Lab3

Christian von Koch, Josefin Bladh, William Anzen, Anton Gefvert

2020-09-23

Assignment 1 - Christian von Koch

```
set.seed(12345)
transitionMatrix=diag(0.5, 10)
diag(transitionMatrix[,-1])=0.5
transitionMatrix[10,1]=0.5
emissionMatrix=matrix(0,10,10)
for(i in 1:10) {
  for (j in 1:10) {
    if((j+7-i) %% 10 >= 5) {
      emissionMatrix[i,j]=0.2
    } else {
      emissionMatrix[i,j]=0
    }
  }
}
emissionMatrix
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]  0.2  0.2  0.2  0.0  0.0  0.0  0.0  0.0  0.2   0.2
##  [2,]  0.2  0.2  0.2  0.2  0.0  0.0  0.0  0.0  0.0   0.2
##  [3,]  0.2  0.2  0.2  0.2  0.2  0.0  0.0  0.0  0.0   0.0
##  [4,]  0.0  0.2  0.2  0.2  0.2  0.2  0.0  0.0  0.0   0.0
##  [5,]  0.0  0.0  0.2  0.2  0.2  0.2  0.2  0.0  0.0   0.0
##  [6,]  0.0  0.0  0.0  0.2  0.2  0.2  0.2  0.2  0.0   0.0
##  [7,]  0.0  0.0  0.0  0.0  0.2  0.2  0.2  0.2  0.2   0.0
##  [8,]  0.0  0.0  0.0  0.0  0.0  0.2  0.2  0.2  0.2   0.2
##  [9,]  0.2  0.0  0.0  0.0  0.0  0.0  0.2  0.2  0.2   0.2
## [10,]  0.2  0.2  0.0  0.0  0.0  0.0  0.0  0.2  0.2   0.2
```

```
states=1:10
symbols=1:10
HMM_model=initHMM(States=states, Symbols=symbols, transProbs=transitionMatrix, emissionProbs=emissionMa
HMM_model
```

```
## $States
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $Symbols
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $startProbs
##   1   2   3   4   5   6   7   8   9  10
## 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
##
## $transProbs
##     to
## from   1   2   3   4   5   6   7   8   9  10
##   1  0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   2  0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##   3  0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0 0.0
##   4  0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0 0.0
##   5  0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0 0.0
##   6  0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0 0.0
##   7  0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0 0.0
##   8  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5 0.0
##   9  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5 0.5
##   10 0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.5
##
## $emissionProbs
##       symbols
## states   1   2   3   4   5   6   7   8   9  10
##     1  0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2
##     2  0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2
##     3  0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0
##     4  0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0
##     5  0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0 0.0
##     6  0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0 0.0
##     7  0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.0
##     8  0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2 0.2
##     9  0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2 0.2
##     10 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.2 0.2 0.2
```

Assignment 2 - Christian von Koch

```r
simulation=simHMM(HMM_model, length=100)
```

Assignment 3 - William Anzen

```r
smoothingFunc <- function(hmm, obs, n){
  alpha <- exp(forward(hmm,obs))
  beta <- exp(backward(hmm,obs))
  smoothing = matrix(0,10,n)
  for(j in 1:n){
    smoothing[,j] <- beta[,j]*alpha[,j]/sum(beta[,j]*alpha[,j])
  }
  return(smoothing)
}
filteringFunc <- function(hmm, obs,n){
  alpha <- exp(forward(hmm,obs))
  filtering = matrix(0,10,n)
  for(j in 1:n){
    filtering[,j] <- alpha[,j]/sum(alpha[,j])
  }
  return(filtering)
}
smoothing = smoothingFunc(HMM_model,simulation$observation,100)
filtering = filteringFunc(HMM_model,simulation$observation,100)
probPath <- viterbi(HMM_model,simulation$observation)
probPath
```

```
##   [1]  8  9 10  1  1  1  1  1  2  2  3  3  3  3  3  4  4  4  4  5  6  7  8  9 10
##  [26]  1  1  1  1  1  2  3  3  3  4  4  4  5  5  6  7  8  9 10  1  1  1  1  2  3
##  [51]  4  4  4  5  6  7  7  8  8  8  9 10 10 10 10  1  1  1  1  2  2  2  3  3  3
##  [76]  3  3  4  5  6  7  8  8  8  8  8  9 10  1  1  1  1  1  1  1  1  1  1  2  2
```

As shown above the most probable path is shown.

