Lab 2

Marijn Jaarsma

2024-09-19

```
library(HMM)
library(entropy)
```

Question 1

```
n_states <- n_emissions <- 10

# Fill up the transition probability matrix
# 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.
transProbs <- matrix(NA, nrow=n_states, ncol=n_states)

for (i in 1:nrow(transProbs)) {
  if (i != nrow(transProbs)) {
    transProbs[i, i:(i + 1)] <- 0.5
  }
  else {
    transProbs[i, c(1, i)] <- 0.5
  }
}

# Fill up the emission probability matrix
# If the robot is in sector i, then the device will report that the robot is in
# the sectors [i - 2, i + 2] with equal probability.
emissionProbs <- matrix(NA, nrow=n_states, ncol=n_emissions)

for (i in 1:nrow(emissionProbs)) {
  if (i < 3) {
    # End part of the loop
    emissionProbs[i, (ncol(emissionProbs) - 2 + i):ncol(emissionProbs)] <- 0.2
    emissionProbs[i, 1:(i + 2)] <- 0.2 # 1 : i + 2
  }
  else if (i > ncol(emissionProbs) - 2) {
    emissionProbs[i, (i - 2):ncol(emissionProbs)] <- 0.2 # i - 2 : 10
    # Beginning part of the loop
    emissionProbs[i, 1:(2 - (ncol(emissionProbs) - i))] <- 0.2
  }
  else {
    emissionProbs[i, (i - 2):(i + 2)] <- 0.2 # i - 2 : i + 2
```

```
  }
}

# Initialize HMM
hmm <- initHMM(States=c(1:n_states), Symbols=c(1:n_emissions),
               transProbs=transProbs, emissionProbs=emissionProbs)
```

Question 2

```
# Simulate 100 time steps
n_steps <- 100
sim <- simHMM(hmm, n_steps)

for (i in 1:5) {
  cat("Time ", i, "\nState: ", sim$states[i], "\nObservation: ",
      sim$observation[i], "\n\n")
}
```

```
## Time  1
## State:  1
## Observation:  3
##
## Time  2
## State:  1
## Observation:  2
##
## Time  3
## State:  2
## Observation:  4
##
## Time  4
## State:  2
## Observation:  1
##
## Time  5
## State:  2
## Observation:  4
```

