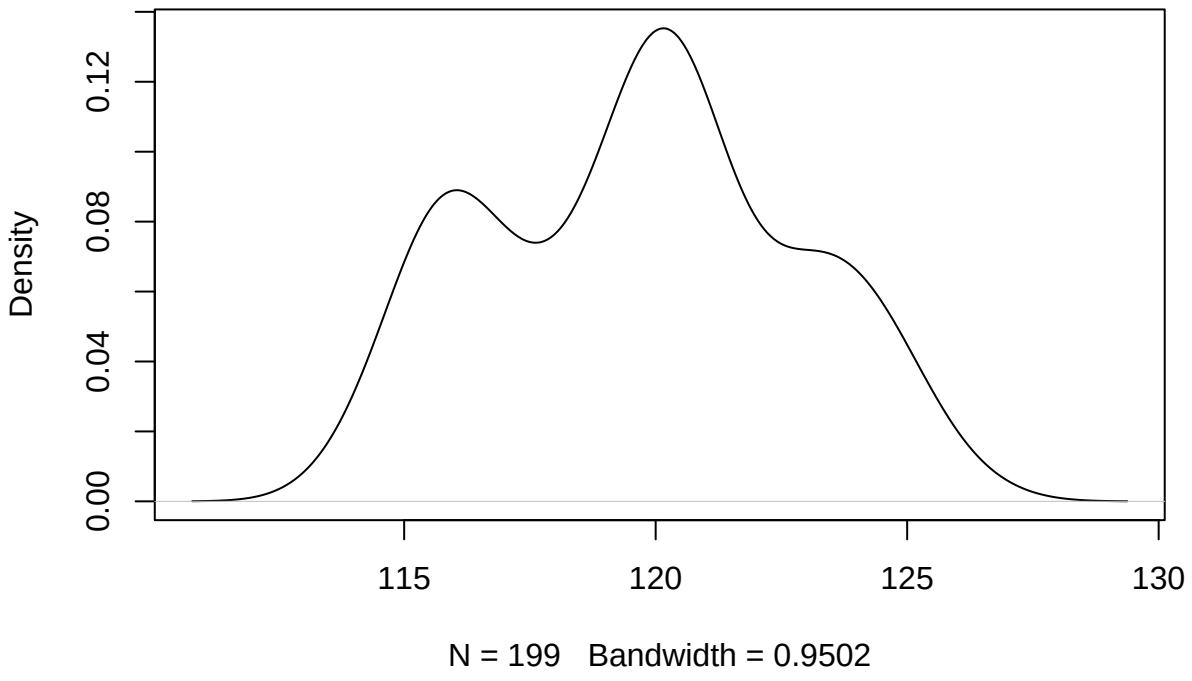
```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```
library(ggplot2)
```

Analyzing personal health data

1. Initial Means

```
sbp = read.csv("SBP.csv")
y = sbp$X1.195862488340358567e.02
plot(density(y))
```




```
set.seed(516)
Guessings = function(y, c) {
  clus = kmeans(y, c)$cluster
  mus = rep(0, c)
  sigmas = rep(0, c)
  pis = rep(0, c)
  for (i in 1:c) {
    mus[i] = mean(y[clus == i])
    sigmas[i] = sd(y[clus == i])
    pis[i] = sum(clus == i) / length(clus)
  }
  return(cbind(mus, sigmas, pis))
}
means = Guessings(y, 3)[, 1]; means
```

```
## [1] 123.7530 120.0365 116.0365
```

```
sigmas = Guessings(y, 3)[, 2]; sigmas
```

```
## [1] 0.8924601 1.0209579 1.0616325
```

I don't think the means will be equal. I think an initial guess of $\mu_1 = 124$, $\mu_2 = 120$, and $\mu_3 = 116$ would be a good choice. The initial guesses of $\sigma's$ are all 1. This is good based on the results above.

