732A96: AML - Computer Lab 2

Julius Kittler (julki092)

September 21, 2019

Contents


```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE,
                      fig.width=6, fig.height=5#, collapse=TRUE
                      )


set.seed(12345)
options(scipen=999)

# General libraries
library(ggplot2)
library(dplyr)

# Specific packages
library(HMM) # ls('package:HMM') # to show all functions
library(entropy)
```

---

1 Assignment 1: Implementation of Hidden Markov Model

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.

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

```
# Prepare hidden states
States = as.character(1:10)

# Prepare observed values
Symbols = as.character(1:10)

# Prepare start probabilities
startProbs = rep(1/10, 10)

# Prepare transProbs: either stay in state (1/2) or move to next state (1/2)
m = diag(10)*1/2 # stay in state with 1/2
m[row(m)+1 == col(m)] = 1/2 # move to next state with 1/2
m[10, 1] = 1/2 # at state 10, we can also move to state 1
transProbs = m

# Prepare emissionProbs: [i-2, i+2] with equal probabilities (1/5)
m = diag(10)*1/5
m[row(m)-2 == col(m)] = 1/5
m[row(m)-1 == col(m)] = 1/5
m[row(m)+1 == col(m)] = 1/5
m[row(m)+2 == col(m)] = 1/5

m[row(m)+8 == col(m)] = 1/5
m[row(m)+9 == col(m)] = 1/5
m[row(m)-8 == col(m)] = 1/5
m[row(m)-9 == col(m)] = 1/5
emissionProbs = m

# Initialize HMM
hmm = initHMM(States = States,
              Symbols = Symbols,
              transProbs = transProbs,
```

```
              startProbs = startProbs,
              emissionProbs = emissionProbs)
print(hmm)
```

```
## $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
```

2 Assignment 2: Simulation

Simulate the HMM for 100 time steps.

```r
set.seed(12345)
sample = simHMM(hmm, 100)
sample
```

```
## $states
##   [1] "9"  "9"  "9"  "9"  "10" "1"  "2"  "2"  "2"  "2"  "3"  "3"  "4"  "4"
##  [15] "4"  "4"  "4"  "4"  "4"  "5"  "6"  "6"  "7"  "8"  "9"  "10" "10" "10"
##  [29] "1"  "2"  "2"  "3"  "3"  "4"  "4"  "4"  "5"  "5"  "5"  "6"  "7"  "7"
##  [43] "8"  "9"  "10" "1"  "2"  "3"  "3"  "4"  "5"  "5"  "6"  "6"  "7"  "7"
##  [57] "8"  "8"  "8"  "8"  "9"  "10" "10" "10" "10" "1"  "1"  "2"  "2"  "2"
##  [71] "2"  "2"  "3"  "3"  "3"  "4"  "5"  "5"  "5"  "6"  "7"  "8"  "8"  "8"
##  [85] "8"  "8"  "9"  "9"  "9"  "10" "10" "10" "1"  "1"  "1"  "1"  "1"  "1"
##  [99] "2"  "3"
##
## $observation
##   [1] "7"  "10" "8"  "10" "2"  "3"  "10" "3"  "4"  "4"  "5"  "4"  "2"  "3"
##  [15] "2"  "6"  "6"  "5"  "4"  "3"  "5"  "5"  "8"  "9"  "10" "9"  "9"  "10"
##  [29] "2"  "10" "2"  "5"  "3"  "2"  "6"  "6"  "4"  "7"  "7"  "6"  "5"  "9"
##  [43] "7"  "10" "10" "3"  "3"  "1"  "3"  "3"  "6"  "5"  "4"  "7"  "7"  "9"
##  [57] "9"  "10" "6"  "9"  "10" "2"  "9"  "9"  "8"  "1"  "3"  "2"  "3"  "4"
##  [71] "3"  "2"  "5"  "4"  "4"  "2"  "4"  "6"  "4"  "6"  "8"  "10" "8"  "7"
##  [85] "6"  "6"  "7"  "8"  "9"  "10" "1"  "9"  "2"  "2"  "3"  "9"  "2"  "10"
##  [99] "4"  "1"
```

3 Assignment 3: Filtered PDF, smoothed PDF and probable path

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.

Recall the smoothing and filtering distribution from lecture 5. Essentially, the filtering distribution gives the probabilities of a certain hidden state at time `t` given all observed states until (and including) time `t`. The smoothing distribution works retrospectively: It gives the probabilities of a hidden state given all observations until `T`, i.e. it uses observations that happened before and observations that happened after `t`.

- *Filtering*: $p(Z^t|x^{0:t}) = \frac{\alpha(Z^t)}{\sum_{z^t}\alpha(z^t)}$
- *Smoothing*: $p(Z^t|x^{0:T}) = \frac{\alpha(Z^t)\beta(Z^t)}{\sum_{z^t}\alpha(z^t)\beta(z^t)}$

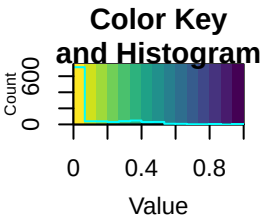
Compute PDFs

```r
# Extract observed values
x = sample$observation

# Compute filtered and smoothed pdf ----------------------------------------
alpha = exp(forward(hmm, x)) # forward probabilities (converted from log scale)
beta = exp(backward(hmm, x)) # backward probabilities (converted from log scale)
product = alpha * beta # element wise product

# Compute filtered pdf
pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)
# prop.table(alpha, 2)

# Compute smoothed pdf
pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)
# posterior(hmm, sim_obs)

# Check that all columns sum up to 1
if (all(round(colSums(pdf_filtered), 4) == 1) &&
    all(round(colSums(pdf_smoothed), 4) == 1)){
  print("All columns sum up to 1")
} else {
  print("There is an issue: Not all columns sum up to 1")
}
```

