Advanced Machine Learning
LAB 2: HIDDEN MARKOV MODELS

Mohammed Bakheet (mohba508)

23/09/2020

Contents

The Purpose:

The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to model the behavior of a robot that walks around a ring. The ring is divided into 10 sectors. At any given time point, the robot is in one of the sectors and decides with equal probability to stay in that sector or move to the next sector. You do not have direct observation of the robot. However, the robot is equipped with a tracking device that you can access. The device is not very accurate though: If the robot is in the sector i, then the device will report that the robot is in the sectors [i-2, i+2] with equal probability.

Question 1:

Build a hidden Markov model (HMM) for the scenario described above.

```
# Initialise HMM
states_vec = c(1:10)
symbols_vec = LETTERS[seq( from = 1, to = 10 )]
start_prob = rep(0.1,10)

#The rebot can stay in the current sector or move to the next sector with equal probability.
trans_probs=matrix(data = 0, nrow = length(states_vec), ncol = length(states_vec))
diag(trans_probs) = 0.5
for (i in 1:dim(trans_probs)[2]){
  if (i == dim(trans_probs)[2]){
    trans_probs[dim(trans_probs)[1],1] = 0.5
  }
  else
  trans_probs[i,i+1] = 0.5
}
trans_probs
```

```
##       [,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
```

```
#The device will report that the robot is in the sectors [i-2, i+2] with equal probability
#initializing the emission probabilities, NAs are used in order to forbid the transition
#between the states.

accuracy = matrix(c(0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2,
                    0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2,
                    0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA,
                    NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA,
                    NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA,
                    NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA,
                    NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA,
                    NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2,
                    0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2,
                    0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2), nrow = 10, ncol = 10)


hmm = initHMM(states_vec, symbols_vec, start_prob, trans_probs, accuracy)
cat('The states of HMM are: \n',hmm$States , '\n')
```

```
## The states of HMM are:
##  1 2 3 4 5 6 7 8 9 10
```

```
cat('The symbols of HMM are: \n',hmm$Symbols , '\n')
```

```
## The symbols of HMM are:
##  A B C D E F G H I J
```

```
cat('The transition probabilities of HMM are: \n')
```

```
## The transition probabilities of HMM are:
```

```
print(hmm$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
```

Question 2:

Simulate the HMM for 100 time steps.

```
length = 100
simulated_hmm = simHMM(hmm, length)
cat('The simulated states are: \n')
```

```
## The simulated states are:
```

```
print(simulated_hmm$states)
```

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

```
cat('\n The simulated observations are: \n')
```

```
##
##  The simulated observations are:
```

```
print(simulated_hmm$observation)
```

```
##   [1] "G" "J" "H" "J" "B" "C" "J" "C" "D" "D" "E" "D" "B" "C" "B" "F" "F" "E"
##  [19] "D" "C" "E" "E" "H" "I" "J" "I" "I" "J" "B" "J" "B" "E" "C" "B" "F" "F"
##  [37] "D" "G" "G" "F" "E" "I" "G" "J" "J" "C" "C" "A" "C" "C" "F" "E" "D" "G"
##  [55] "G" "I" "I" "J" "F" "I" "J" "B" "I" "I" "H" "A" "C" "B" "C" "D" "C" "B"
##  [73] "E" "D" "D" "B" "D" "F" "D" "F" "H" "J" "H" "G" "F" "F" "G" "H" "I" "J"
##  [91] "A" "I" "B" "B" "C" "I" "B" "J" "D" "A"
```

Question 3,4:

Question 3:

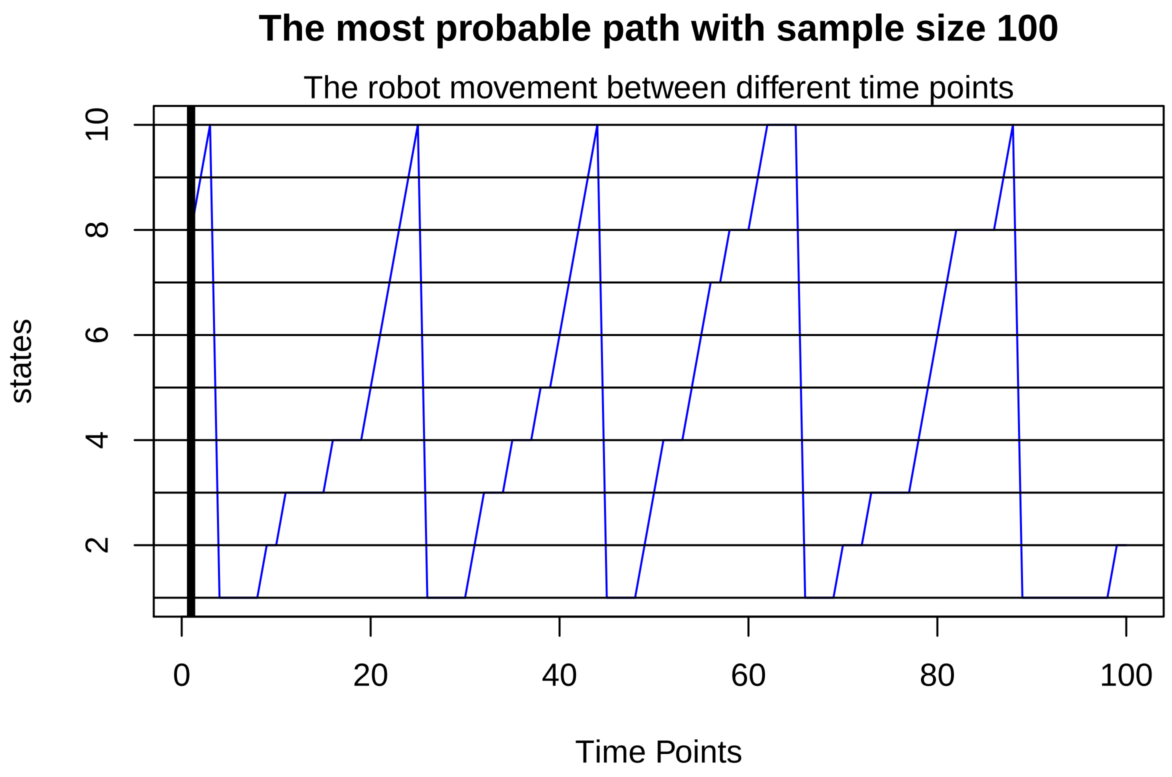
Discard the hidden states from the sample obtained above. Use the remaining obser- vations to compute the filtered and smoothed probability distributions for each of the 100 time points. Compute also the most probable path.

Question 4:

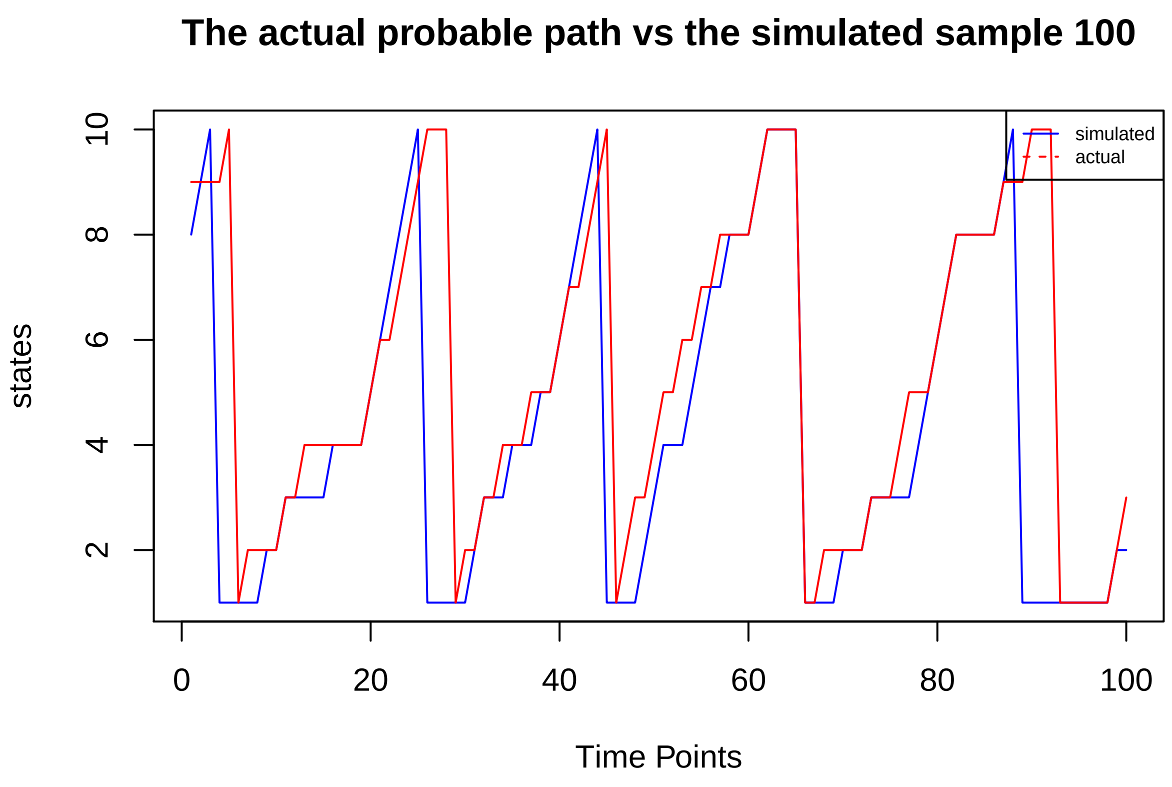
Compute the accuracy of the filtered and smoothed probability distributions, and of the most probable path. That is, compute the percentage of the true hidden states that are guessed by each method.

```r
library(HMM)
RNGversion(vstr = 3.6)
set.seed(12345)

#Question 1: Build a hidden Markov model (HMM) for the scenario described above.

# Initialise HMM
states_vec = c(1:10)
symbols_vec = LETTERS[seq( from = 1, to = 10 )]
start_prob = rep(0.1,10)

#The robot can stay in the current sector or move to the next sector with equal probability.
trans_probs=matrix(data = 0, nrow = length(states_vec), ncol = length(states_vec))
diag(trans_probs) = 0.5
for (i in 1:dim(trans_probs)[2]){
  if (i == dim(trans_probs)[2]){
    trans_probs[dim(trans_probs)[1],1] = 0.5
  }
  else
    trans_probs[i,i+1] = 0.5
}


#the device will report that the robot is in the sectors [i-2, i+2] with equal probability
#initializing the emission probabilities, NAs are used in order to forbid the transition
#between the states.

accuracy = matrix(c(0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2,
                    0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2,
                    0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA,
                    NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA,
                    NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA,
                    NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA,
                    NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA,
                    NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2,
                    0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2,
                    0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2), nrow = 10, ncol = 10)

#Initializaing Hidden Markov Model
hmm = initHMM(states_vec, symbols_vec, start_prob, trans_probs, accuracy)

#Question 2: Simulate the HMM for 100 time steps.

simulate_hmm = function(no_of_simulations){

  length = no_of_simulations

  cat('>>>>>>>>>>>>>>>>>>>>>>>>>>>>Simulating for ', length , 'time steps >>>>>>>>>>>>>>>>>>>>>>>>>>>>
  simulated_hmm = simHMM(hmm, length)

  cat('The simulated states are: \n')
  #print(simulated_hmm$states)
  cat('The simulated observations are: \n')
  #print(simulated_hmm$observation)

  #Question 3: Discard the hidden states from the sample obtained above. Use the remaining
  #observations to compute the filtered and smoothed probability distributions for each of
  #the 100 time points. Compute also the most probable path.

  #Calculating the filter
  normalized_filter = calculate_filter(hmm, simulated_hmm$observation)
  #Calculating the smooth
  smoothed = calculate_smooth(hmm, simulated_hmm$observation)
  #Calculating the path
  probable_path = calculate_path(hmm, simulated_hmm$observation)

  #Plotting the most probable path:
  plot_path(length, probable_path)
  #Question 4: Compute the accuracy of the filtered and smoothed probability distributions,
  #and of the most probable path. That is, compute the percentage of the true hidden states
  #that are guessed by each method. The accuracy of the filtered probabilities

  calculate_filter_accuracy(normalized_filter, simulated_hmm$states)

  #The accuracy of the smoothed probabilities
  calculate_smooth_accuracy(smoothed, simulated_hmm$states)

  #Calculating the probability of the most probable path
  ac_vs_sim_path(length, probable_path, simulated_hmm$states)

  prob_path_accuracy(probable_path, simulated_hmm$states)

  calculate_entropy(length, normalized_filter)

}

calculate_filter = function(hmm,observations){

  #The forward-function computes the forward probabilities. The forward probability for state X
  #up to observation at time k is defined as the probability of observing the sequence of observations
  #e_1, ... ,e_k and that the state at time k is X
  filtered = forward(hmm, observations)
  cat('The filtered probabilities are: \n')
  #Normalizing the filtered states:
  filtered = exp(filtered)
  normalized_filter = prop.table(filtered,2)
  if (length(observations)==100){
    z_100 = normalized_filter[,100]
    transition_prob = hmm$transProbs
    cat('The solution for question 7: \n')
    cat('The probability of the hidden states for the time step 101 is: \n')
    print(transition_prob%*%z_100)
  }
  return(normalized_filter)

}

calculate_smooth = function(hmm, observations){

  #The posterior function computes the posterior probabilities of being in state X at time
  #k for a given sequence of observations and a given Hidden Markov Model.
  smoothed = posterior(hmm, observations)
  cat('The posterior probabilities of states given the observations are: \n')
  #print(smoothed)
  return(smoothed)

}

calculate_path = function(hmm,observations){
  #Calculating the most probable path:
  probable_path = viterbi(hmm,observations)
  return(probable_path)
}

plot_path = function(length, probable_path){
  plot(c(1:length),
       probable_path,
       main = paste("The most probable path with sample size",length),
       xlab = "Time Points",
       ylab = "states",
       type = 'l',
       col = 'blue',
       lwd = 1)
  mtext('The robot movement between different time points')
  abline(h = pretty(probable_path, 10),
         v = pretty(1, 100),
         col = "black")
}

ac_vs_sim_path = function(length, probable_path, states){
  plot(c(1:length),
       probable_path,
       main = paste("The actual probable path vs the simulated sample",length),
       xlab = "Time Points",
       ylab = "states",
       type = 'l',
       col = 'blue',
       lwd = 1)
  lines(c(1:length),states, col = 'red')
  legend('topright',
         legend = c("simulated", "actual"),
         col = c('blue','red'),
         lty=1:5,
         cex=0.6)

}

calculate_filter_accuracy = function(normalized_filter, states){
  #Taking the maximum of the normalized filter of each time point
  most_probable_filter = apply(normalized_filter, 2, which.max)
  most_probable_filter = as.vector(most_probable_filter)
  #Taking the maximum of the actual simulated states
  actual_states = states

  #Comparing the actual states with the filtered probable states:
  filter_accuracy = which(most_probable_filter == actual_states)
  cat('The states that are equal to the filter are: \n')
  print(length(filter_accuracy))

  cat('The accuracy of filtered probabilities is: \n')
  cat(length(filter_accuracy)/length(actual_states), '\n')
}

calculate_smooth_accuracy = function(smoothed, states){
  actual_states = states
  most_probable_smoothed = apply(smoothed, 2, which.max)
  most_probable_smoothed = as.vector(most_probable_smoothed)

  #Comparing the actual states with the smoothed probable states:

  smooth_accuracy = which(most_probable_smoothed == actual_states)
  cat('The states that are equal to the smoothed are: \n')
  print(length(smooth_accuracy))
  cat('The accuracy of smoothed probabilities is: \n')
  cat(length(smooth_accuracy)/length(actual_states),'\n')
}

prob_path_accuracy = function(probable_path, states){
  cat('The accuracy of the most probable path is: \n')
  acuracy_most_pp = length(which(probable_path==states))/length(probable_path)
  cat(acuracy_most_pp, '\n')
}

calculate_entropy = function(length, normalized_filter){
  filtered_entropy = apply(normalized_filter, 2, entropy.empirical)
  plot(1:length,filtered_entropy,
       type = 'l',
       col = 'red',
       xlab = 'Time Step',
       ylab = 'Empirical Entropy',
       main = paste('The Empirical Entropy for ',length, 'Sample Size'))
}


for(sample in c(100,200,300,400, 1000)){
  simulate_hmm(sample)
}
```