Assignment 4 - William Anzen

```r
smoothPath <- apply(smoothing, MARGIN=2,FUN=which.max)
filterPath <- apply(filtering, MARGIN = 2, FUN=which.max)
accuracyFunc <- function(data, inputdata){
  n=length(inputdata)
  nTrue <- sum(data == inputdata)
  return(nTrue/n)
}
probPathAcc <- accuracyFunc(probPath,simulation$states)
smoothPathAcc <- accuracyFunc(smoothPath, simulation$states)
filterPathAcc <- accuracyFunc(filterPath, simulation$states)
probPathAcc
```

```
## [1] 0.56
```

```r
smoothPathAcc
```

```
## [1] 0.74
```

```r
probPathAcc
```

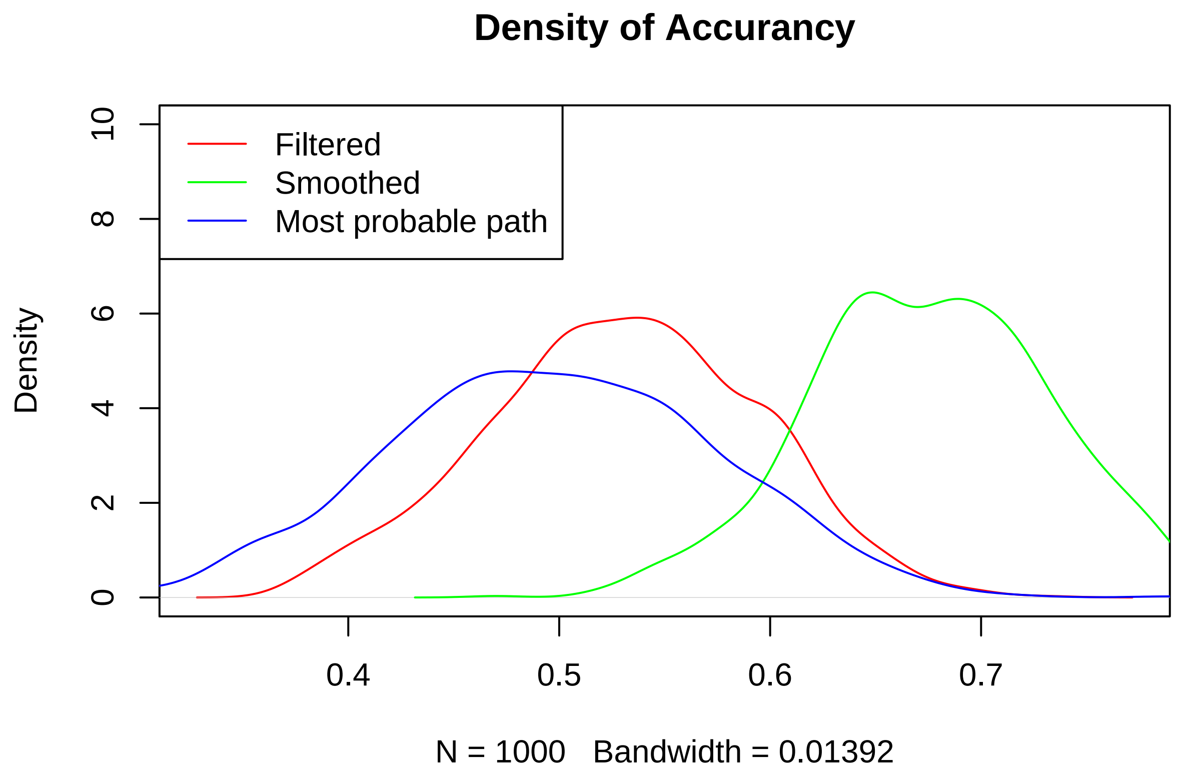
```
## [1] 0.56
```

We see that the smoothing probability has the highest accuracy and the most probable path has the lowest accuracy. The filtering has a accuracy quite close to the most probable path.