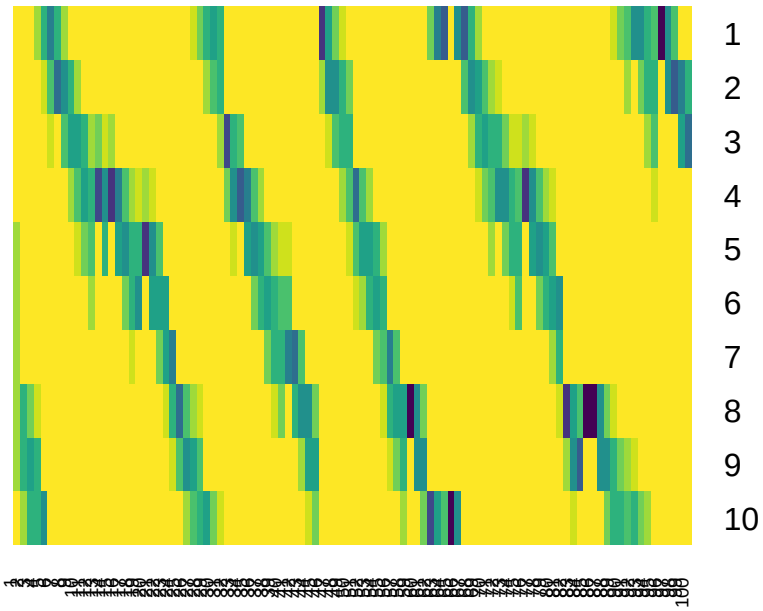
```
## [1] "All columns sum up to 1"
```

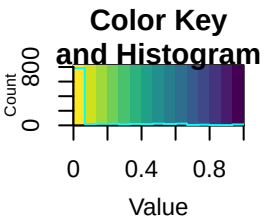
Visualize PDFs

We can see better results for the smoothed pdf, which uses more data (also future observations relative to `t`). We can see that its results are better because the areas/paths with high probabilites are very slim and clear-cut (unlike for the filtered pdf).

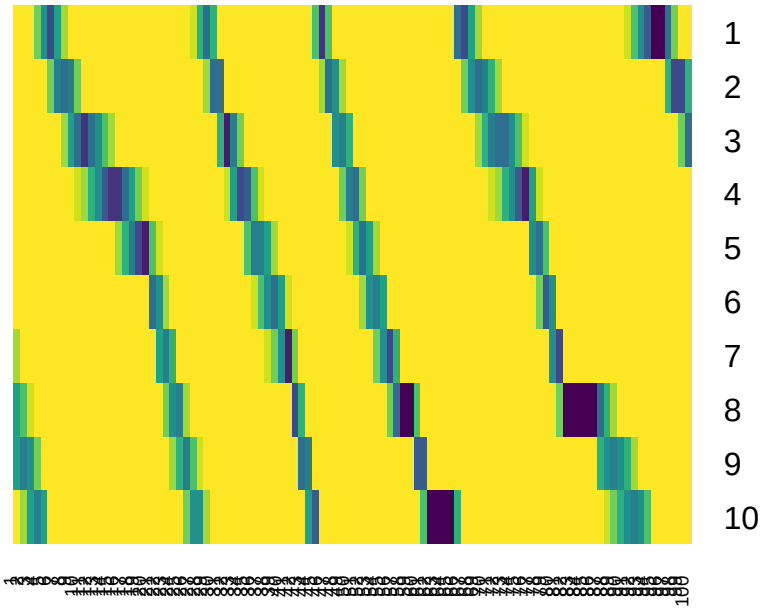


Filtered PDF





Smoothed PDF



Most probable path

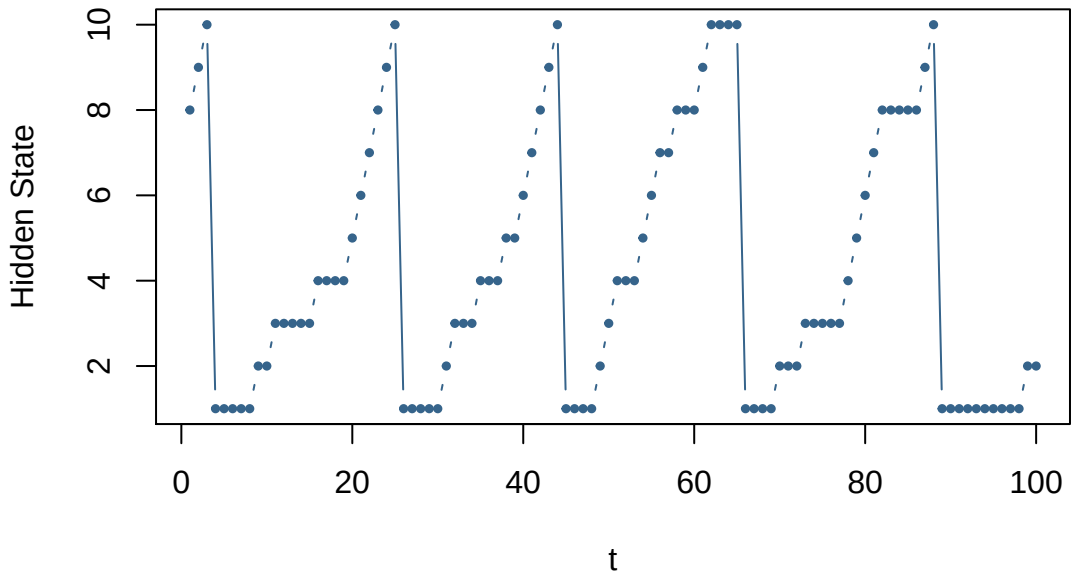
Essentially, the most probable path should consist of 100 values $\in [1, 10]$ for the hidden variable. We know that there cannot be any jumps: The robot can only walk from sector 1 to 2, 2 to 3 etc. It cannot jump e.g. from 1 to 3. We also know that there cannot be any walks back to a previous sector since according to the question, the robot can either "stay" or "move to the next sector".