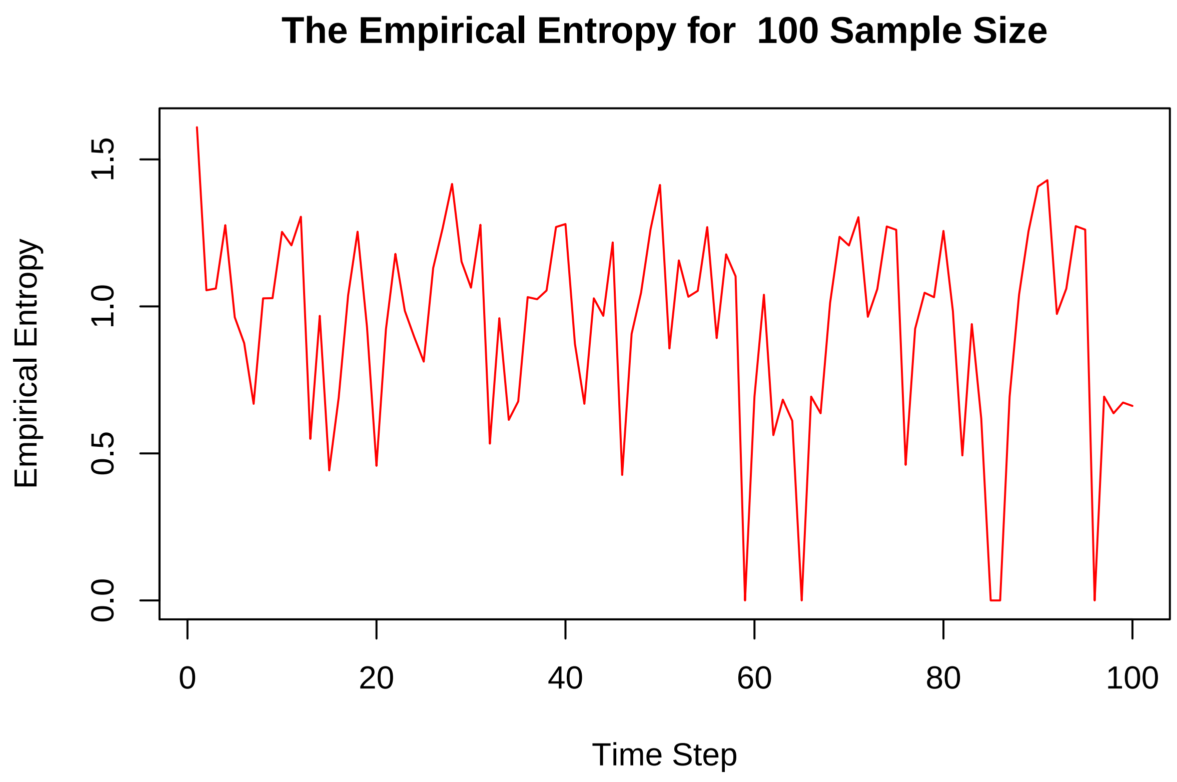
```
## >>>>>>>>>>>>>>>>>>>>>>>>>>>>Simulating for  100 time steps >>>>>>>>>>>>>>>>>>>>>>>>>>>>
## The simulated states are:
## The simulated observations are:
## The filtered probabilities are:
## The solution for question 7:
## The probability of the hidden states for the time step 101 is:
##
## from   [,1]
##   1  0.1875
##   2  0.5000
##   3  0.3125
##   4  0.0000
##   5  0.0000
##   6  0.0000
##   7  0.0000
##   8  0.0000
##   9  0.0000
##   10 0.0000
## The posterior probabilities of states given the observations are:
```



```
## The states that are equal to the filter are:
## [1] 53
## The accuracy of filtered probabilities is:
## 0.53
## The states that are equal to the smoothed are:
## [1] 74
## The accuracy of smoothed probabilities is:
## 0.74
```

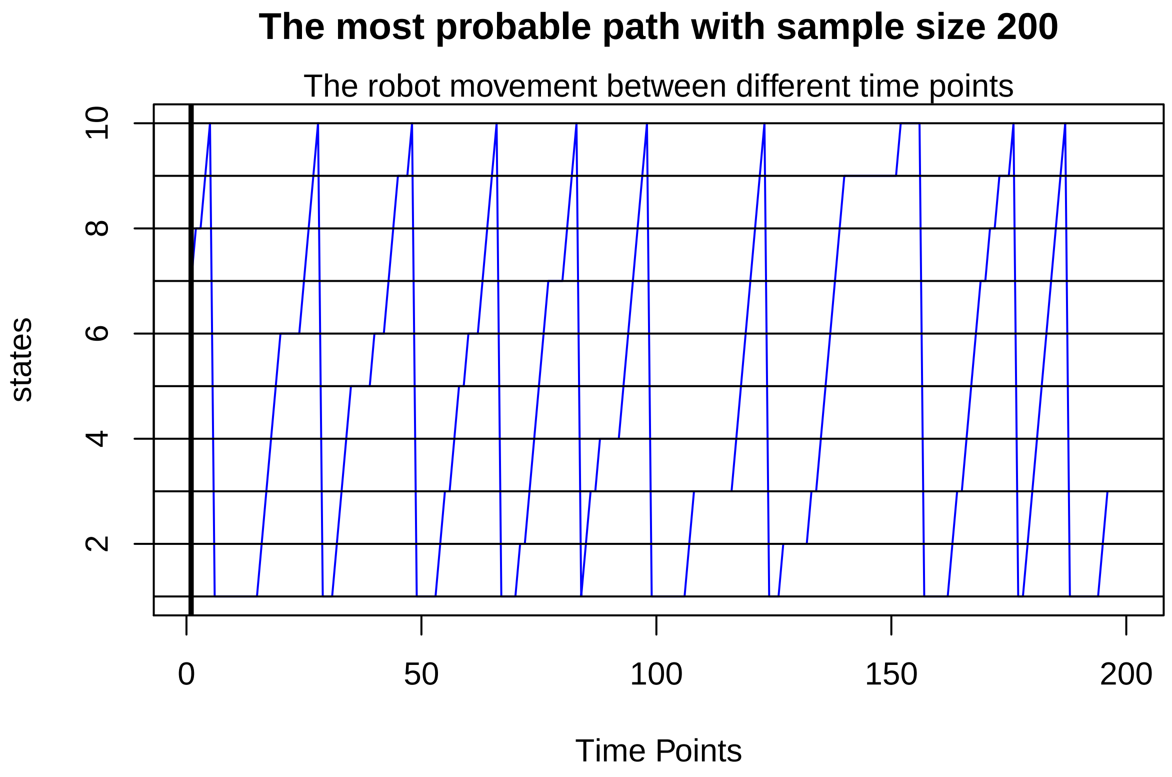


```
## The accuracy of the most probable path is:
## 0.56
```

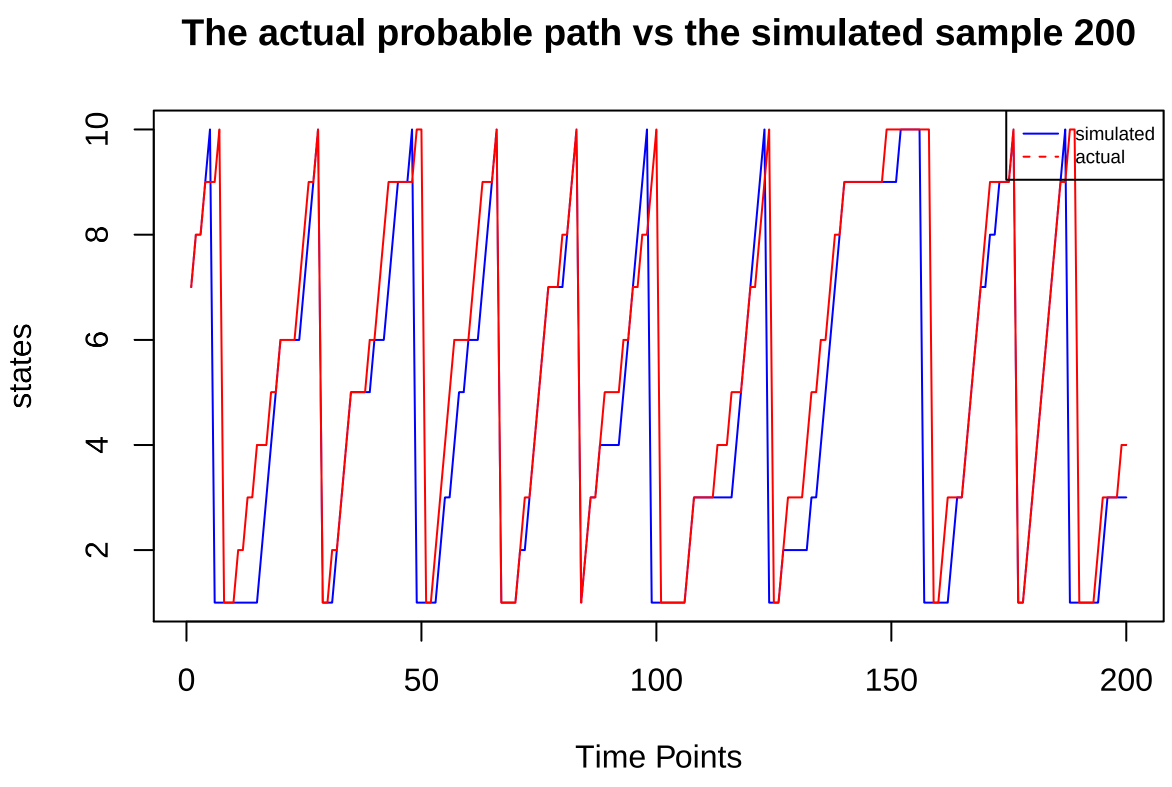


```
## >>>>>>>>>>>>>>>>>>>>>>>>>>>>Simulating for  200 time steps >>>>>>>>>>>>>>>>>>>>>>>>>>>>
## The simulated states are:
```