Assignment 5 - Josefin Bladh

```r
accuracyFunc2=function(sim,hmm,char, n){
  n=length(sim$observation)
  if(char=="F"){
    filtering=filteringFunc(HMM_model,sim$observation, n)
    path=apply(filtering,2,which.max)
  }else if (char=="S"){
    smoothing=smoothingFunc(HMM_model,sim$observation, n)
    path=apply(smoothing,2,which.max)
  }else if (char=="P"){
    path=viterbi(HMM_model,sim$observation)
  }
  return(accuracyFunc(sim$states,path))
}

simulatedAcc=matrix(0,1000,3)
for (i in 1:1000){
 simulation=simHMM(HMM_model,100)
 simulatedAcc[i,1]=accuracyFunc2(simulation,HMM_model,"F")
 simulatedAcc[i,2]=accuracyFunc2(simulation,HMM_model,"S")
 simulatedAcc[i,3]=accuracyFunc2(simulation,HMM_model,"P")
}
plot(density(simulatedAcc[,1]),col="red", main="Density of Accurancy", ylim=c(0,10))
lines(density(simulatedAcc[,2]),col="green")
lines(density(simulatedAcc[,3]),col="blue")
legend("topleft", box.lty = 1, legend = c("Filtered","Smoothed","Most probable path"),
 col=c("red","green","blue"), lwd = 1)
```



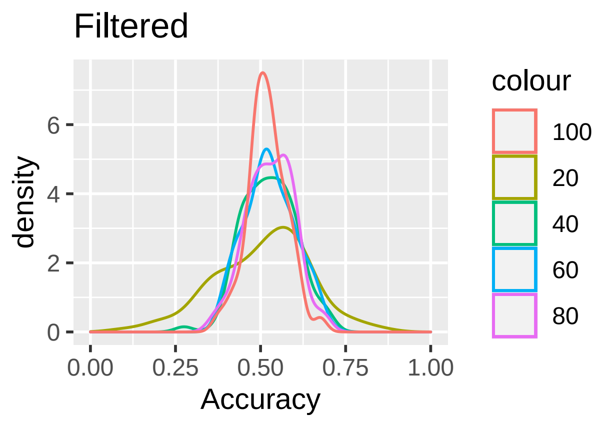
The smoothed distribution should be more accurate than the filtered distribution as the smoothing conditions on all the data but the filtering only conditions on the data up until each timepoint. The smoothed uses both forward and backward but the filtering uses only forward.

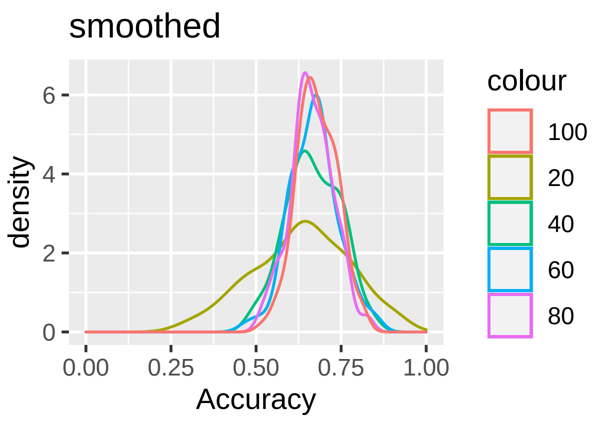
The Viterbi algorithm calculates the most probable sequence taking the neighbors in consideration, but the smoothing maximizes the marginal probability for each individual timestep and does not take sequence in consideration.

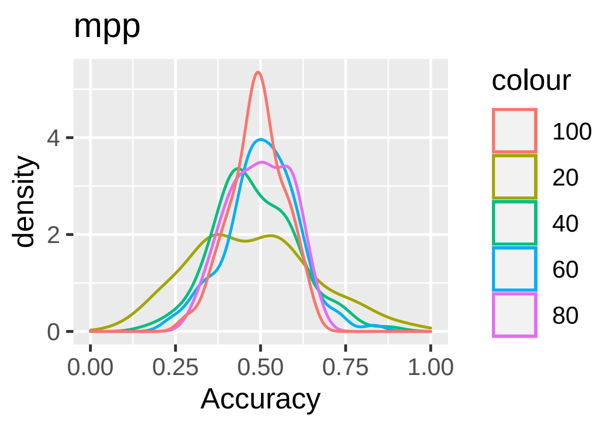
```r
calculate_accuracies = function(hmm, steps=100){
  simulated = simHMM(hmm, steps)
  observed = simulated$observation
  hidden = simulated$states
  alpha = exp(forward(hmm, observed))
  beta = exp(backward(hmm, observed))
  filtered = filteringFunc(hmm, observed, steps)
  smoothed = smoothingFunc(hmm, observed, steps)
  most_probable_path = viterbi(hmm, observed)
  filtered.most_probable = apply(filtered, 2, which.max)
  smoothed.most_probable = apply(smoothed, 2, which.max)
  filtered.acc = sum(filtered.most_probable == hidden) / ncol(filtered)
  smoothed.acc = sum(smoothed.most_probable == hidden) / ncol(smoothed)
  most_probable_path.acc = sum(most_probable_path == hidden) / length(most_probable_path)
  return(matrix(c(
    filtered.acc,
    smoothed.acc,
    most_probable_path.acc),
    nrow=1,ncol=3))
}

set.seed(1234566)
runs = 100
accuracies_3d = array(dim=c(5, runs, 3))
obs_num = c(20, 40, 60, 80, 100)
for (o in 1:5){
  for (i in 1:runs){
    obs = obs_num[o]
    accuracies_3d[o,i,] = as.array(calculate_accuracies(HMM_model, obs))
  }
}
```

