Lab 2

Simon Jorstedt, Simge Cinar, Marijn Jaarsma

2024-09-23

Statement of Contribution

We had mostly the same solutions for problems 1-6, with a slight disagreement on problem 7. For questions 1-6, a combination of our code was used for these solutions.

Problem 1

In this lab we will construct and simulate a Hidden Markov Model where we imagine an agent walking along a cycle of states such that the agent in each time step will stay or move on to the next step with equal probabilities. There are ten states, so the agent can move from state 1 to 2, 2 to 3, etc, and from state 10 to 1, or stay in its current state with equal probability. The hidden states in the model will represent the true position of the agent, while the emissions will be randomly selected from the set $\{i-2, i-1, i, i+1, i+2\}$ when the agent is in state i. For the simulations, we let the agent start in a uniformly randomly selected state. Below we create the programmatic setup such as the transition probability matrix, the emission probability matrix and the HMM. The states are the integers $1, \cdots, 10$, and in this case so are the emission symbols.

```
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(0, 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
  }
}

emissionProbs <- matrix(c(c(0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2),
                          c(0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2),
                          c(0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0),
                          c(0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0, 0.0),
                          c(0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0, 0.0),
                          c(0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0, 0.0),
                          c(0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.0),
                          c(0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2, 0.2),
                          c(0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2, 0.2),
                          c(0.2, 0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 0.2, 0.2)
                          ), nrow = 10, ncol = 10, byrow = TRUE)

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

Problem 2

Now we shall simulate the HMM that we initiated previously. This comes down to just running the `simHMM` function.