2. Transition Probabilities

If the transition probabilities were all equal, then we would observe the three peaks in the above density plot to be equally high. If pii were to be smaller than the pij for $j \neq i$, then the density plot should have wide peaks (large variance).

```r
pi = Guessings(y, 3)[, 3]
sbp <- sbp %>%
  mutate(state = kmeans(y, 3)$cluster)
colnames(sbp) = c("y", "state")
n = length(y)
```

```r
p11 = p12 = p13 = p21 = p22 = p23 = p31 = p32 = p33 = 0
for (i in 2:n) {
  prev = sbp$state[i - 1]
  cur = sbp$state[i]
  if (prev == 1 & cur == 1) {
    p11 = p11 + 1
  } else if (prev == 1 & cur == 2) {
    p12 = p12 + 1
  } else if (prev == 1 & cur == 3) {
    p13 = p13 + 1
  } else if (prev == 2 & cur == 1) {
    p21 = p21 + 1
  } else if (prev == 2 & cur == 2) {
    p22 = p22 + 1
  } else if (prev == 2 & cur == 3) {
    p23 = p23 + 1
  } else if (prev == 3 & cur == 1) {
    p31 = p31 + 1
  } else if (prev == 3 & cur == 2) {
    p32 = p32 + 1
  } else {
    p33 = p33 + 1
  }
}
pij = matrix(nrow = 3, ncol = 3)
pij[1, ] = c(p11, p12, p13) / (p11 + p12 + p13)
pij[2, ] = c(p21, p22, p23) / (p21 + p22 + p23)
pij[3, ] = c(p31, p32, p33) / (p31 + p32 + p33)
pij
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.7291667 0.1666667 0.1041667
## [2,] 0.1250000 0.7500000 0.1250000
## [3,] 0.0483871 0.2096774 0.7419355
```

```r
P = pij
```

The initial guess of the tpm is shown above.

3. Initial distribution

```r
unif_ini = function(s) {
  return(rep(1 / s, s))
}
v0 = unif_ini(3); v0
```

```
## [1] 0.3333333 0.3333333 0.3333333
```

We do not have more than one realization of the HMM so we simply assume the initial distribution is uniform.

Coding Part

Changing representation of parameters

```
Q = log(P)
v = log(v0)
```

Likelihood Evaluation

1. Backward algorithm and Forward algorithm

```
backward_prob = function(y, log_ini, log_tpm, means, s, n) {
  log_ems = function(y, x) {
    log(dnorm(y, mean = means[x], sd = 1))
  }
  bs = matrix(nrow = s, ncol = n)
  bs[, n] = 0
  for (t in (n-1):1) {
    sum = rep(0, s)
    for (i in 1:s) {
      sum[i] = 0
      for (j in 1:s) {
        sum[i] = sum[i] + exp(log_tpm[i, j] + log_ems(y[t + 1], j) + bs[j, t + 1])
      }
    }
    bs[, t] = log(sum)
  }
  return(bs)
}
```

```
forward_prob = function(y, log_ini, log_tpm, means, s, n) {
  log_ems = function(y, x) {
    log(dnorm(y, mean = means[x], sd = 1))
  }
  as = matrix(nrow = s, ncol = n)
  for (i in 1:s) {
    as[i, 1] = log_ini[i] + log_ems(y[1], i)
  }
  for (t in 1:(n-1)) {
    for (i in 1:s) {
      temp_sum = 0
      for (j in 1:s) {
        temp_sum = temp_sum + exp(log_tpm[j, i] + as[j, t])
      }
      as[i, t + 1] = log_ems(y[t + 1], i) + log(temp_sum)
    }
  }
  return(as)
}
```

2. Negative log-likelihood

```
neg_logs = function(y, log_ini, log_tpm, means, s, n) {
  log_ems = function(y, x) {
    log(dnorm(y, mean = means[x], sd = 1))
  }
  b0 = 0
  asum = 0
  bs = backward_prob(y, log_ini, log_tpm, means, s, n)
  as = forward_prob(y, log_ini, log_tpm, means, s, n)
  for (i in 1:s) {
    b0 = b0 + exp(log_ini[i] + log_ems(y[1], i) + bs[i, 1])
    asum = asum + exp(as[i, n])
  }
  return(c(-log(b0), -log(asum)))
}
neg_logs(y, v, Q, means, 3, 199)
```

```
## [1] 397.217 397.217
```

Parameter Estimation

An important helper function for the rest few alogorithms

```
log_ems = function(y, x, mius) {
  log(dnorm(y, mean = mius[x], sd = 1))
}
```