From the lecture, we know that the *Forward-Backward Algorithm* is not ideal for finding the post probable path because it allows for jumps (see the fuzzy paths in the previous heatmaps). In contrast to that, the *Viterbi Algorithm* does not allow for jumps. Instead, it directly computes the most probable path given the observations and an HMM. Therefore, we choose to use the `viterbi(hmm, x)` function here.

```
# Compute most probable path
path = viterbi(hmm, x)
path
```

```
##   [1] "8"  "9"  "10" "1"  "1"  "1"  "1"  "1"  "2"  "2"  "3"  "3"  "3"  "3"
##  [15] "3"  "4"  "4"  "4"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "1"  "1"  "1"
##  [29] "1"  "1"  "2"  "3"  "3"  "3"  "4"  "4"  "4"  "5"  "5"  "6"  "7"  "8"
##  [43] "9"  "10" "1"  "1"  "1"  "1"  "2"  "3"  "4"  "4"  "4"  "5"  "6"  "7"
##  [57] "7"  "8"  "8"  "8"  "9"  "10" "10" "10" "10" "1"  "1"  "1"  "1"  "2"
##  [71] "2"  "2"  "3"  "3"  "3"  "3"  "3"  "4"  "5"  "6"  "7"  "8"  "8"  "8"
##  [85] "8"  "8"  "9"  "10" "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"  "1"
##  [99] "2"  "2"
```

```
# Plot most probable path
plot(path, type = "b", pch = 19, col = "steelblue4",
     xlab = "t", ylab = "Hidden State", cex = 0.5)
```



4 Assignment 4: Accuracy

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.

Hint: Note that the function `forward` in the `HMM` package returns probabilities in log scale. You may need to use the functions `exp` and `prop.table` in order to obtain a normalized probability distribution. You may also want to use the functions `apply` and `which.max` to find out the most probable states. Finally, recall that you can compare two vectors A and B elementwise as A==B, and that the function `table` will count the number of times that the different elements in a vector occur in the vector.

As expected, the accuracy is higher for the most probable path acc. to the smoothed PDF than for the filtered PDF. This is because the smoothed PDF is conditional on all observations (not just until `t`). As we could see in the heatmaps before, the paths were more clear cut and therefore any jumps less unlikely.

```
# Accuracy for filtered PDF
pred = rownames(pdf_filtered)[apply(pdf_filtered, 2, which.max)]
accuracy = mean(pred == sample$states)
cat("\nAccuracy for most probable path acc. to filtered PDF: ", accuracy)
```

```
##
## Accuracy for most probable path acc. to filtered PDF:  0.53
```

```
# Accuracy for smoothed PDF
pred = rownames(pdf_smoothed)[apply(pdf_smoothed, 2, which.max)]
accuracy = mean(pred == sample$states)
cat("Accuracy for most probable path acc. to smoothed PDF: ", accuracy)
```

```
## Accuracy for most probable path acc. to smoothed PDF:  0.74
```

```
# Accuracy for smoothed PDF
accuracy = mean(path == sample$states)
cat("Accuracy for most probable path acc. to viterbi: ", accuracy)
```

```
## Accuracy for most probable path acc. to viterbi:  0.56
```

5 Assignment 5: Accuracy Comparison

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?

Approach

Here, we generate samples for 10, 20, ..., 350 observations. For all samples, we compute the accuracies based on smoothed and filtered PDF and based on Viterbi's most probable path. Note that a large number of observations leads to numerical issues at some point (underflow) when increasing the number of observations further.

Observations

- The accuracies of the smoothed distribution seem generally larger than the accuracies of the filtered distribution and the viterbi algorithm
- It does not seem like there is a consistent improvement of the accuracies over time, even though the accuracies do seem to increase noticeably from N = 10 to N = 100 for the filtered and smoothed distribution.

Smoothed vs. filtered and viterbi

The smoothed distribution gives better accuracies than the filtered distribution because it is given more information than the filtered distribution: It is conditional on $x^{0:T}$ instead of on $x^{0:t}$.

The smoothed distribution gives better accuracies than the path acc. to viterbi because the viterbi algorithm can only go from state to state (instead of jumping) over certain states. Sometimes, this may prevent the algorithm from actually suggesting the correct state for a certain step.