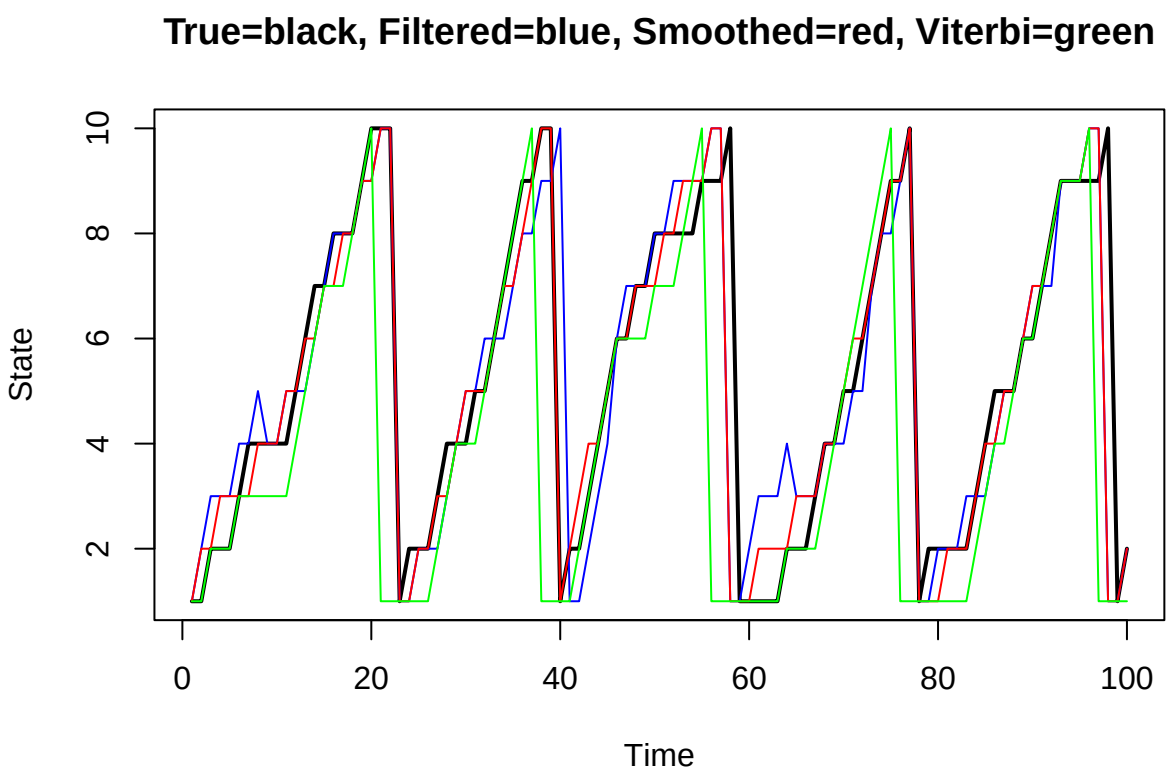
```
cat("...")
```

```
## ...
```

Question 3

```
# Save observations of simulation
obs <- sim$observation
```

```r
compute_paths <- function(hmm, obs) {
  paths <- list()

  # Compute alpha(z_t) and beta(z_t)
  alpha_t <- exp(forward(hmm, obs))
  beta_t <- exp(backward(hmm, obs))

  # Normalize distributions
  filter_distr <- prop.table(alpha_t, margin=2)
  smooth_distr <- prop.table(alpha_t * beta_t, margin=2)

  # Get most likely positions
  paths$filter_path <- apply(filter_distr, 2, which.max)
  paths$smooth_path <- apply(smooth_distr, 2, which.max)
  paths$most_prob_path <- viterbi(hmm, obs)

  return(paths)
}

paths <- compute_paths(hmm, obs)

# Plot paths of each distribution
plot(sim$states, type="l", lwd=2,
     main="True=black, Filtered=blue, Smoothed=red, Viterbi=green",
     xlab="Time", ylab="State")
points(paths$filter_path, type="l", col="blue")
points(paths$smooth_path, type="l", col="red")
points(paths$most_prob_path, type="l", col="green")
```



Question 4

```
# Compute the accuracies of each path versus the states
compute_accuracies <- function(paths, states) {
  filter_acc <- sum(paths$filter_path == states) / length(states)
  smooth_acc <- sum(paths$smooth_path == states) / length(states)
  viterbi_acc <- sum(paths$most_prob_path == states) / length(states)

  cat("Filtered accuracy: ", filter_acc, "\nSmoothed accuracy: ", smooth_acc,
      "\nViterbi accuracy: ", viterbi_acc)
}

compute_accuracies(paths, sim$states)
```

```
## Filtered accuracy:  0.47
## Smoothed accuracy:  0.66
## Viterbi accuracy:  0.46
```

Question 5

```
# Repeat process for different simulations
for (i in 1:5) {
  sim <- simHMM(hmm, n_steps)
  paths <- compute_paths(hmm, sim$observation)

  cat("Iteration ", i, "\n")
  compute_accuracies(paths, sim$states)
  cat("\n\n")
}
```