EM (Baum-Welch) Algorithm

```
EM = function(y, theta_0, s, epsilon = 1e-6) {

  n = length(y)
  vmat = matrix(nrow = s, ncol = 1000)
  vmat[, 1] = theta_0[, 1]
  v_ori = matrix(nrow = s, ncol = 1000)
  Qmat = matrix(nrow = s, ncol = s * 1000)
  Qmat[, 1:s] = theta_0[, 2:(1+s)]
  miumat = matrix(nrow = s, ncol = 1000)
  miumat[, 1] = theta_0[, (2+s)]

  lgammat = function(i, t, ltheta_k) {
    as = forward_prob(y, ltheta_k[, 1], ltheta_k[, 2:(1+s)], ltheta_k[, (2+s)], s, n)
    bs = backward_prob(y, ltheta_k[, 1], ltheta_k[, 2:(1+s)], ltheta_k[, (2+s)], s, n)
    sum = 0
    for (j in 1:s) {
      sum = sum + exp(as[j, t] + bs[j, t])
    }
    return(as[i, t] + bs[i, t] - log(sum))
  }
```

```
lgt = function(i, j, t, ltheta_k) {
  Q = ltheta_k[, 2:(1+s)]
  as = forward_prob(y, ltheta_k[, 1], Q, ltheta_k[, (2+s)], s, n)
  bs = backward_prob(y, ltheta_k[, 1], Q, ltheta_k[, (2+s)], s, n)
  sum = 0
  for (ind1 in 1:s) {
    for (ind2 in 1:s) {
      sum = sum + exp(bs[ind2, t] + log_ems(y[t], ind2, ltheta_k[, (2+s)]) + Q[ind1, ind2] +
                      as[ind1, t - 1])
    }
  }
  return(bs[j, t] + log_ems(y[t], j, ltheta_k[, (2+s)]) + Q[i, j] + as[i, t - 1] - log(sum))
}

getLtheta = function(index) {
  start = (index - 1) * s + 1
  end = (index - 1) * s + s
  return(cbind(vmat[, index], Qmat[, start:end], miumat[, index]))
}

getLtheta2 = function(index) {
  start = (index - 1) * s + 1
  end = (index - 1) * s + s
  for (i in 1:s) {
    v_ori[i, index] = exp(vmat[i, index])
  }
  return(cbind(v_ori[, index], Qmat[, start:end], miumat[, index]))
}

for (indi in 1:s) {
  vmat[indi, 2] = lgammat(indi, 1, getLtheta(1))

  sum_temp_1 = 0
  sum_temp_2 = 0
  for (t1 in 1:n) {
    sum_temp_1 = sum_temp_1 + exp(lgammat(indi, t1, getLtheta(1)) + log(y[t1]))
    sum_temp_2 = sum_temp_2 + exp(lgammat(indi, t1, getLtheta(1)))
  }
  miumat[indi, 2] = sum_temp_1 / sum_temp_2

  for (indj in 1:s) {
    sum_temp_3 = 0
    sum_temp_4 = 0
    for (t2 in 2:n) {
      sum_temp_3 = sum_temp_3 + exp(lgt(indi, indj, t2, getLtheta(1)))
      sum_temp_4 = sum_temp_4 + exp(lgammat(indi, t2 - 1, getLtheta(1)))
    }
    Qmat[indi, s + indj] = log(sum_temp_3) - log(sum_temp_4)
  }
}

k = 2
for (i in 1:2) {
```

```
    for (j in 1:s) {
      v_ori[j, i] = exp(vmat[j, i])
    }
  }
  while (norm(getLtheta2(k) - getLtheta2(k - 1), type = "F") >= epsilon) {
    as = forward_prob(y, getLtheta(k)[, 1], getLtheta(k)[, 2:(1+s)], getLtheta(k)[, (2+s)], s, n)
    bs = backward_prob(y, getLtheta(k)[, 1], getLtheta(k)[, 2:(1+s)], getLtheta(k)[, (2+s)], s, n)
    for (indi in 1:s) {
      vmat[indi, k + 1] = lgammat(indi, 1, getLtheta(k))

      sum_temp_1 = 0
      sum_temp_2 = 0
      for (t1 in 1:n) {
        sum_temp_1 = sum_temp_1 + exp(lgammat(indi, t1, getLtheta(k)) + log(y[t1]))
        sum_temp_2 = sum_temp_2 + exp(lgammat(indi, t1, getLtheta(k)))
      }
      miumat[indi, k + 1] = sum_temp_1 / sum_temp_2

      for (indj in 1:s) {
        sum_temp_3 = 0
        sum_temp_4 = 0
        for (t2 in 2:n) {
          sum_temp_3 = sum_temp_3 + exp(lgt(indi, indj, t2, getLtheta(k)))
          sum_temp_4 = sum_temp_4 + exp(lgammat(indi, t2 - 1, getLtheta(k)))
        }
        Qmat[indi, s * k + indj] = log(sum_temp_3) - log(sum_temp_4)
      }
    }
    if (k >= 1000) {
      break
    }
    k = k + 1
  }
  mles = getLtheta(min(k, 1000))
  v2 = mles[, 1]
  v02 = exp(v2)
  Q2 = mles[, 2:(1+s)]
  pij2 = exp(Q2)
  means2 = mles[, (2+s)]
  return(cbind(v02, pij2, means2))
}
theta_0 = cbind(v, Q, means)
theta_k = EM(y, theta_0, 3)
v2 = theta_k[, 1]
pij2 = theta_k[, 2:4]
means2 = theta_k[, 5]
theta_k
```

```
##              v02                                    means2
## [1,] 3.954674e-78 0.87247898 0.02636834 0.10115268 124.0260
## [2,] 1.000000e+00 0.03061280 0.89617934 0.07320786 120.1642
## [3,] 1.752194e-11 0.05116163 0.13871895 0.81011942 115.9741
```

The MLEs are reported above. It seems that we are getting closer to the truth.