```
# 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:  5
## Observation:  4
##
## Time  2
## State:  6
## Observation:  4
##
## Time  3
## State:  7
## Observation:  6
##
## Time  4
## State:  8
## Observation:  6
##
## Time  5
## State:  8
## Observation:  7
```

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

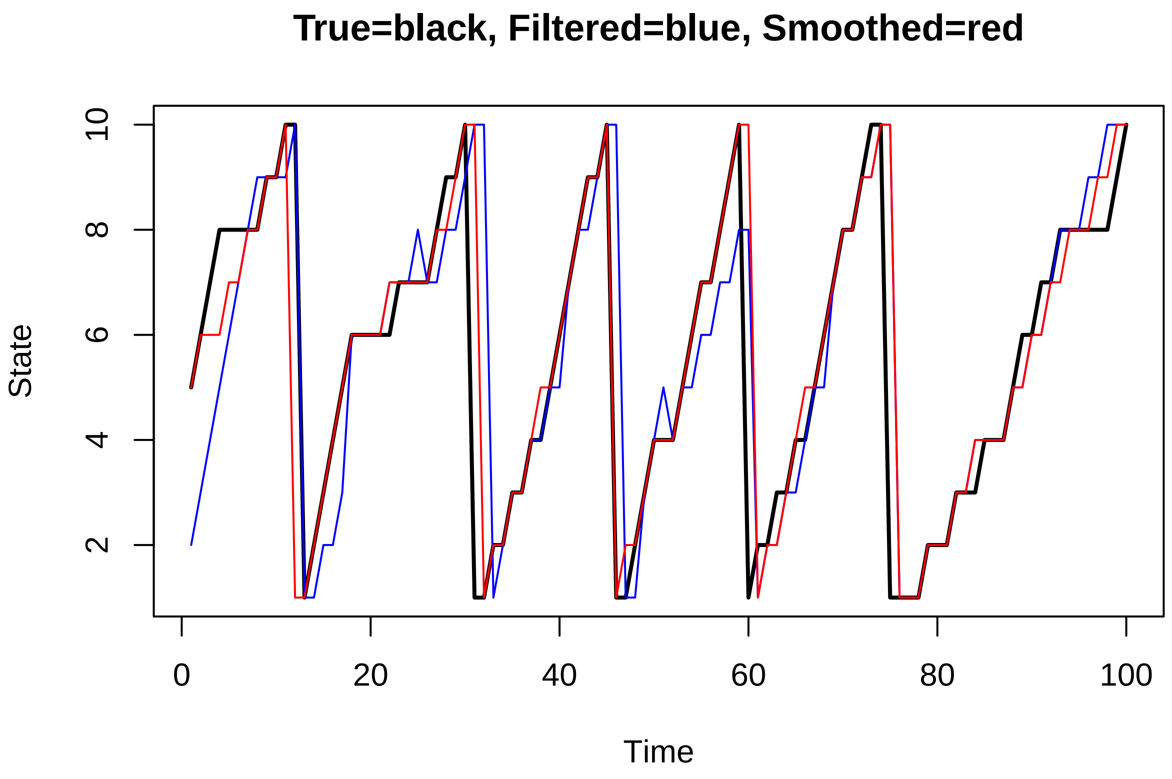
```
## ...
```

Problem 3

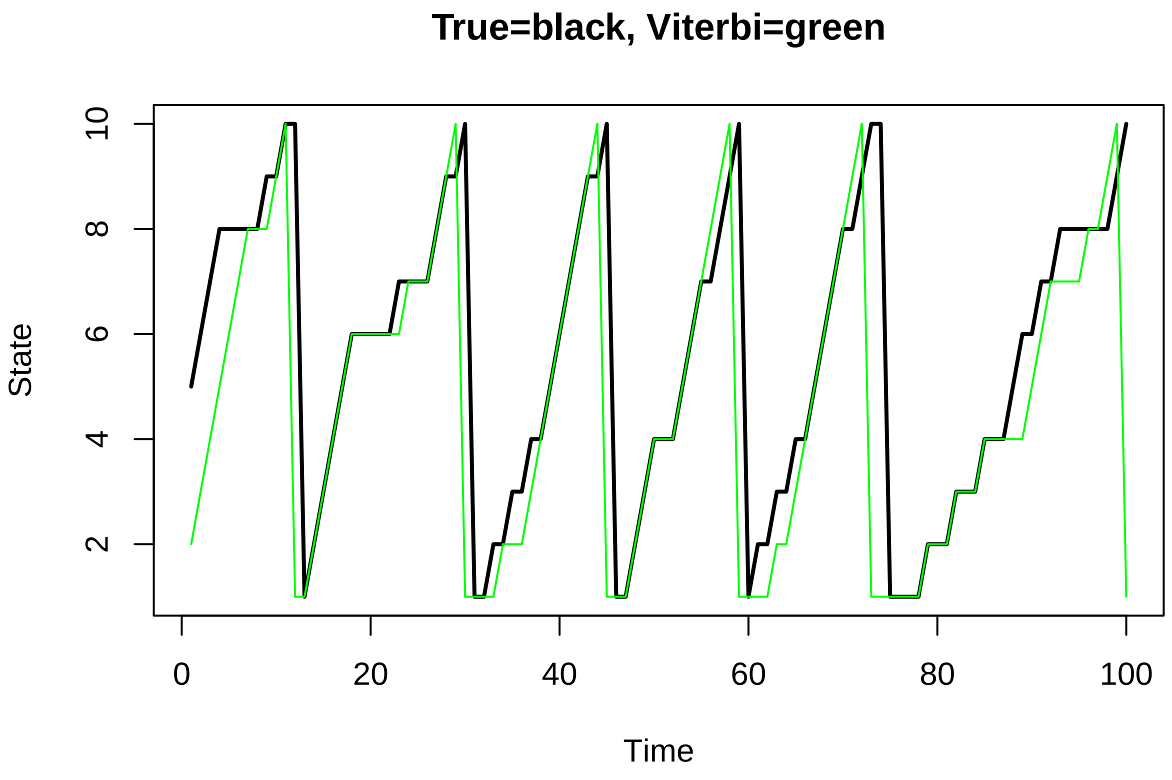
Now we shall use our emission observations above to compute the smoothed and filtered probability distributions using `HMM::backward` and `HMM:forward` respectively. We will also compute the most probable path using the Viterbi algorithm. In Figure 1 below we also visualize the true path (black), the emitted/observed path (blue), and the most probable path (red). We create two custom functions to calculate the most probable "filtered" and "smoothed" states.

```
# 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 true vs smoothed and filtered distributions
plot(sim$states, type="l", lwd=2,
     main="True=black, Filtered=blue, Smoothed=red",
     xlab="Time", ylab="State")
points(paths$filter_path, type="l", col="blue")
points(paths$smooth_path, type="l", col="red")
```



```r
# Plot true vs Viterbi
plot(sim$states, type="l", lwd=2,
     main="True=black, Viterbi=green",
     xlab="Time", ylab="State")
points(paths$most_prob_path, type="l", col="green")
```



Problem 4 and 5

Now compute accuracy of filtered and smoothed probability distributions and of the most probable path.