We then plot the densities for each observation number, for each algorithm.





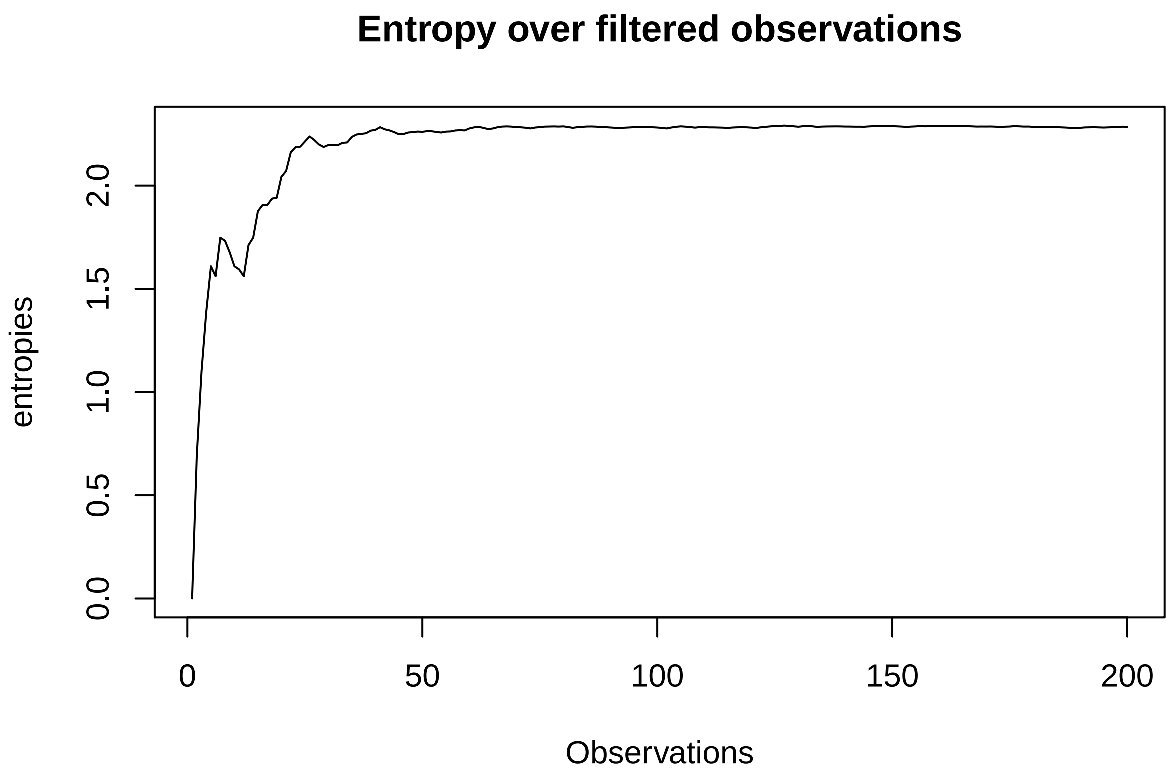


In the figure above, we see that more observations doesn't increase our accuracy for the different metrics, we only get less variance. Since our accuracy does not increase with more observations, we *cannot* say that we better know where the robot is.