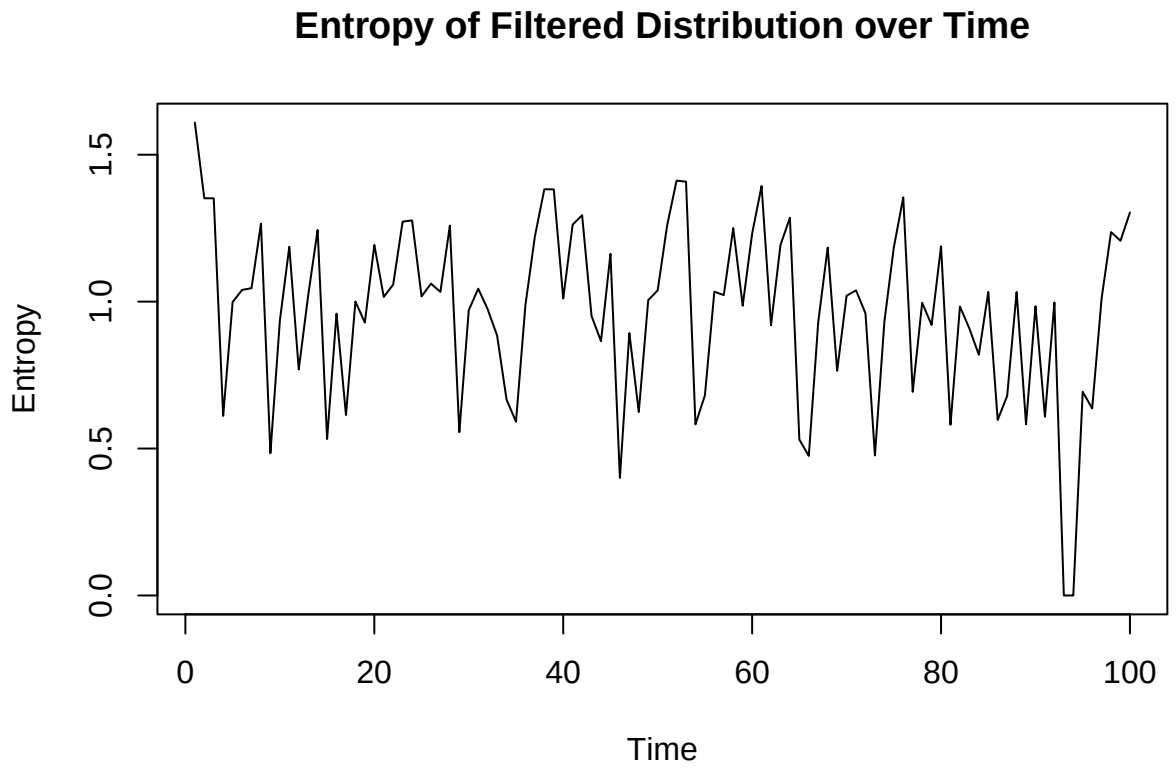
```
## Iteration  1
## Filtered accuracy:  0.57
## Smoothed accuracy:  0.67
## Viterbi accuracy:  0.54
##
## Iteration  2
## Filtered accuracy:  0.45
## Smoothed accuracy:  0.68
## Viterbi accuracy:  0.57
##
## Iteration  3
## Filtered accuracy:  0.53
## Smoothed accuracy:  0.62
## Viterbi accuracy:  0.35
##
## Iteration  4
## Filtered accuracy:  0.6
## Smoothed accuracy:  0.65
## Viterbi accuracy:  0.39
##
## Iteration  5
## Filtered accuracy:  0.64
## Smoothed accuracy:  0.63
## Viterbi accuracy:  0.44
```

The smoothed distributions should be more accurate because it uses more information than the filtered distribution. The filtered distribution only uses the observed data up to time t, whereas the smoothed distribution used the past as well as the future data. Additionally, the smoothed distribution is likely to be more accurate than the most probable path because the most probable path works under the constraint that each predicted state should be valid given the previous state within the model. However, if the previous state was inaccurately predicted, this error may cascade down to future predictions.

Question 6

```
# Compute alpha(z_t) and beta(z_t)
alpha_t <- exp(forward(hmm, obs))
```

```r
# Normalize distributions
filter_distr <- prop.table(alpha_t, margin=2)

# Compute entropy over time
entr <- c()
for (i in 1:ncol(filter_distr)) {
  entr[i] <- entropy.empirical(filter_distr[, i])
}

# Plot entropy
plot(entr, type="l", main="Entropy of Filtered Distribution over Time",
     xlab="Time", ylab="Entropy")
```



When we plot the entropy of the filtered distributions at different times, it becomes clear that the uncertainty does not necessarily decrease later in time.

Question 7

```r
# Save last time step of filter distribution
cond_p_z_T <- filter_distr[, ncol(filter_distr)]
cond_p_z_Tp1 <- c()

for (i in 1:n_states) {
```

```
  p_stay <- cond_p_z_T[i] * 0.5 # Probability of staying in sector

  # Probability of moving to sector
  if (i == 1) {
    p_move <- cond_p_z_T[length(cond_p_z_T)] * 0.5
  }
  else {
    p_move <- cond_p_z_T[i - 1] * 0.5
  }

  # Combine to conditional
  cond_p_z_Tp1[i] <- p_stay + p_move
}

cat("p(z_101 | x_1:100) = \n")
```

```
## p(z_101 | x_1:100) =
```

```
names(cond_p_z_Tp1) <- 1:n_states
cond_p_z_Tp1
```

```
##          1          2          3          4          5          6          7
## 0.23255814 0.34883721 0.26744186 0.08139535 0.00000000 0.00000000 0.00000000
##          8          9         10
## 0.00000000 0.00000000 0.06976744
```