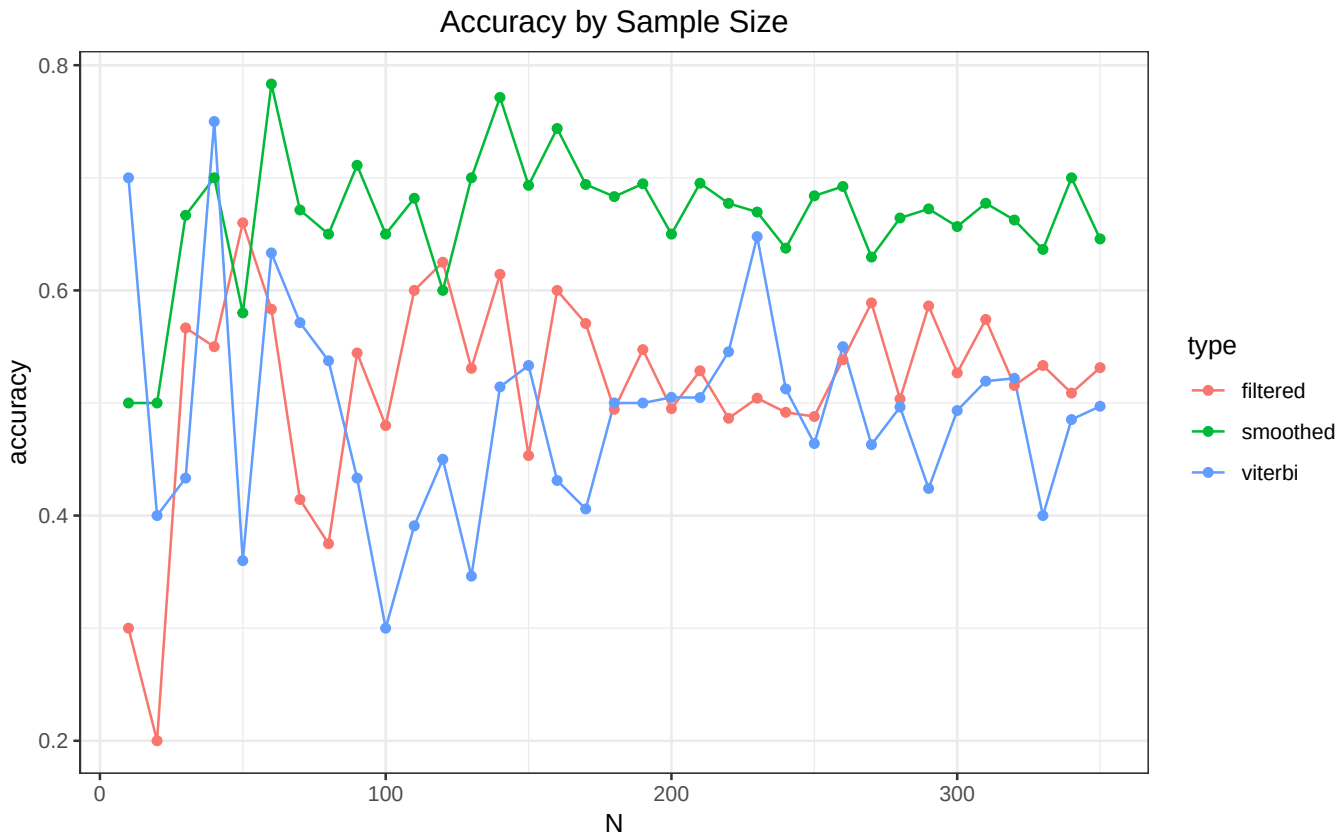
```
get_accuracies = function(n){

  sample = simHMM(hmm, n)
  x = sample$observation
  true = sample$states

  # Compute filtered and smoothed pdf ----------------------------------------
  alpha = exp(forward(hmm, x))
  beta = exp(backward(hmm, x))
  product = alpha * beta

  # Compute filtered pdf
  pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)

  # Compute smoothed pdf
  pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)

  # Compute accuracies -------------------------------------------------------

  pred_s = rownames(pdf_smoothed)[apply(pdf_smoothed, 2, which.max)]
  pred_f = rownames(pdf_filtered)[apply(pdf_filtered, 2, which.max)]
  pred_v = viterbi(hmm, x)

  acc = c(mean(pred_s == true), mean(pred_f == true), mean(pred_v == true))
  names(acc) = c("smoothed", "filtered", "viterbi")

  return(acc)
```

```
}
```

```
# Get the values
set.seed(12345)
n = seq(10, 350, 10)
M = vapply(n, get_accuracies, numeric(3))
colnames(M) = n
M = round(M, 4)
M
```

```
##           10  20     30   40   50     60     70     80     90  100    110
## smoothed 0.5 0.5 0.6667 0.70 0.58 0.7833 0.6714 0.6500 0.7111 0.65 0.6818
## filtered 0.3 0.2 0.5667 0.55 0.66 0.5833 0.4143 0.3750 0.5444 0.48 0.6000
## viterbi  0.7 0.4 0.4333 0.75 0.36 0.6333 0.5714 0.5375 0.4333 0.30 0.3909
##            120    130    140    150    160    170    180    190   200
## smoothed 0.600 0.7000 0.7714 0.6933 0.7438 0.6941 0.6833 0.6947 0.650
## filtered 0.625 0.5308 0.6143 0.4533 0.6000 0.5706 0.4944 0.5474 0.495
## viterbi  0.450 0.3462 0.5143 0.5333 0.4312 0.4059 0.5000 0.5000 0.505
##             210    220    230    240   250    260    270    280    290
## smoothed 0.6952 0.6773 0.6696 0.6375 0.684 0.6923 0.6296 0.6643 0.6724
## filtered 0.5286 0.4864 0.5043 0.4917 0.488 0.5385 0.5889 0.5036 0.5862
## viterbi  0.5048 0.5455 0.6478 0.5125 0.464 0.5500 0.4630 0.4964 0.4241
##             300    310    320    330    340    350
## smoothed 0.6567 0.6774 0.6625 0.6364 0.7000 0.6457
## filtered 0.5267 0.5742 0.5156 0.5333 0.5088 0.5314
## viterbi  0.4933 0.5194 0.5219 0.4000 0.4853 0.4971
```

```