If we look at the suggested *entropy.empirical* function, we can run a HMM with 200 observations, and look the most probable points from 1 to i, where i goes from 1 to 200, and plot the entropy as i increases.

```r
simulated = simHMM(HMM_model, 200)
observed = simulated$observation
hidden = simulated$states
filtered = filteringFunc(HMM_model, observed, 200)
smoothed = smoothingFunc(HMM_model, observed, 200)
filtered.most_probable = apply(filtered, 2, which.max)
entropies = rep(0, 200)
for(i in 1:200){
  entropies[i] = entropy.empirical(table(factor(filtered.most_probable[1:i], 1:10)))
}

plot(1:200, entropies, type="l", main="Entropy over filtered observations", xlab="Observations")
```



As we see in the graph above, the entropy increases until around 40 observations, where it stagnates. Since the entropy never decreases, we can see that more observations does not increase our knowledge of where the robot is.

Assignment 7 - Josefin Bladh

```
prob101=as.vector(filtering[,100]%*%transitionMatrix)
prob101
```

```
##  [1] 0.0000 0.1875 0.5000 0.3125 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Using the observations in time step 100 and the transformation matrix will give us the probabilities of the hidden states for time step 101 as seen above.

Appendix

```
# 1

set.seed(12345)
transitionMatrix=diag(0.5, 10)
diag(transitionMatrix[,-1])=0.5
transitionMatrix[10,1]=0.5
emissionMatrix=matrix(0,10,10)
for(i in 1:10) {
  for (j in 1:10) {
    if((j+7-i) %% 10 >= 5) {
      emissionMatrix[i,j]=0.2
    } else {
      emissionMatrix[i,j]=0
    }
  }
}
emissionMatrix
states=1:10
symbols=1:10
HMM_model=initHMM(States=states, Symbols=symbols, transProbs=transitionMatrix, emissionProbs=emissionMa
HMM_model

# 2

simulation=simHMM(HMM_model, length=100)

# 3

smoothingFunc <- function(hmm, obs, n){
  alpha <- exp(forward(hmm,obs))
  beta <- exp(backward(hmm,obs))
  smoothing = matrix(0,10,n)
  for(j in 1:n){
    smoothing[,j] <- beta[,j]*alpha[,j]/sum(beta[,j]*alpha[,j])
  }
  return(smoothing)
}
filteringFunc <- function(hmm, obs,n){
  alpha <- exp(forward(hmm,obs))
  filtering = matrix(0,10,n)
  for(j in 1:n){
    filtering[,j] <- alpha[,j]/sum(alpha[,j])
  }
  return(filtering)
}
smoothing = smoothingFunc(HMM_model,simulation$observation,100)
filtering = filteringFunc(HMM_model,simulation$observation,100)
probPath <- viterbi(HMM_model,simulation$observation)
probPath

# 4
```

```r
smoothPath <- apply(smoothing, MARGIN=2,FUN=which.max)
filterPath <- apply(filtering, MARGIN = 2, FUN=which.max)
accuracyFunc <- function(data, inputdata){
  n=length(inputdata)
  nTrue <- sum(data == inputdata)
  return(nTrue/n)
}
probPathAcc <- accuracyFunc(probPath,simulation$states)
smoothPathAcc <- accuracyFunc(smoothPath, simulation$states)
filterPathAcc <- accuracyFunc(filterPath, simulation$states)
probPathAcc
smoothPathAcc
probPathAcc

# 5

accuracyFunc2=function(sim,hmm,char, n){
  n=length(sim$observation)
  if(char=="F"){
    filtering=filteringFunc(HMM_model,sim$observation, n)
    path=apply(filtering,2,which.max)
  }else if (char=="S"){
    smoothing=smoothingFunc(HMM_model,sim$observation, n)
    path=apply(smoothing,2,which.max)
  }else if (char=="P"){
    path=viterbi(HMM_model,sim$observation)
  }
  return(accuracyFunc(sim$states,path))
}

simulatedAcc=matrix(0,1000,3)
for (i in 1:1000){
 simulation=simHMM(HMM_model,100)
 simulatedAcc[i,1]=accuracyFunc2(simulation,HMM_model,"F")
 simulatedAcc[i,2]=accuracyFunc2(simulation,HMM_model,"S")
 simulatedAcc[i,3]=accuracyFunc2(simulation,HMM_model,"P")
}
plot(density(simulatedAcc[,1]),col="red", main="Density of Accurancy", ylim=c(0,10))
lines(density(simulatedAcc[,2]),col="green")
lines(density(simulatedAcc[,3]),col="blue")
legend("topleft", box.lty = 1, legend = c("Filtered","Smoothed","Most probable path"),
 col=c("red","green","blue"), lwd = 1)