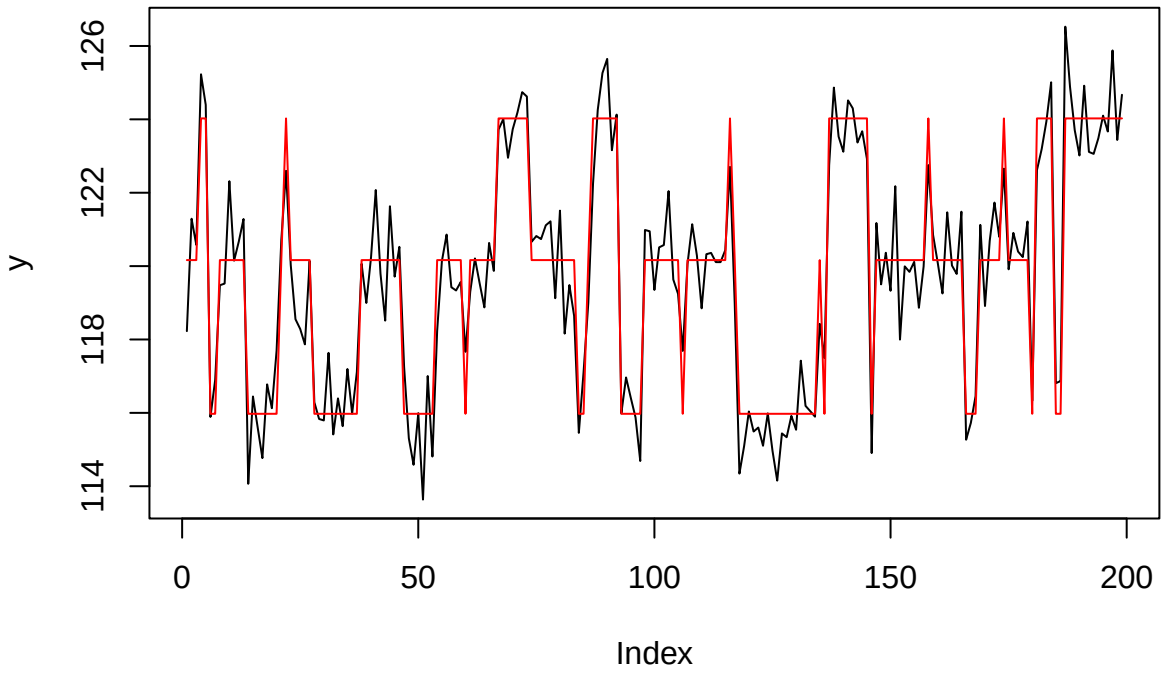
Hidden State Inference

1. Viterbi Algorithm

```
viterbi = function(y, theta, s) {
  n = length(y)
  v2 = theta[, 1]
  pij2 = theta[, 2:(1+s)]
  means2 = theta[, (2+s)]
  dmat = matrix(nrow = s, ncol = n)
  fmat = matrix(nrow = s, ncol = n)
  xt = rep(0, n)
  for (i in 1:s) {
    dmat[i, 1] = v2[i] + log_ems(y[1], i, means2)
  }
  fmat[, 1] = 0
  for (t1 in 2:n) {
    for (i in 1:s) {
      dmat[i, t1] = log_ems(y[t1], i, means2) + max(dmat[, t1 - 1] + pij2[, i])
      fmat[i, t1] = which.max(dmat[, t1 - 1] + pij2[, i])
    }
  }
  xt[n] = which.max(dmat[, n])
  for (t2 in (n-1):1) {
    xt[t2] = fmat[xt[t2 + 1], t2 + 1]
  }
  return(xt)
}
```

2. Estimating and plotting the hidden states

```
xt = viterbi(y, theta_k, 3)
plot(y, type = 'l')
lines(means2[xt], col = "red")
```