# Plot the values
df = as.data.frame(t(M))
df$N = as.numeric(rownames(df))

df = df %>% tidyr::gather(key = "type", value = "accuracy", -N)
ggplot(df, aes(x = N, y = accuracy, color = type)) +
  geom_line() + geom_point() +
  labs(title = "Accuracy by Sample Size") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```



6 Assignment 6: Entropy Comparison

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`.

Accuracy

Based on the accuracies from the previous question, we could conclude that more observations do not always help us know better where the robot is. Once a certain sample size has been reached (in the example above ca. 100), the accuracy does not increase noticeably anymore.

Entropy

Ideally, we want entropy to be as small as possible because then, the distributions would give us high certainty that the robot is at a particular state. Note that every column in our filtered and smoothed distribution represents its own distribution for the hidden state at time `t`. Therefore, we have to estimate the entropy for every column of our filtered and smoothed distribution. Our main focus here is the filtered distribution since we know that the smoothed distribution does not only use information from the past but also from the future. We expect the entropy to slightly decrease with increasing `t` until a certain limit is reached around which it then oscillates.

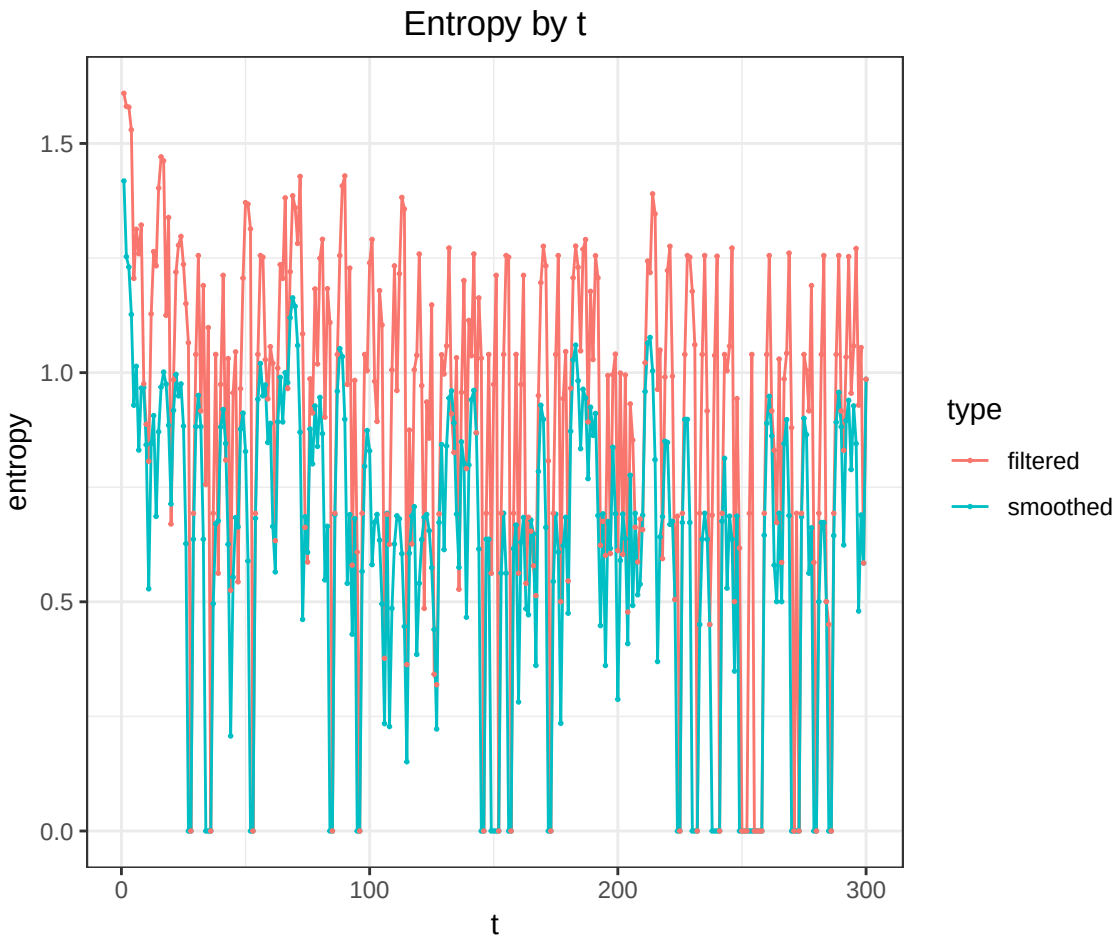
Observations

- The entropy has a burn-in period within approximately $t = 1, 2, ..., 25$. Its maximum is in the very beginning at $t = 1$.
- After the burn-in period, the entropy does not seem to decrease noticeably anymore. Instead, it oscillates noticeably.
- The smoothed distribution generally has a smaller entropy than the filtered distribution which is expected since it uses the information from all t.

Interpretation

Overall, we have a random process. This means that we generally can't be fully certain about the underlying states consistently. Hence, it makes sense that after some burn-in period the entropy won't decrease noticeably anymore and the accuracy won't increase noticeably anymore either.