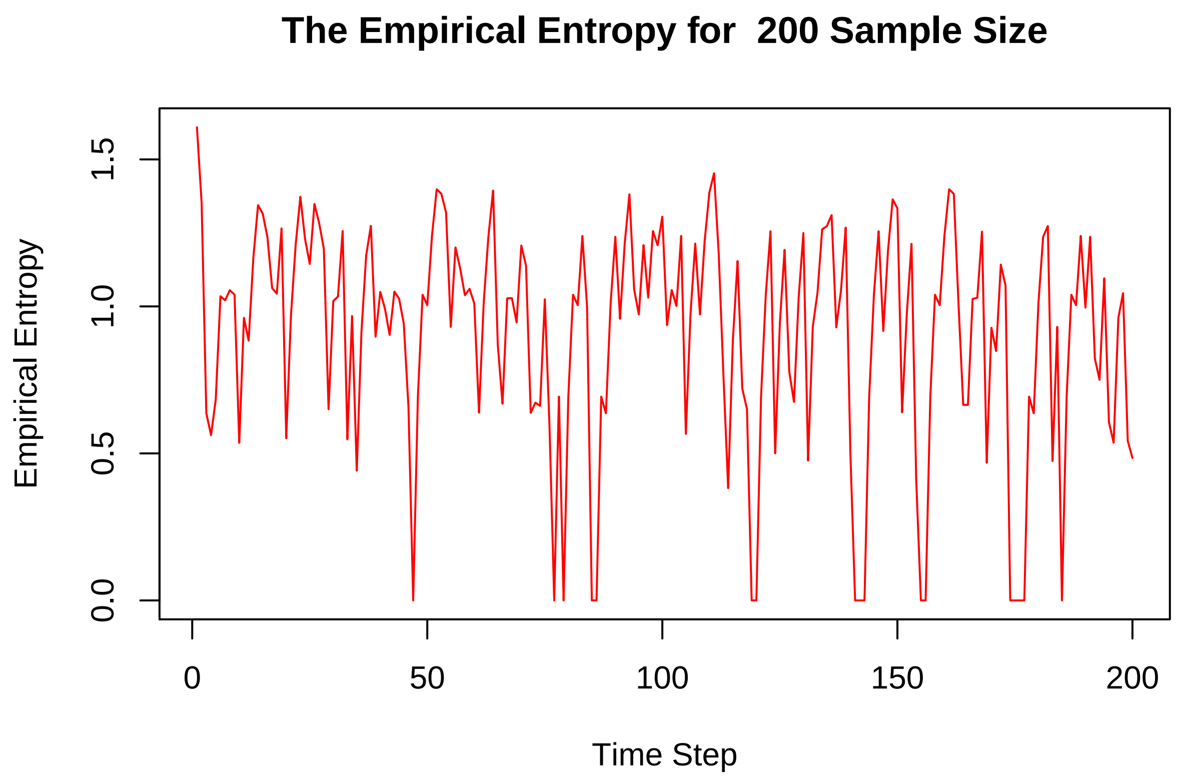
```
## The simulated observations are:
## The filtered probabilities are:
## The posterior probabilities of states given the observations are:
```



```
## The states that are equal to the filter are:
## [1] 110
## The accuracy of filtered probabilities is:
## 0.55
## The states that are equal to the smoothed are:
## [1] 142
## The accuracy of smoothed probabilities is:
## 0.71
```

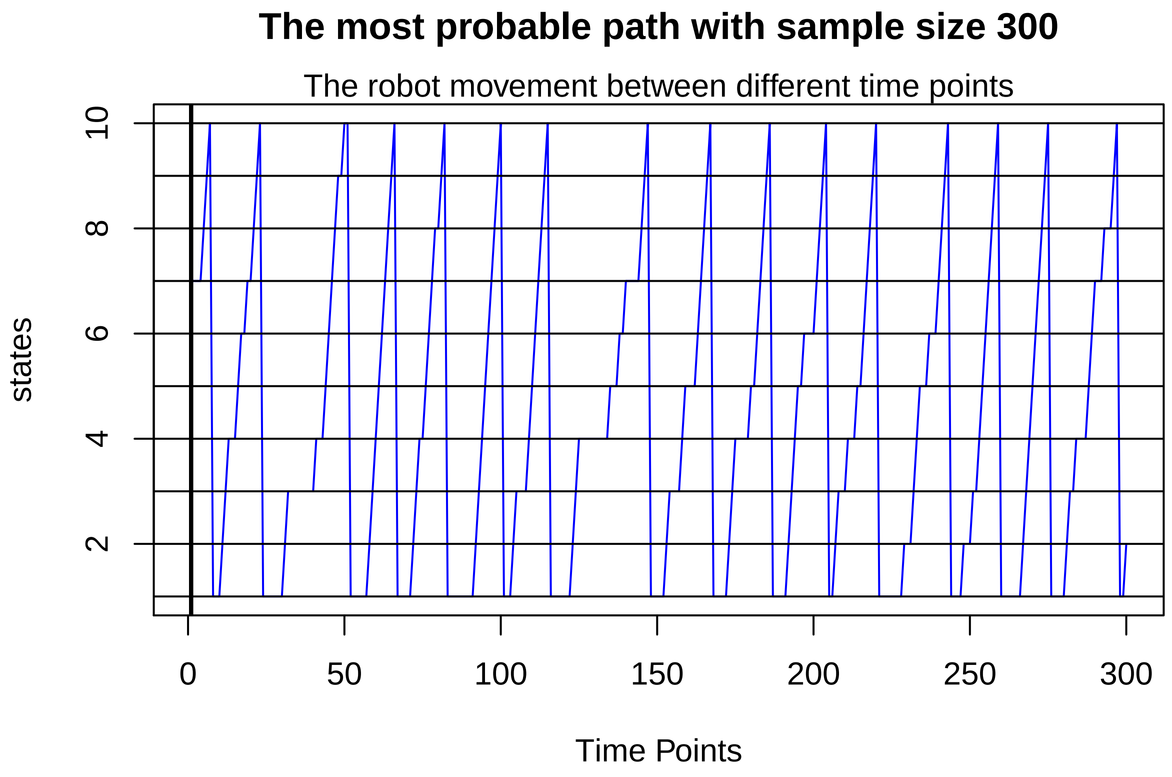


```
## The accuracy of the most probable path is:
## 0.58
```