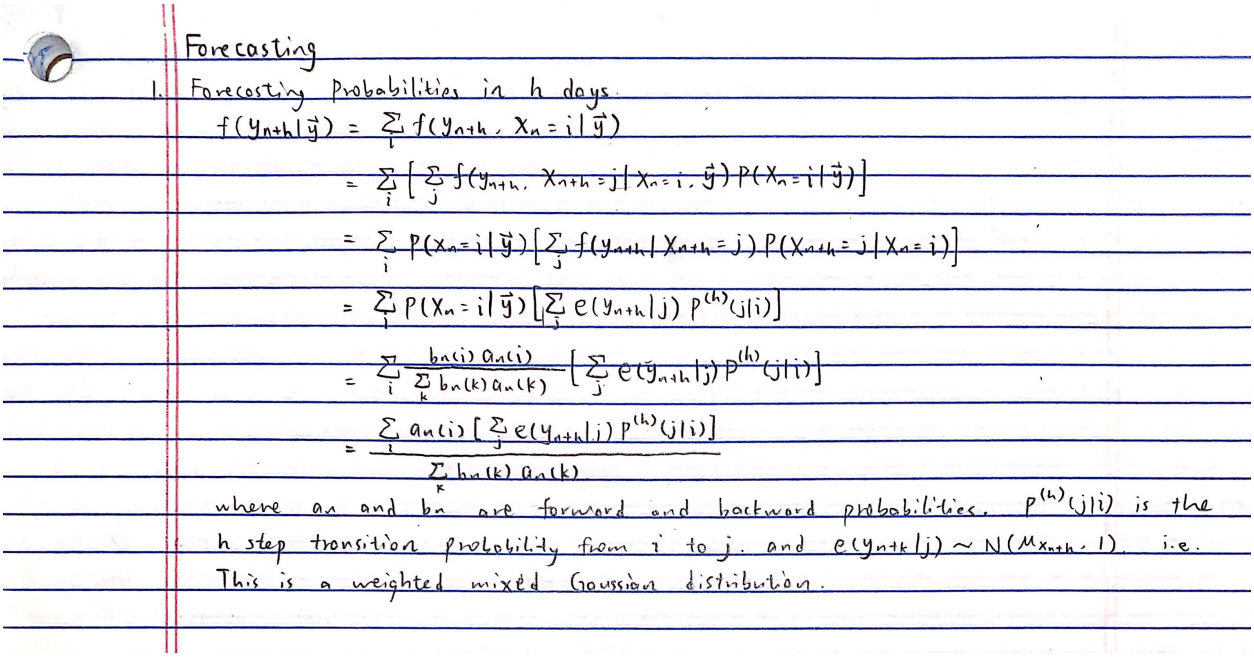
The most probable path is constructed and superimposed on the original data. Beautifully done.

Forecasting

A function for calculating n-step transition probability.

```
pijn = function(n, i, j, P) {
  s = ncol(P)
  if (n == 0) {
    if (i == j) {
      return(1)
    } else {
      return(0)
    }
  } else if (n == 1) {
    return(P[i, j])
  } else {
    sum = 0
    for (k in 1:s) {
      sum = sum + pijn(n - 1, i, k, P) * P[k, j]
    }
    return(sum)
  }
}
```

1. Forecasting probabilities for the observation in h days



Forecasting

1. Forecasting probabilities in h days.

$$f(y_{n+h}|\vec{y}) = \sum_i f(y_{n+h}, X_n = i|\vec{y})$$

$$= \sum_i \left[\sum_j f(y_{n+h}, X_{n+h} = j | X_n = i, \vec{y}) P(X_n = i|\vec{y})\right]$$

$$= \sum_i P(X_n = i|\vec{y}) \left[\sum_j f(y_{n+h}|X_{n+h} = j) P(X_{n+h} = j | X_n = i)\right]$$

$$= \sum_i P(X_n = i|\vec{y}) \left[\sum_j e(y_{n+h}|j)\, p^{(h)}(j|i)\right]$$

$$= \sum_i \frac{b_n(i)\alpha_n(i)}{\sum_k b_n(k)\alpha_n(k)} \left[\sum_j e(y_{n+h}|j) p^{(h)}(j|i)\right]$$

$$= \frac{\sum_i \alpha_n(i)\left[\sum_j e(y_{n+h}|j) p^{(h)}(j|i)\right]}{\sum_k b_n(k)\alpha_n(k)}$$

where α_n and b_n are forward and backward probabilities. $p^{(h)}(j|i)$ is the h step transition probability from i to j. and $e(y_{n+k}|j) \sim N(\mu_{x_{n+h}}, 1)$. i.e. This is a weighted mixed Gaussian distribution.

The distribution of $f(y_{n+h}|\mathbf{y})$. h default is set to 1, corresponding to the question "what would be the SBP tomorrow".