```
# New solution with entropies for each t (eval = FALSE) -----------------------

N = 300
sample = simHMM(hmm, N)
x = sample$observation
true = sample$states

# Compute filtered and smoothed pdf
alpha = exp(forward(hmm, x))
beta = exp(backward(hmm, x))
product = alpha * beta

pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)
pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)

# Compute entropy for filtered and smoothed pdf at each t
entropy_s = apply(pdf_smoothed, 2, entropy::entropy.empirical)
entropy_f = apply(pdf_filtered, 2, entropy::entropy.empirical)

# Plot the entropy
df_plot = data.frame(smoothed = entropy_s, filtered = entropy_f, t = 1:N)
df_plot = df_plot %>% tidyr::gather(key = "type", value = "entropy", -t)
ggplot(df_plot, aes(x = t, y = entropy, color = type)) +
  geom_line() + geom_point(size = 0.3) +
  labs(title = "Entropy by t") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```



7 Assignment 7: Probabilities of hidden states

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

We are looking for $p(Z^{t+1}|x^{0:T})$.

We don't know Z^t but we have the smoothed and filtered distribution for Z^t. Multiplying this distribution with the transition matrix will give us $p(Z^{t+1}|x^{0:T})$. Note that in this case, it does not matter whether we use the smoothed or the filtered distribution.

Prepare smoothed and filtered distribution

```
n = 100
set.seed(12345)
sample = simHMM(hmm, n)
x = sample$observation

# Compute filtered and smoothed pdf
alpha = exp(forward(hmm, x))
```

```r
beta = exp(backward(hmm, x))
product = alpha * beta

pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)
pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)

# Check if they are identical for t = T
if (all(pdf_smoothed[, n] == pdf_filtered[, n])){
  print("pdf_smoothed and pdf_filtered are identical for t = T = 100")
} else {
  print("There is an issue: pdf_smoothed and pdf_filtered are not identical")
}
```

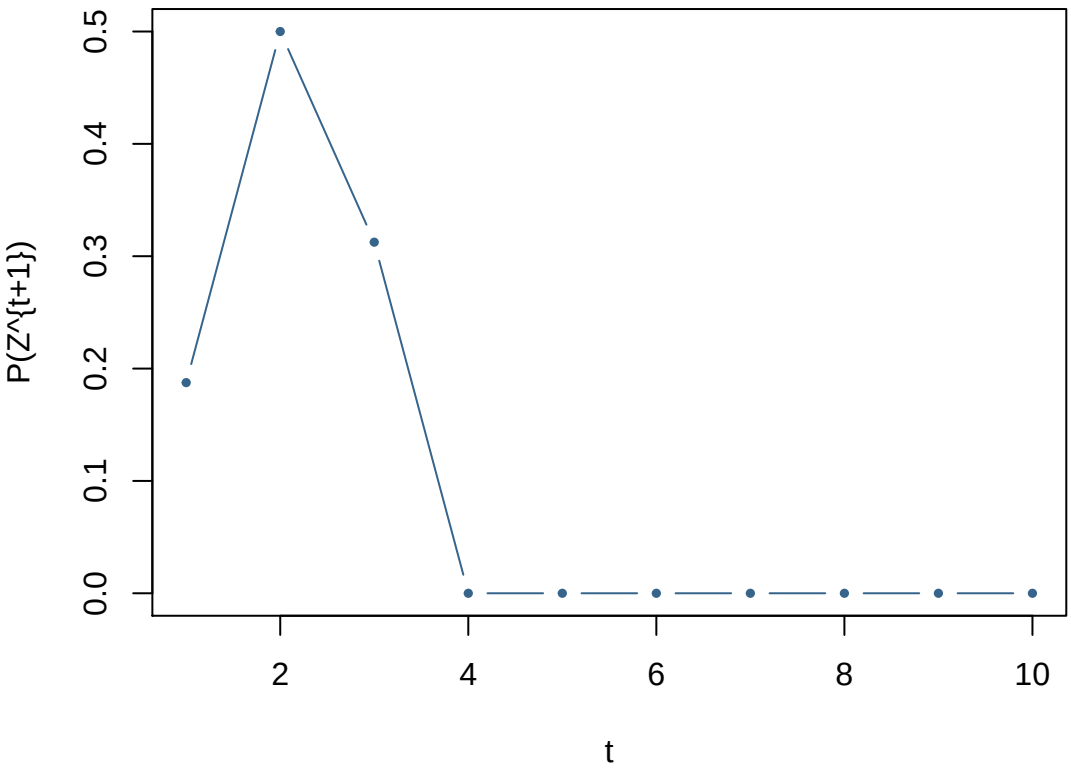
```
## [1] "pdf_smoothed and pdf_filtered are identical for t = T = 100"
```

Compute p(Z^{t+1}|x^{0:T})

```r
# Compute and print p(Z^{t+1}|x^{0:T})
Z_tp1 = transProbs %*% pdf_smoothed[, n]
rownames(Z_tp1) = States
Z_tp1
```

```
##      [,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
```

```r
# Plot p(Z^{t+1}|x^{0:T})
plot(Z_tp1, type = "b", pch = 19, col = "steelblue4",
     xlab = "t", ylab = "P(Z^{t+1})", cex = 0.5,
     main = paste0("Most probable Z^{t+1}: ", names(Z_tp1[which.max(Z_tp1), ])))
```




```
# Print most probable state
names(Z_tp1[which.max(Z_tp1), ])
```

```
## [1] "2"
```