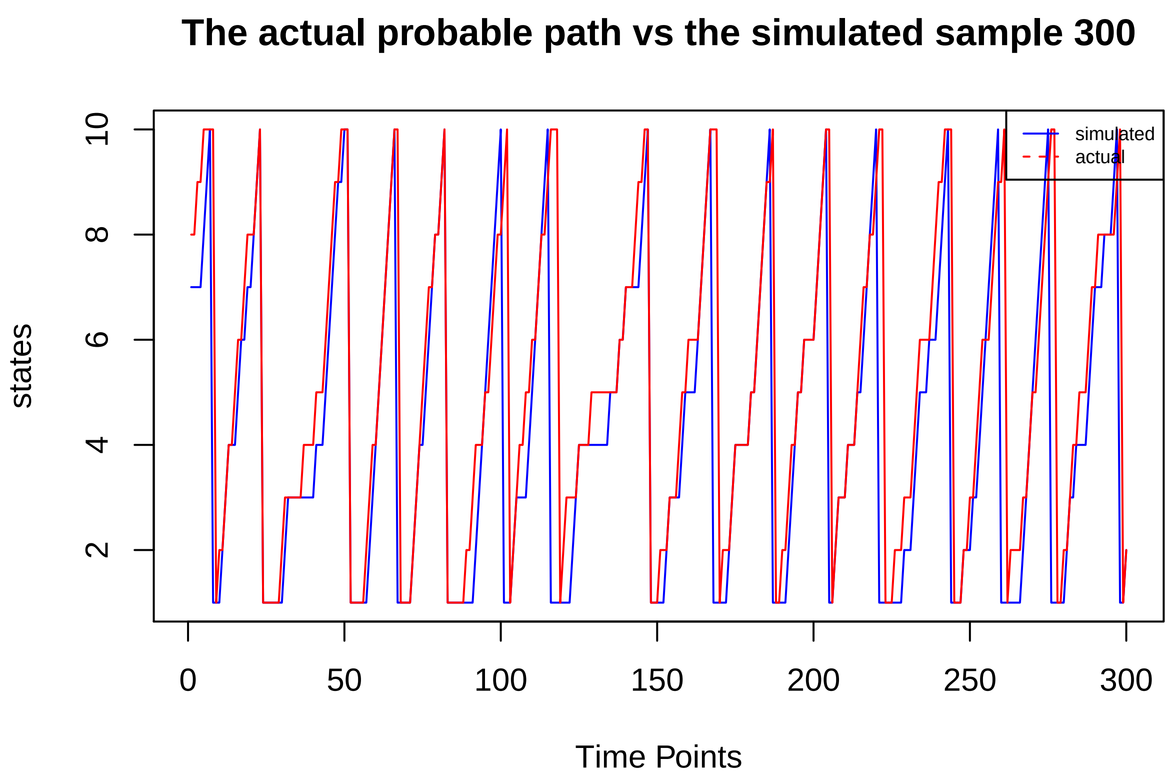


```
## >>>>>>>>>>>>>>>>>>>>>>>>>>>>Simulating for  300 time steps >>>>>>>>>>>>>>>>>>>>>>>>>>>>
## The simulated states are:
```

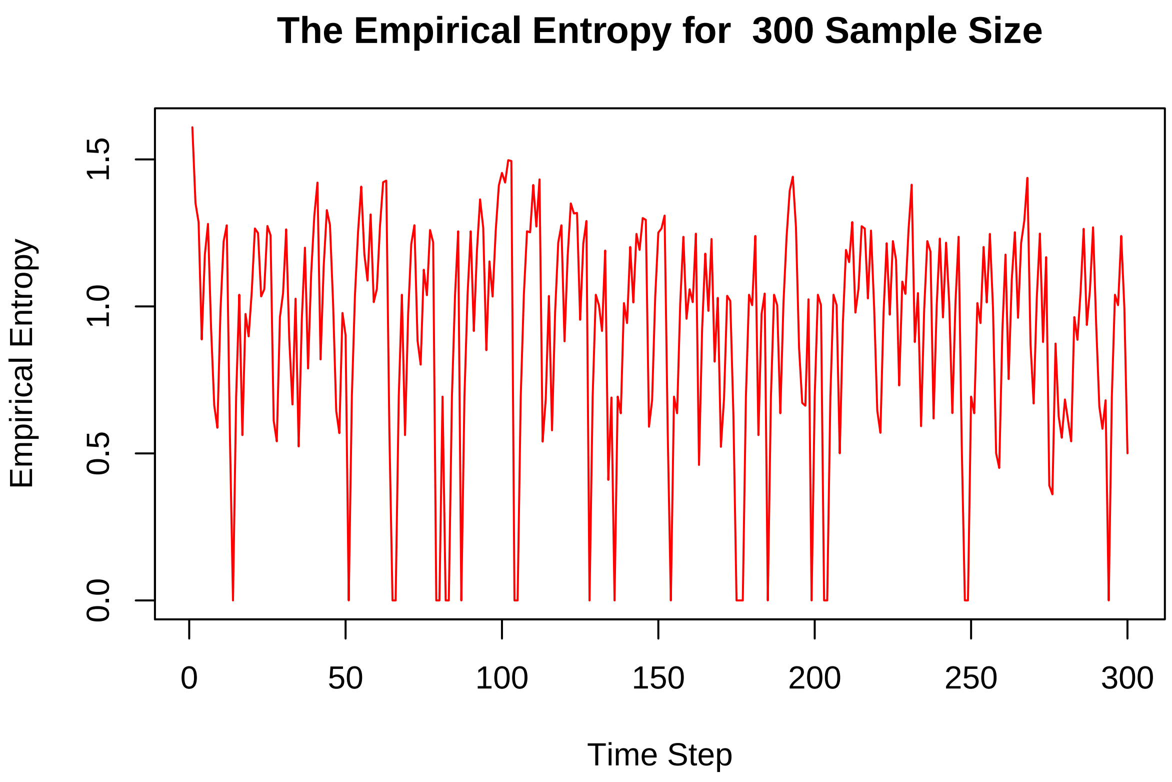
```
## The simulated observations are:
## The filtered probabilities are:
## The posterior probabilities of states given the observations are:
```



```
## The states that are equal to the filter are:
## [1] 183
## The accuracy of filtered probabilities is:
## 0.61
## The states that are equal to the smoothed are:
## [1] 207
## The accuracy of smoothed probabilities is:
## 0.69
```

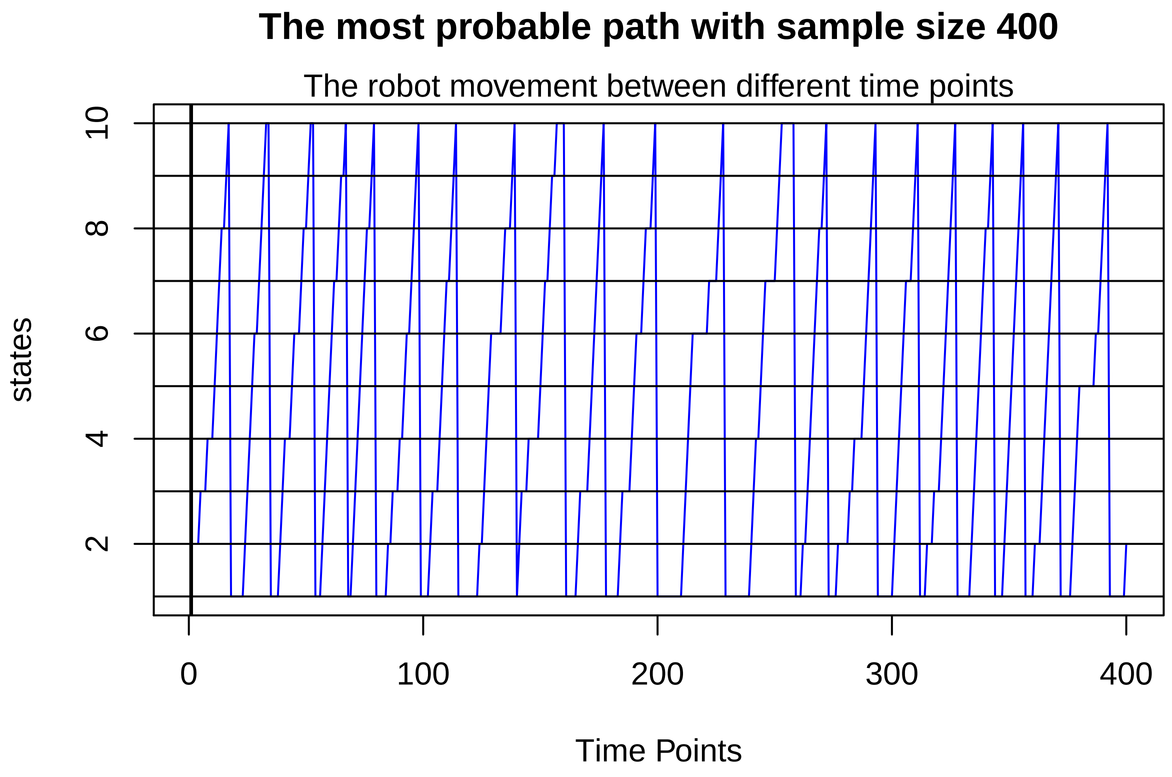


```
## The accuracy of the most probable path is:
## 0.5166667
```