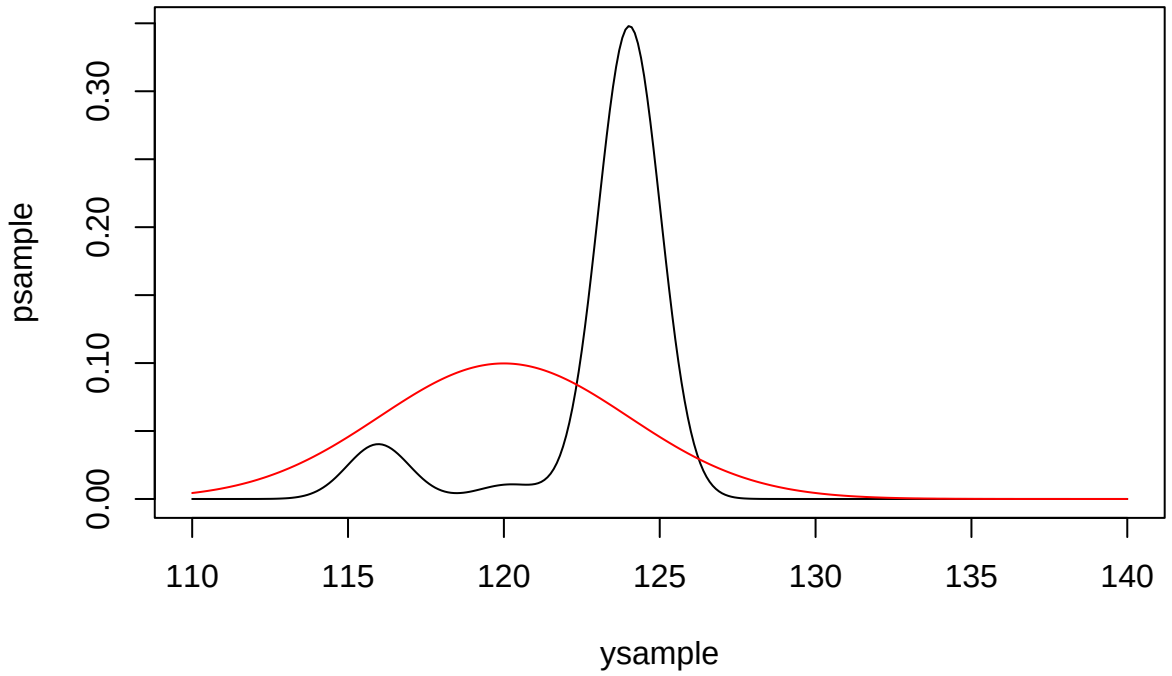
```
forecasting_dist = function(yf, y, P, h = 1) {
  n = length(y)
  s = ncol(P)
  as = forward_prob(y, v2, pij2, means2, s, n)
  sum_temp_j = 0
  sum_temp_i = 0
  sum_temp_k = 0
  for (i in 1:s) {
    for (j in 1:s) {
      sum_temp_j = sum_temp_j + exp(log_ems(yf, j, means2)) * pijn(h, i, j, P)
    }
    sum_temp_i = sum_temp_i + exp(log(sum_temp_j) + as[i, n])
  }
  for (k in 1:s) {
    sum_temp_k = sum_temp_k + exp(as[k, n])
  }
  return(sum_temp_i / sum_temp_k)
}
```

2. and 3. Proposal and Implementation of algorithm

I propose to use rejection sampling to sample from this distribution since although we have its pdf, we do not know how to sample from it directly. Then we can make a probabilistic statement on what will be the SBP tomorrow by giving a 95% confidence interval using bootstrapping.

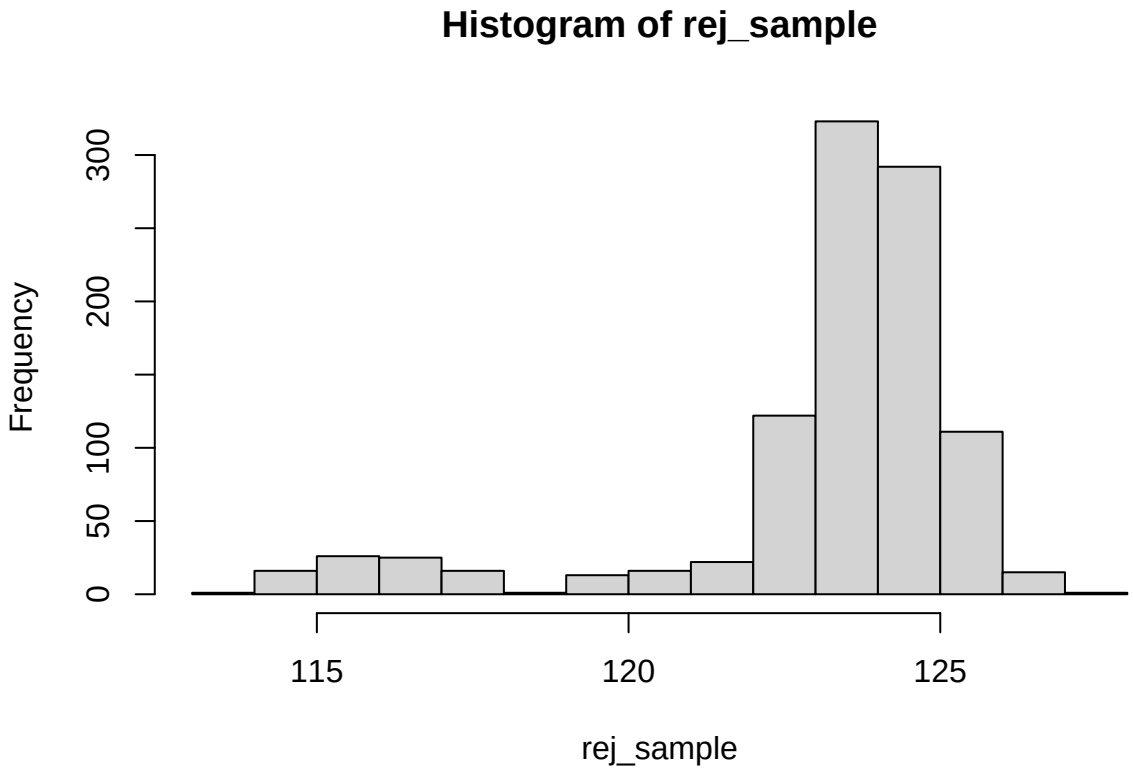
```
ysample = seq(110, 140, by = 0.1)
psample = forecasting_dist(ysample, y, pij2)
```

```r
plot(x = ysample, y = psample, type = "l")
lines(x = ysample, y = dnorm(ysample, mean = 120, sd = 4), col = "red")
```