8 Appendix

```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE,
                      fig.width=6, fig.height=5#, collapse=TRUE
                      )

set.seed(12345)
options(scipen=999)

# General libraries
library(ggplot2)
library(dplyr)

# Specific packages
library(HMM) # ls('package:HMM') # to show all functions
```

```r
library(entropy)


# Auxilary functions
analyze_cm = function(cm, true){

  stopifnot(true %in% colnames(cm))
  levels = c(true, colnames(cm)[-which(colnames(cm) == true)]) # ORDER: 1; 0
  cm = as.data.frame(cm); colnames(cm)[1:2] = c("True", "Pred")
  N = sum(cm$Freq)
  Npos = sum(cm$Freq[which(cm$True == levels[1])])
  Nneg = sum(cm$Freq[which(cm$True == levels[2])])
  TP = sum(cm$Freq[which(cm$True == levels[1] & cm$Pred == levels[1])])
  TN = sum(cm$Freq[which(cm$True == levels[2] & cm$Pred == levels[2])])
  FP = sum(cm$Freq[which(cm$True == levels[2] & cm$Pred == levels[1])])

  FN = sum(cm$Freq[which(cm$True == levels[1] & cm$Pred == levels[2])])
  return(data.frame(MCR = (FP+FN)/N, Accuracy = (TP + TN)/N,
                    Recall = TP/Npos, # recall = TPR = sensitivity,
                    Precision = TP/(TP + FP),
                    FPR = FP/Nneg, TNR = TN/Nneg)) # TNR = specificity
}

# cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
# knitr::kable(analyze_cm(cm, true = "yes"))


# Rstudio guide

## Setup
#```{r setup, include=TRUE, results='hide', message=FALSE, warning=FALSE}
#```

## Appendix
# ```{r, ref.label = knitr::all_labels(), echo = TRUE, eval = FALSE}
#```

## Add image
#```{r label, out.height = "400px", out.width = "800px"}
#knitr::include_graphics("image.png")
#```

## Add image
#![Caption for the picture.](/path/to/image.png)
```

```r
# ------------------------------------------------------------------------------
# Assignment 1
# ------------------------------------------------------------------------------

# Prepare hidden states
States = as.character(1:10)

# Prepare observed values
Symbols = as.character(1:10)

# Prepare start probabilities
startProbs = rep(1/10, 10)

# Prepare transProbs: either stay in state (1/2) or move to next state (1/2)
m = diag(10)*1/2 # stay in state with 1/2
m[row(m)+1 == col(m)] = 1/2 # move to next state with 1/2
m[10, 1] = 1/2 # at state 10, we can also move to state 1
transProbs = m

# Prepare emissionProbs: [i-2, i+2] with equal probabilities (1/5)
m = diag(10)*1/5
m[row(m)-2 == col(m)] = 1/5
m[row(m)-1 == col(m)] = 1/5
m[row(m)+1 == col(m)] = 1/5
m[row(m)+2 == col(m)] = 1/5

m[row(m)+8 == col(m)] = 1/5
m[row(m)+9 == col(m)] = 1/5
m[row(m)-8 == col(m)] = 1/5
m[row(m)-9 == col(m)] = 1/5
emissionProbs = m

# Initialize HMM
hmm = initHMM(States = States,
              Symbols = Symbols,
              transProbs = transProbs,
              startProbs = startProbs,
              emissionProbs = emissionProbs)
print(hmm)


# Alternative Emission Probabilities (eval = FALSE) ----------------------------

emissionProbs = matrix(0, ncol = N, nrow = 10)

# 0.2 For i-2 to i+2
```

```r
for (i in 1:N) {
  for (j in c(3:-1)) {
    emissionProbs[((i-j)%%N)+1,i] = 0.2
  }
}

emissionProbs


# ------------------------------------------------------------------------------
# Assignment 2
# ------------------------------------------------------------------------------

set.seed(12345)
sample = simHMM(hmm, 100)
sample


# ------------------------------------------------------------------------------
# Assignment 3
# ------------------------------------------------------------------------------

# Extract observed values
x = sample$observation

# Compute filtered and smoothed pdf ---------------------------------------
alpha = exp(forward(hmm, x)) # forward probabilities (converted from log scale)
beta = exp(backward(hmm, x)) # backward probabilities (converted from log scale)
product = alpha * beta # element wise product

# Compute filtered pdf
pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)
# prop.table(alpha, 2)

# Compute smoothed pdf
pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)
# posterior(hmm, sim_obs)

# Check that all columns sum up to 1
if (all(round(colSums(pdf_filtered), 4) == 1) &&
    all(round(colSums(pdf_smoothed), 4) == 1)){
  print("All columns sum up to 1")
} else {
  print("There is an issue: Not all columns sum up to 1")
}
```

```r
# Visualize filtered pdf
gplots::heatmap.2(pdf_filtered,dendrogram='none', Rowv=FALSE, Colv=FALSE,
                  trace='none', col=viridis::viridis(15, direction = -1),
                  main = "Filtered PDF")

# ggplot(data = reshape2::melt(pdf_filtered),
#        aes(y=states, x=index, fill=value)) +
#        geom_raster()


# Visualize smoothed pdf
gplots::heatmap.2(pdf_smoothed,dendrogram='none', Rowv=FALSE, Colv=FALSE,
                  trace='none', col=viridis::viridis(15, direction = -1),
                  main = "Smoothed PDF")

# ggplot(data = reshape2::melt(pdf_smoothed),
#        aes(y=states, x=index, fill=value)) +
#        geom_raster()


# Compute most probable path
path = viterbi(hmm, x)
path

# Plot most probable path
plot(path, type = "b", pch = 19, col = "steelblue4",
     xlab = "t", ylab = "Hidden State", cex = 0.5)


# ------------------------------------------------------------------------------
# Assignment 4
# ------------------------------------------------------------------------------

# Accuracy for filtered PDF
pred = rownames(pdf_filtered)[apply(pdf_filtered, 2, which.max)]
accuracy = mean(pred == sample$states)
cat("\nAccuracy for most probable path acc. to filtered PDF: ", accuracy)

# Accuracy for smoothed PDF
pred = rownames(pdf_smoothed)[apply(pdf_smoothed, 2, which.max)]