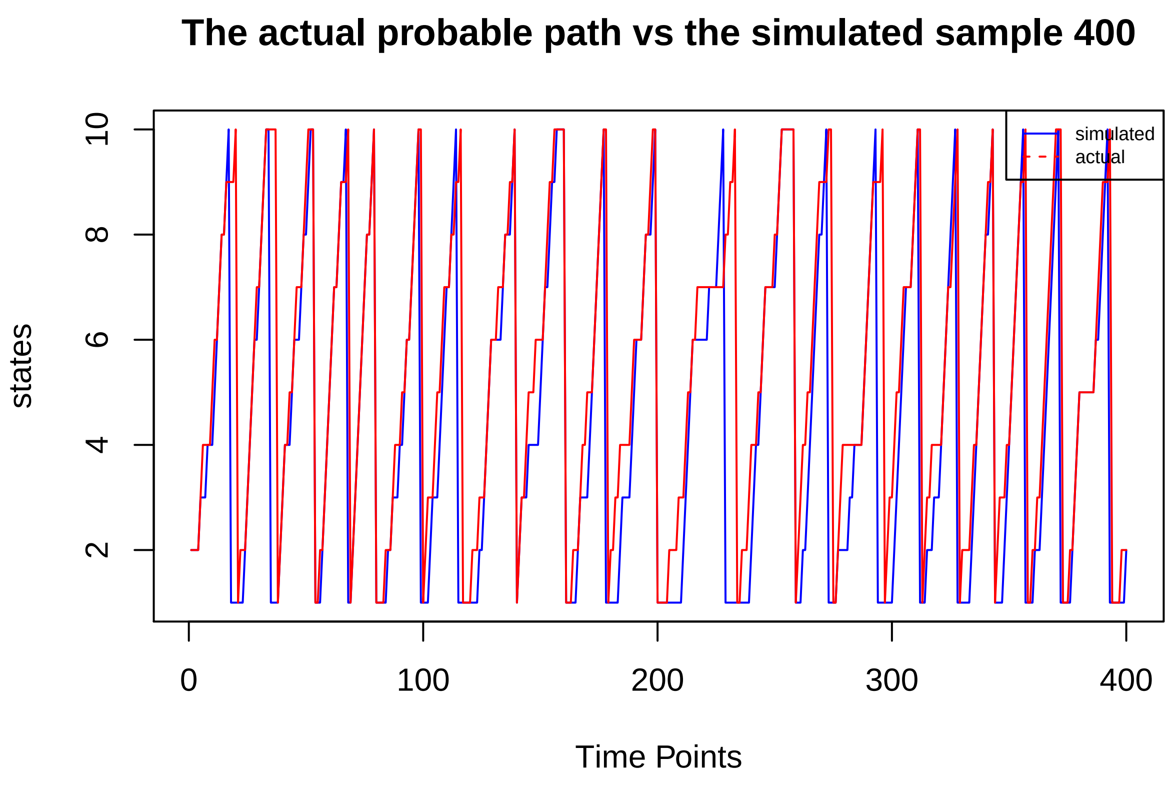


```
## >>>>>>>>>>>>>>>>>>>>>>>>>>>>>Simulating for  400 time steps >>>>>>>>>>>>>>>>>>>>>>>>>>>>>
## The simulated states are:
```

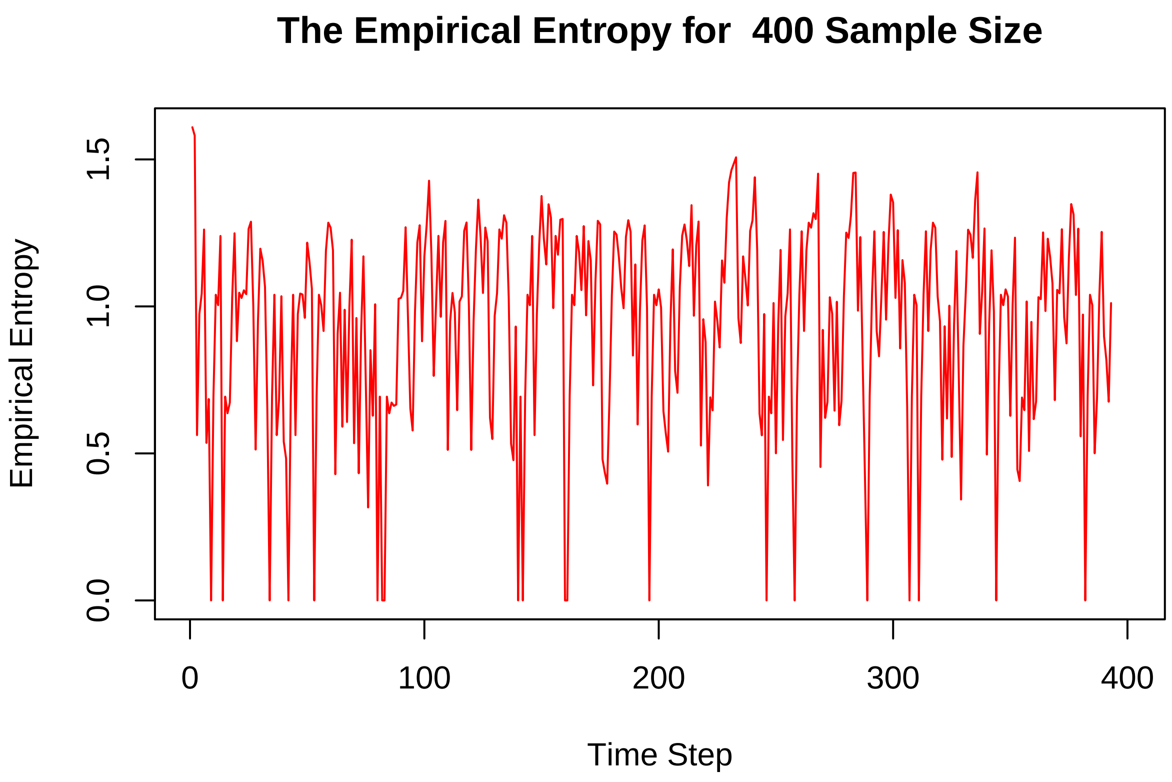
```
## The simulated observations are:
## The filtered probabilities are:
## The posterior probabilities of states given the observations are:
```



```
## The states that are equal to the filter are:
## [1] 203
## The accuracy of filtered probabilities is:
## 0.5075
## The states that are equal to the smoothed are:
## [1] 242
## The accuracy of smoothed probabilities is:
## 0.605
```

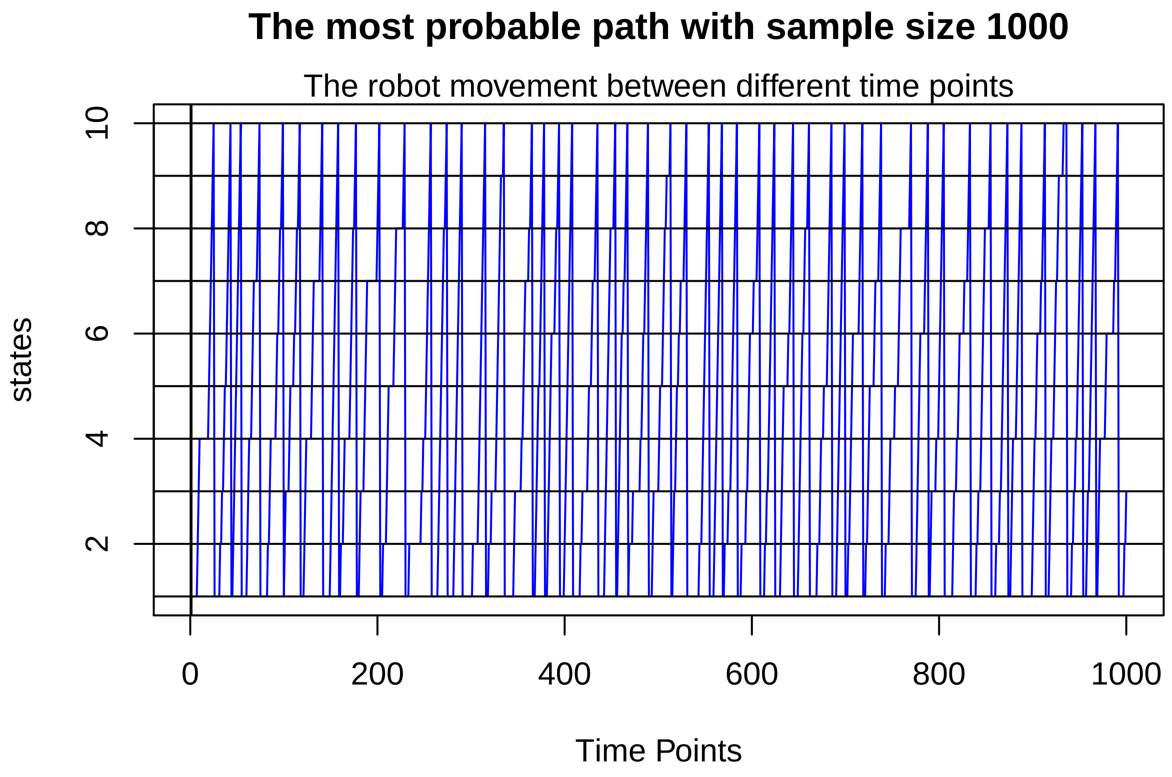


```
## The accuracy of the most probable path is:
## 0.54
```