Note that the distribution looks very different from the plot of the original data. This is because we are predicting on what will be the SBP on day 200, not just any day. Note that on day 199, we are in state 1, and by checking the MLE_P we can see that the probability of staying in state 1 is 0.8725. That's why the distribution is very tall at ~124, i.e. state 1 on day 200.

```r
M = max(psample / dnorm(ysample, mean = 120, sd = 4))
rejection_sample = function(M, n) {
  S = c()
  while (length(S) <= n) {
    ydraw = rnorm(1, 120, 4)
    u = runif(1)
    if (u <= forecasting_dist(ydraw, y, pij2) / (M * dnorm(ydraw, 120, 4))) {
      S = append(S, ydraw)
    }
  }
  return(S)
}
rej_sample = rejection_sample(M, 999)
hist(rej_sample)
```



The rejection sampling works well. The sample looks exactly like the distribution.

Now we use bootstrapping to construct a confidence interval

```
sample_matrix = matrix(nrow = 1000, ncol = 1000)
sample_matrix[, 1] = rej_sample
for (i in 2:1000) {
  sample_matrix[, i] = sample(sample_matrix[, 1], 1000, replace = TRUE)
}
mles = colMeans(sample_matrix)
CI = quantile(mles, probs = c(0.025, 0.975))
```

A 95% Confidence Interval for tomorrow's SBP is [123.0016988, 123.3119849].

Model Selection

Helper functions for calculating p(s) and generating initial guesses

```
param = function(s) {
  return(s * (s - 1) + s)
}

theta_generator = function(s, P) {
  v = unif_ini(s)
  means = Guessings(y, s)[, 1]
  theta = cbind(v, P, means)
  return(theta)
}
```

1. Computing AIC and the BIC for different values of the number of states s

Looking at the plot of the original data, it's reasonable to guess that the Hidden Markov Chain have 3 or 2 states (if we ignore the little bump on the right). We can try 4 states. However, due to the fact that it takes five hundred years to run the EM, we stop at 4 and do not go to 5 or 6 or 7 states.

s = 2

```
sbp2 <- sbp %>%
  mutate(state = kmeans(y, 2)$cluster)
colnames(sbp2) = c("y", "state")

p11 = p12 = p21 = p22 = 0
for (i in 2:n) {
  prev = sbp2$state[i - 1]
  cur = sbp2$state[i]
  if (prev == 1 & cur == 1) {
    p11 = p11 + 1
  } else if (prev == 1 & cur == 2) {
    p12 = p12 + 1
  } else if (prev == 2 & cur == 1) {
    p21 = p21 + 1
  } else {
    p22 = p22 + 1
  }
}
pij = matrix(nrow = 2, ncol = 2)
pij[1, ] = c(p11, p12) / (p11 + p12)
pij[2, ] = c(p21, p22) / (p21 + p22)
pij
```

```
##           [,1]      [,2]
## [1,] 0.8083333 0.1916667
## [2,] 0.3076923 0.6923077
```

```
P = pij
```

```
theta_k2 = EM(y, theta_generator(2, P), 2)
```

```
as2 = neg_logs(y, log(theta_k2[, 1]), log(theta_k2[, 2:3]), theta_k2[, 4], 2, n)[1]; as2
```

```
## [1] 560.9234
```

```
AIC2 = 2 * as2 + 2 * param(2); AIC2
```

```
## [1] 1129.847
```

```
BIC2 = 2 * as2 + log(n) * param(2); BIC2
```

```
## [1] 1143.02
```