# 6

calculate_accuracies = function(hmm, steps=100){
  simulated = simHMM(hmm, steps)
  observed = simulated$observation
  hidden = simulated$states
  alpha = exp(forward(hmm, observed))
  beta = exp(backward(hmm, observed))
  filtered = filteringFunc(hmm, observed, steps)
  smoothed = smoothingFunc(hmm, observed, steps)
  most_probable_path = viterbi(hmm, observed)
  filtered.most_probable = apply(filtered, 2, which.max)
  smoothed.most_probable = apply(smoothed, 2, which.max)
  filtered.acc = sum(filtered.most_probable == hidden) / ncol(filtered)
  smoothed.acc = sum(smoothed.most_probable == hidden) / ncol(smoothed)
  most_probable_path.acc = sum(most_probable_path == hidden) / length(most_probable_path)
  return(matrix(c(
    filtered.acc,
    smoothed.acc,
    most_probable_path.acc),
    nrow=1,ncol=3))
}

set.seed(1234566)
runs = 100
accuracies_3d = array(dim=c(5, runs, 3))
obs_num = c(20, 40, 60, 80, 100)
for (o in 1:5){
  for (i in 1:runs){
    obs = obs_num[o]
    accuracies_3d[o,i,] = as.array(calculate_accuracies(HMM_model, obs))
  }
}

df_filtered = data.frame("20"=accuracies_3d[1,,1],
                         "40"=accuracies_3d[2,,1],
                         "60"=accuracies_3d[3,,1],
                         "80"=accuracies_3d[4,,1],
                         "100"=accuracies_3d[5,,1])
filtered_plot = ggplot(df_filtered, aes(x=Accuracy)) +
  geom_density(aes(x=X20, col='20')) +
  geom_density(aes(x=X40, col='40')) +
  geom_density(aes(x=X60, col='60')) +
  geom_density(aes(x=X80, col='80')) +
  geom_density(aes(x=X100, col='100')) +
  ggtitle("Filtered") + xlim(0,1)

df_smoothed  = data.frame("20"=accuracies_3d[1,,2],
                         "40"=accuracies_3d[2,,2],
                         "60"=accuracies_3d[3,,2],
                         "80"=accuracies_3d[4,,2],
                         "100"=accuracies_3d[5,,2])
smoothed_plot = ggplot(df_smoothed, aes(x=Accuracy)) +
  geom_density(aes(x=X20, col='20')) +
  geom_density(aes(x=X40, col='40')) +
  geom_density(aes(x=X60, col='60')) +
  geom_density(aes(x=X80, col='80')) +
  geom_density(aes(x=X100, col='100')) +
  ggtitle("smoothed") + xlim(0,1)

df_mpp  = data.frame("20"=accuracies_3d[1,,3],
                     "40"=accuracies_3d[2,,3],
                     "60"=accuracies_3d[3,,3],
                     "80"=accuracies_3d[4,,3],
                     "100"=accuracies_3d[5,,3])
mpp_plot = ggplot(df_mpp, aes(x=Accuracy)) +
  geom_density(aes(x=X20, col='20')) +
  geom_density(aes(x=X40, col='40')) +
  geom_density(aes(x=X60, col='60')) +
  geom_density(aes(x=X80, col='80')) +
  geom_density(aes(x=X100, col='100')) +
  ggtitle("mpp") + xlim(0,1)

grid.arrange(filtered_plot, smoothed_plot, mpp_plot, ncol=2)

simulated = simHMM(HMM_model, 200)
observed = simulated$observation
hidden = simulated$states
filtered = filteringFunc(HMM_model, observed, 200)
smoothed = smoothingFunc(HMM_model, observed, 200)
filtered.most_probable = apply(filtered, 2, which.max)
entropies = rep(0, 200)
for(i in 1:200){
  entropies[i] = entropy.empirical(table(factor(filtered.most_probable[1:i], 1:10)))
}

plot(1:200, entropies, type="l", main="Entropy over filtered observations", xlab="Observations")

# 7

prob101=as.vector(filtering[,100]%*%transitionMatrix)
prob101
```