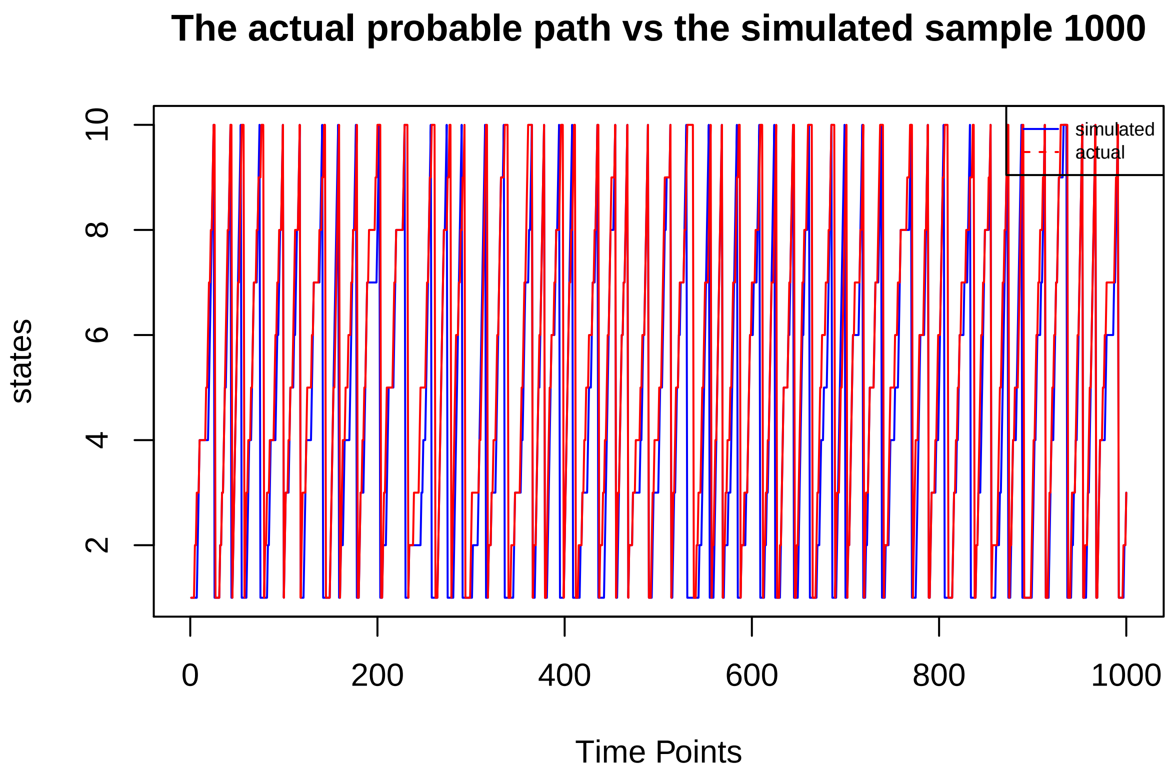


```
## >>>>>>>>>>>>>>>>>>>>>>>>>>>>>Simulating for  1000 time steps >>>>>>>>>>>>>>>>>>>>>>>>>>>>>
## The simulated states are:
```

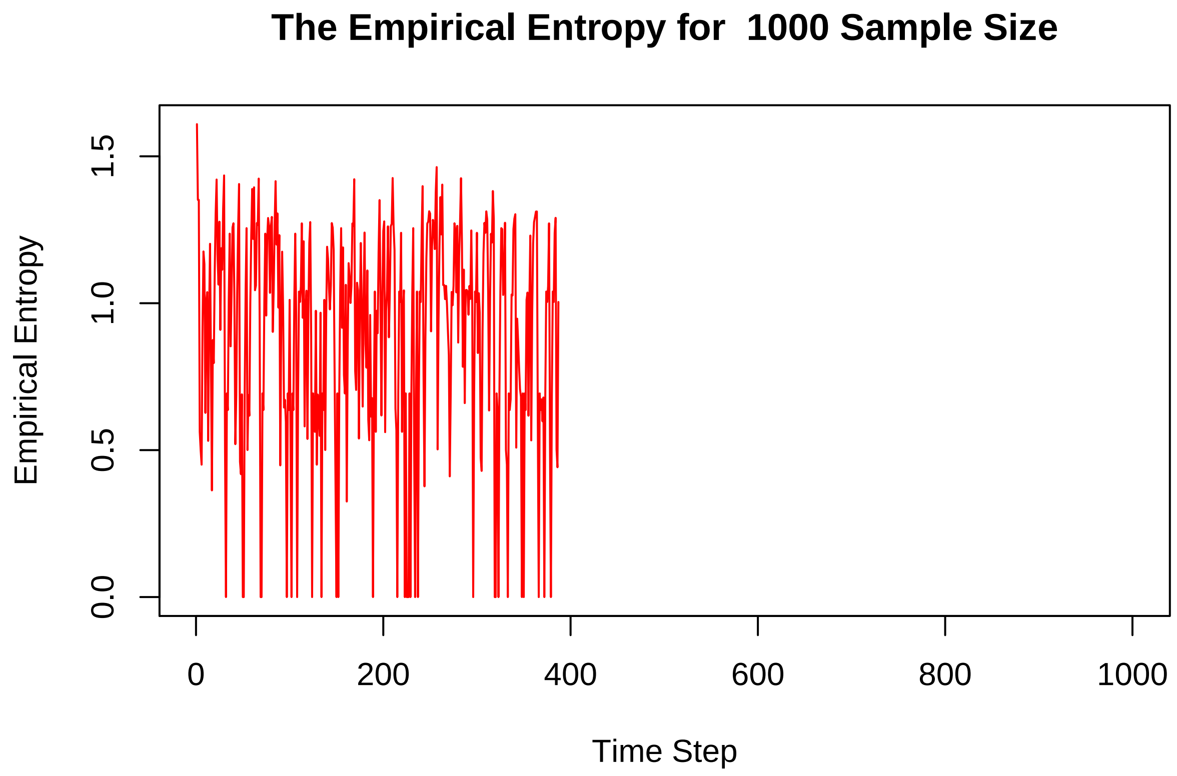
```
## The simulated observations are:
## The filtered probabilities are:
## The posterior probabilities of states given the observations are:
```



```
## The states that are equal to the filter are:
## [1] 209
## The accuracy of filtered probabilities is:
## 0.209
## The states that are equal to the smoothed are:
## [1] 685
## The accuracy of smoothed probabilities is:
## 0.685
```



```
## The accuracy of the most probable path is:
## 0.495
```



Question 5:

Repeat the previous exercise with different simulated samples. In general, the smoothed distributions should be more accurate than the filtered distributions. Why ? In general, the smoothed distributions should be more accurate than the most probable paths, too. Why ?

Answer 5:

The smoothed distribution is more accurate than the filtered one because it takes all the observations up to time T for z^t, whereas the filtered distribution takes the observation up to t for the state z^t, so, by considerting all the observation and not subset of the observation the smoothed distribution is more accurate the filtered distribution.

In the most probable path we take the state that maximizes the expression up to time z^{t+1}. For instance, to maximize over z^0 we have to maximize for all the possible values of z^1 because we don't know the value of z^1 thus, in most probable path we don't take all the data into consideration (only current and next states and observations), and that's why smoothed distribution is more accurate than the most probable path.

Question 6:

Is it true that the more observations you have the better you know where the robot is ?
Hint: You may want to compute the entropy of the filtered distributions with the function entropy.empirical of the package entropy.

Answer 6:

From the graphs, the entropy appears to be random no matter the sample size, and that's because it depends only on the previous observation, so, that means the increase in the number of observations doesn't give us a better knowledge of where the robot is.

Question 7:

Consider any of the samples above of length 100. Compute the probabilities of the hidden states for the time step 101.

Answer 7:

The probability of the hidden state for the time step 101 is equal to the hidden state 100 multiplied by the transition probability. That is $p(z^{101}) = p(z^{101}|z^{100}) = the - transition - probabilities - between - states * p(z^{100})$. The solution is above.

References

https://cran.r-project.org/web/packages/HMM/HMM.pdf.