accuracy = mean(pred == sample$states)
cat("Accuracy for most probable path acc. to smoothed PDF: ", accuracy)

# Accuracy for smoothed PDF
accuracy = mean(path == sample$states)
cat("Accuracy for most probable path acc. to viterbi: ", accuracy)
```

```r
# ------------------------------------------------------------------------------
# Assignment 5
# ------------------------------------------------------------------------------

get_accuracies = function(n){

  sample = simHMM(hmm, n)
  x = sample$observation
  true = sample$states

  # Compute filtered and smoothed pdf ------------------------------------------
  alpha = exp(forward(hmm, x))
  beta = exp(backward(hmm, x))
  product = alpha * beta

  # Compute filtered pdf
  pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)

  # Compute smoothed pdf
  pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)

  # Compute accuracies ---------------------------------------------------------

  pred_s = rownames(pdf_smoothed)[apply(pdf_smoothed, 2, which.max)]
  pred_f = rownames(pdf_filtered)[apply(pdf_filtered, 2, which.max)]
  pred_v = viterbi(hmm, x)

  acc = c(mean(pred_s == true), mean(pred_f == true), mean(pred_v == true))
  names(acc) = c("smoothed", "filtered", "viterbi")

  return(acc)
}


# Get the values
set.seed(12345)
n = seq(10, 350, 10)
M = vapply(n, get_accuracies, numeric(3))
colnames(M) = n
M = round(M, 4)
M


# Plot the values
df = as.data.frame(t(M))
```

```r
df$N = as.numeric(rownames(df))

df = df %>% tidyr::gather(key = "type", value = "accuracy", -N)
ggplot(df, aes(x = N, y = accuracy, color = type)) +
  geom_line() + geom_point() +
  labs(title = "Accuracy by Sample Size") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# ------------------------------------------------------------------------------
# Assignment 6
# ------------------------------------------------------------------------------

# New solution with entropies for each t (eval = FALSE) ------------------------

N = 300
sample = simHMM(hmm, N)
x = sample$observation
true = sample$states

# Compute filtered and smoothed pdf
alpha = exp(forward(hmm, x))
beta = exp(backward(hmm, x))
product = alpha * beta

pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)
pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)

# Compute entropy for filtered and smoothed pdf at each t
entropy_s = apply(pdf_smoothed, 2, entropy::entropy.empirical)
entropy_f = apply(pdf_filtered, 2, entropy::entropy.empirical)

# Plot the entropy
df_plot = data.frame(smoothed = entropy_s, filtered = entropy_f, t = 1:N)
df_plot = df_plot %>% tidyr::gather(key = "type", value = "entropy", -t)
ggplot(df_plot, aes(x = t, y = entropy, color = type)) +
  geom_line() + geom_point(size = 0.3) +
  labs(title = "Entropy by t") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# Old solution with entropy averages across all t (eval = FALSE) ---------------

get_entropies = function(n){

  sample = simHMM(hmm, n)
```

```r
  x = sample$observation
  true = sample$states

  # Compute filtered and smoothed pdf ----------------------------------------
  alpha = exp(forward(hmm, x))
  beta = exp(backward(hmm, x))
  product = alpha * beta

  # Compute filtered pdf
  pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)

  # Compute smoothed pdf
  pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)

  # Compute entropies --------------------------------------------------------

  entropy_s = mean(apply(pdf_smoothed, 2, entropy::entropy.empirical))
  entropy_f = mean(apply(pdf_filtered, 2, entropy::entropy.empirical))

  entropies = c(entropy_s, entropy_f)
  names(entropies) = c("smoothed", "filtered")

  return(entropies)
}


# Get the values
n = seq(50, 350, 50)
M = vapply(n, get_entropies, numeric(2))
colnames(M) = n
M = round(M, 4)
M

# Plot the values
df = as.data.frame(t(M))
df$N = as.numeric(rownames(df))

df = df %>% tidyr::gather(key = "type", value = "entropy", -N)
ggplot(df, aes(x = N, y = entropy, color = type)) +
  geom_line() + geom_point() +
  labs(title = "Avg. entropy by Sample Size") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# ----------------------------------------------------------------------------
# Assignment 7
```

```r
# ------------------------------------------------------------------------------

n = 100
set.seed(12345)
sample = simHMM(hmm, n)
x = sample$observation

# Compute filtered and smoothed pdf
alpha = exp(forward(hmm, x))
beta = exp(backward(hmm, x))
product = alpha * beta

pdf_filtered = sweep(alpha, MARGIN = 2, colSums(alpha), `/`)
pdf_smoothed = sweep(product, MARGIN = 2, colSums(product), `/`)

# Check if they are identical for t = T
if (all(pdf_smoothed[, n] == pdf_filtered[, n])){
  print("pdf_smoothed and pdf_filtered are identical for t = T = 100")
} else {
  print("There is an issue: pdf_smoothed and pdf_filtered are not identical")
}


# Compute and print p(Z^{t+1}|x^{0:T})
Z_tp1 = transProbs %*% pdf_smoothed[, n]
rownames(Z_tp1) = States
Z_tp1

# Plot p(Z^{t+1}|x^{0:T})
plot(Z_tp1, type = "b", pch = 19, col = "steelblue4",
     xlab = "t", ylab = "P(Z^{t+1})", cex = 0.5,
     main = paste0("Most probable Z^{t+1}: ", names(Z_tp1[which.max(Z_tp1), ])))

# Print most probable state
names(Z_tp1[which.max(Z_tp1), ])


# Estimate the transition probabilities (eval = FALSE) ------------------------
path = rownames(pdf_smoothed)[apply(pdf_smoothed, 2, which.max)]
M = data.frame(Z_tp1 = path[2:n], Z_t = path[1:(n-1)])
M = table(as.numeric(M$Z_tp1), as.numeric(M$Z_t))

transitionEstimate = round(sweep(M, MARGIN = 2, rowSums(M), `/`), 4)
transitionEstimate
```