```r
# 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)
}

# 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")
}
```

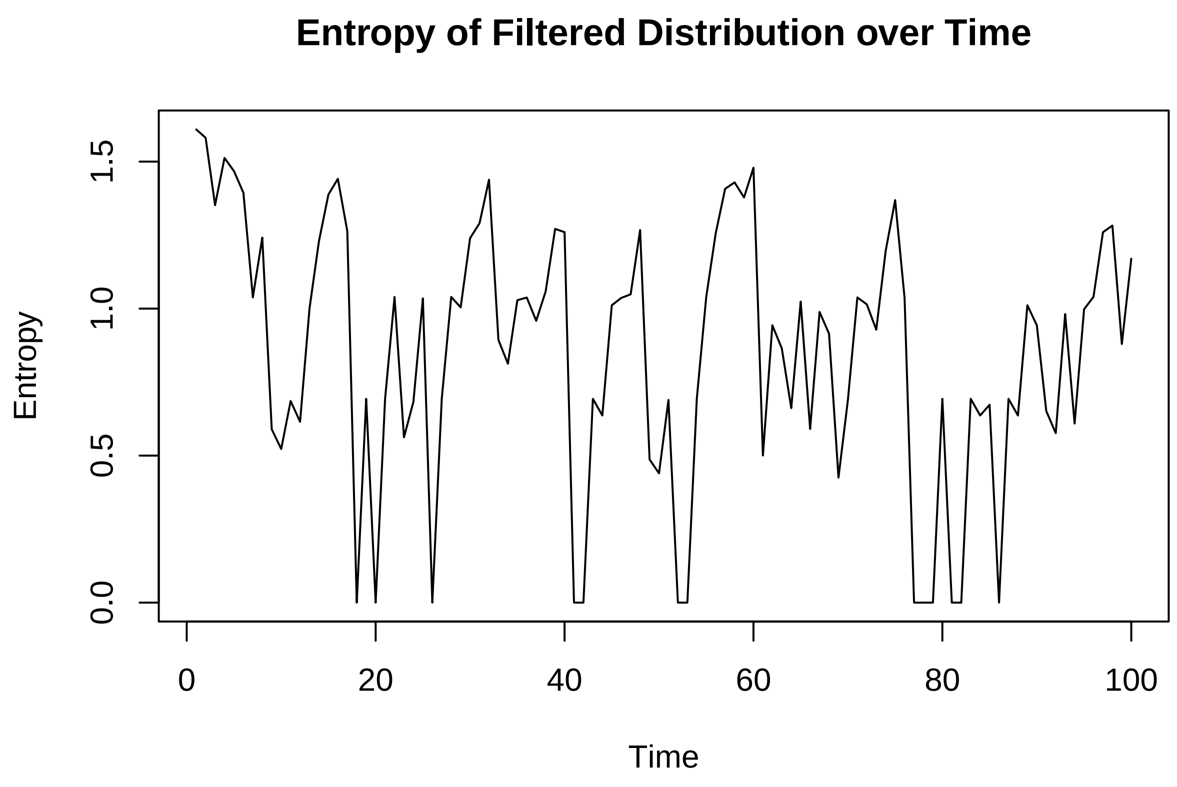
```
## Iteration  1
## Filtered accuracy:  0.6
## Smoothed accuracy:  0.7
## Viterbi accuracy:  0.55
##
## Iteration  2
## Filtered accuracy:  0.51
## Smoothed accuracy:  0.67
## Viterbi accuracy:  0.53
##
## Iteration  3
## Filtered accuracy:  0.53
## Smoothed accuracy:  0.75
## Viterbi accuracy:  0.53
##
## Iteration  4
## Filtered accuracy:  0.59
## Smoothed accuracy:  0.71
## Viterbi accuracy:  0.42
##
## Iteration  5
## Filtered accuracy:  0.62
## Smoothed accuracy:  0.64
## 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.

Problem 6

We hypothesize it to be true that over time, more observations means the accuracy of the estimate of the robots location increases. However, let us investigate this.

```r
# 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)

# 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.

Problem 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
## 0.37660256 0.14615385 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##          8          9         10
## 0.01955128 0.12339744 0.33429487
```

Alternative way:

```
t(hmm$transProbs) %*% filter_distr[,100]
```

```
##
## to         [,1]
##   1  0.37660256
##   2  0.14615385
##   3  0.00000000
##   4  0.00000000
##   5  0.00000000
##   6  0.00000000
##   7  0.00000000
##   8  0.01955128
##   9  0.12339744
##   10 0.33429487
```