Appendix

```r
RNGversion('3.5.1')
knitr::opts_chunk$set(echo = TRUE)
library(HMM)
library(entropy)
set.seed(12345)
```

```
# Initialise HMM
states_vec = c(1:10)
symbols_vec = LETTERS[seq( from = 1, to = 10 )]
start_prob = rep(0.1,10)

#The rebot can stay in the current sector or move to the next sector with equal probability.
trans_probs=matrix(data = 0, nrow = length(states_vec), ncol = length(states_vec))
diag(trans_probs) = 0.5
for (i in 1:dim(trans_probs)[2]){
  if (i == dim(trans_probs)[2]){
    trans_probs[dim(trans_probs)[1],1] = 0.5
  }
  else
  trans_probs[i,i+1] = 0.5
}
trans_probs

#The device will report that the robot is in the sectors [i-2, i+2] with equal probability
#initializing the emission probabilities, NAs are used in order to forbid the transition
#between the states.

accuracy = matrix(c(0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2,
                    0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2,
                    0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA,
                    NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA,
                    NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA,
                    NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA,
                    NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA,
                    NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2,
                    0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2,
                    0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2), nrow = 10, ncol = 10)


hmm = initHMM(states_vec, symbols_vec, start_prob, trans_probs, accuracy)
cat('The states of HMM are: \n',hmm$States , '\n')
cat('The symbols of HMM are: \n',hmm$Symbols , '\n')
cat('The transition probabilities of HMM are: \n')
print(hmm$transProbs)

length = 100
simulated_hmm = simHMM(hmm, length)
cat('The simulated states are: \n')
print(simulated_hmm$states)

cat('\n The simulated observations are: \n')
print(simulated_hmm$observation)
library(HMM)
RNGversion(vstr = 3.6)
set.seed(12345)

#Question 1: Build a hidden Markov model (HMM) for the scenario described above.
```

```r
# Initialise HMM
states_vec = c(1:10)
symbols_vec = LETTERS[seq( from = 1, to = 10 )]
start_prob = rep(0.1,10)

#The robot can stay in the current sector or move to the next sector with equal probability.
trans_probs=matrix(data = 0, nrow = length(states_vec), ncol = length(states_vec))
diag(trans_probs) = 0.5
for (i in 1:dim(trans_probs)[2]){
  if (i == dim(trans_probs)[2]){
    trans_probs[dim(trans_probs)[1],1] = 0.5
  }
  else
    trans_probs[i,i+1] = 0.5
}


#the device will report that the robot is in the sectors [i-2, i+2] with equal probability
#initializing the emission probabilities, NAs are used in order to forbid the transition
#between the states.

accuracy = matrix(c(0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2,
                    0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA, 0.2,
                    0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA, NA,
                    NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA, NA,
                    NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA, NA,
                    NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA, NA,
                    NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2, NA,
                    NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2, 0.2,
                    0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2, 0.2,
                    0.2, 0.2, NA, NA, NA, NA, NA, 0.2, 0.2, 0.2), nrow = 10, ncol = 10)

#Initializaing Hidden Markov Model
hmm = initHMM(states_vec, symbols_vec, start_prob, trans_probs, accuracy)

#Question 2: Simulate the HMM for 100 time steps.

simulate_hmm = function(no_of_simulations){

  length = no_of_simulations

  cat('>>>>>>>>>>>>>>>>>>>>>>>>>>>>Simulating for ', length , 'time steps >>>>>>>>>>>>>>>>>>>>>>>>>>>>
  simulated_hmm = simHMM(hmm, length)

  cat('The simulated states are: \n')
  #print(simulated_hmm$states)
  cat('The simulated observations are: \n')
  #print(simulated_hmm$observation)

  #Question 3: Discard the hidden states from the sample obtained above. Use the remaining
  #observations to compute the filtered and smoothed probability distributions for each of
  #the 100 time points. Compute also the most probable path.
  #Calculating the filter
  normalized_filter = calculate_filter(hmm, simulated_hmm$observation)
  #Calculating the smooth
  smoothed = calculate_smooth(hmm, simulated_hmm$observation)
  #Calculating the path
  probable_path = calculate_path(hmm, simulated_hmm$observation)

  #Plotting the most probable path:
  plot_path(length, probable_path)
  #Question 4: Compute the accuracy of the filtered and smoothed probability distributions,
  #and of the most probable path. That is, compute the percentage of the true hidden states
  #that are guessed by each method. The accuracy of the filtered probabilities

  calculate_filter_accuracy(normalized_filter, simulated_hmm$states)

  #The accuracy of the smoothed probabilities
  calculate_smooth_accuracy(smoothed, simulated_hmm$states)

  #Calculating the probability of the most probable path
  ac_vs_sim_path(length, probable_path, simulated_hmm$states)

  prob_path_accuracy(probable_path, simulated_hmm$states)

  calculate_entropy(length, normalized_filter)

}

calculate_filter = function(hmm,observations){

  #The forward-function computes the forward probabilities. The forward probability for state X
  #up to observation at time k is defined as the probability of observing the sequence of observations
  #e_1, ... ,e_k and that the state at time k is X
  filtered = forward(hmm, observations)
  cat('The filtered probabilities are: \n')
  #Normalizing the filtered states:
  filtered = exp(filtered)
  normalized_filter = prop.table(filtered,2)
  if (length(observations)==100){
    z_100 = normalized_filter[,100]
    transition_prob = hmm$transProbs
    cat('The solution for question 7: \n')
    cat('The probability of the hidden states for the time step 101 is: \n')
    print(transition_prob%*%z_100)
  }
  return(normalized_filter)

}

calculate_smooth = function(hmm, observations){

  #The posterior function computes the posterior probabilities of being in state X at time
  #k for a given sequence of observations and a given Hidden Markov Model.
  smoothed = posterior(hmm, observations)
  cat('The posterior probabilities of states given the observations are: \n')
  #print(smoothed)
  return(smoothed)

}

calculate_path = function(hmm,observations){
  #Calculating the most probable path:
  probable_path = viterbi(hmm,observations)
  return(probable_path)
}

plot_path = function(length, probable_path){
  plot(c(1:length),
       probable_path,
       main = paste("The most probable path with sample size",length),
       xlab = "Time Points",
       ylab = "states",
       type = 'l',
       col = 'blue',
       lwd = 1)
  mtext('The robot movement between different time points')
  abline(h = pretty(probable_path, 10),
         v = pretty(1, 100),
         col = "black")
}

ac_vs_sim_path = function(length, probable_path, states){
  plot(c(1:length),
       probable_path,
       main = paste("The actual probable path vs the simulated sample",length),
       xlab = "Time Points",
       ylab = "states",
       type = 'l',
       col = 'blue',
       lwd = 1)
  lines(c(1:length),states, col = 'red')
  legend('topright',
         legend = c("simulated", "actual"),
         col = c('blue','red'),
         lty=1:5,
         cex=0.6)

}

calculate_filter_accuracy = function(normalized_filter, states){
  #Taking the maximum of the normalized filter of each time point
  most_probable_filter = apply(normalized_filter, 2, which.max)
  most_probable_filter = as.vector(most_probable_filter)
  #Taking the maximum of the actual simulated states
  actual_states = states

  #Comparing the actual states with the filtered probable states:
```

```r
  filter_accuracy = which(most_probable_filter == actual_states)
  cat('The states that are equal to the filter are: \n')
  print(length(filter_accuracy))

  cat('The accuracy of filtered probabilities is: \n')
  cat(length(filter_accuracy)/length(actual_states), '\n')
}

calculate_smooth_accuracy = function(smoothed, states){
  actual_states = states
  most_probable_smoothed = apply(smoothed, 2, which.max)
  most_probable_smoothed = as.vector(most_probable_smoothed)

  #Comparing the actual states with the smoothed probable states:

  smooth_accuracy = which(most_probable_smoothed == actual_states)
  cat('The states that are equal to the smoothed are: \n')
  print(length(smooth_accuracy))
  cat('The accuracy of smoothed probabilities is: \n')
  cat(length(smooth_accuracy)/length(actual_states),'\n')
}

prob_path_accuracy = function(probable_path, states){
  cat('The accuracy of the most probable path is: \n')
  acuracy_most_pp = length(which(probable_path==states))/length(probable_path)
  cat(acuracy_most_pp, '\n')
}

calculate_entropy = function(length, normalized_filter){
  filtered_entropy = apply(normalized_filter, 2, entropy.empirical)
  plot(1:length,filtered_entropy,
       type = 'l',
       col = 'red',
       xlab = 'Time Step',
       ylab = 'Empirical Entropy',
       main = paste('The Empirical Entropy for ',length, 'Sample Size'))
}


for(sample in c(100,200,300,400, 1000)){
  simulate_hmm(sample)
}
```