s = 3

```
as3 = neg_logs(y, log(v2), log(pij2), means2, 3, n)[1]; as3
```

```
## [1] 384.0006
```

```
AIC3 = 2 * as3 + 2 * param(3); AIC3
```

```
## [1] 786.0012
```

```
BIC3 = 2 * as3 + log(n) * param(3); BIC3
```

```
## [1] 815.6409
```

s = 4

```
sbp4 <- sbp %>%
  mutate(state = kmeans(y, 4)$cluster)
colnames(sbp4) = c("y", "state")

p11 = p12 = p13 = p14 =
  p21 = p22 = p23 = p24 =
  p31 = p32 = p33 = p34 =
  p41 = p42 = p43 = p44 = 0
for (i in 2:n) {
  prev = sbp4$state[i - 1]
  cur = sbp4$state[i]
  if (prev == 1 & cur == 1) {
    p11 = p11 + 1
  } else if (prev == 1 & cur == 2) {
    p12 = p12 + 1
  } else if (prev == 1 & cur == 3) {
    p13 = p13 + 1
  } else if (prev == 1 & cur == 4) {
    p14 = p14 + 1
  } else if (prev == 2 & cur == 1) {
    p21 = p21 + 1
  } else if (prev == 2 & cur == 2) {
    p22 = p22 + 1
  } else if (prev == 2 & cur == 3) {
    p23 = p23 + 1
  } else if (prev == 2 & cur == 4) {
    p24 = p24 + 1
  } else if (prev == 3 & cur == 1) {
    p31 = p31 + 1
  } else if (prev == 3 & cur == 2) {
    p32 = p32 + 1
  } else if (prev == 3 & cur == 3) {
    p33 = p33 + 1
  } else if (prev == 3 & cur == 4) {
    p34 = p34 + 1
  } else if (prev == 4 & cur == 1) {
    p41 = p41 + 1
  } else if (prev == 4 & cur == 2) {
    p42 = p42 + 1
  } else if (prev == 4 & cur == 3) {
    p43 = p43 + 1
  } else {
```

```
    p44 = p44 + 1
  }
}
pij = matrix(nrow = 4, ncol = 4)
pij[1, ] = c(p11, p12, p13, p14) / (p11 + p12 + p13 + p14)
pij[2, ] = c(p21, p22, p23, p24) / (p21 + p22 + p23 + p24)
pij[3, ] = c(p31, p32, p33, p34) / (p31 + p32 + p33 + p34)
pij[4, ] = c(p41, p42, p43, p44) / (p41 + p42 + p43 + p44)
pij
```

```
##           [,1]       [,2]       [,3]       [,4]
## [1,] 0.5483871 0.27419355 0.11290323 0.06451613
## [2,] 0.4871795 0.33333333 0.05128205 0.12820513
## [3,] 0.1136364 0.02272727 0.77272727 0.09090909
## [4,] 0.0754717 0.13207547 0.03773585 0.75471698
```

```
P = pij
```

I commented out this section because it takes hours to run and now I do not have enough time to knit the file with this section included.

```
# theta_k4 = EM(y, theta_generator(4, P), 4)
```

I got the result the first time but I did not record the exact numbers. But I expect the negative log to be smaller than s = 3 and AIC and BIC to be higher.

```
# as4 = neg_logs(y, log(theta_k4[, 1]), log(theta_k4[, 2:5]), theta_k4[, 6], 4, n)[1]; as4
# AIC4 = 2 * as4 + 2 * param(4); AIC4
# BIC4 = 2 * as4 + log(n) * param(4); BIC4
```

2. Conclusion

As shown, negative log likelihood is constantly decreasing–that is expected since more states fits the data more closely and hence making the likelihood larger. AIC and BIC are the lowest at state = 3, so 3 is at least a local minimum. Higher AIC and BIC when states are greater than 3 means the model is over-fitting. But at 2 the accuracy is low, meaning the model is under-fitting.

Combining the information we have from the data, I believe 3 could be the global minimum. Therefore, I think the best model is HMM with 3 states, and hence the best possible set of parameters is the following:

```
theta_k
```

```
##              v02                                      means2
## [1,] 3.954674e-78 0.87247898 0.02636834 0.10115268 124.0260
## [2,] 1.000000e+00 0.03061280 0.89617934 0.07320786 120.1642
## [3,] 1.752194e-11 0.05116163 0.13871895 0.81011942 115.9741
```