# LAB 2: HIDDEN MARKOV MODELS

xiali125@student.liu.se linfr259@student.liu.se qinzh916@student.liu.se
huali824@student.liu.se

2025-09-21

## Contents



## Introduction

This report details the process of modeling a robot's movement on a circular track using a Hidden Markov Model (HMM). The robot's true position is a hidden state, while a noisy tracking device provides observations. The objective is to use the HMM framework to simulate the robot's path, infer its location using filtering and smoothing algorithms, and evaluate the performance of these methods.

## Analysis and Results

This section addresses the seven questions from the assignment, presenting the results generated by the R code and providing interpretations for each step.

## Question 1: Build an HMM

**Answer:**
A Hidden Markov Model was constructed to represent the scenario. The model consists of 10 hidden states (the robot's true location) and 10 observation symbols (the tracker's report). The initial state probabilities are uniform (0.1 for each state).

The transition matrix reflects a 50% chance of staying in the current sector and a 50% chance of moving to the next, with wrap-around from sector 10 to 1.

The emission matrix assigns a uniform probability of 0.2 to the five sectors in the range `[i-2, i+2]` for any true state `i`, also accounting for the circular track.

The structure of the initialized HMM is printed below.

```
# The code from your script for Q1 is executed here
library(HMM)

states <- 1:10
symbols <- 1:10
num_states <- length(states)
num_symbols <- length(symbols)

startProbs <- rep(1/num_states, num_states)

transProbs <- matrix(0, nrow = num_states, ncol = num_states)
for (i in 1:num_states) {
  transProbs[i, i] <- 0.5
  next_state <- ifelse(i == num_states, 1, i + 1)
  transProbs[i, next_state] <- 0.5
}

emissionProbs <- matrix(0, nrow = num_states, ncol = num_symbols)
for (i in 1:num_states) {
  observation_range <- (i - 2):(i + 2)
  wrapped_range <- (observation_range - 1) %% num_symbols + 1
  for (j in wrapped_range) {
    emissionProbs[i, j] <- 0.2
  }
}

hmm <- initHMM(States = states,
               Symbols = symbols,
               startProbs = startProbs,
               transProbs = transProbs,
               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
```

## Question 2: Simulate the HMM

**Answer:**
The constructed HMM was used to simulate a path of 100 time steps. This process generates both the sequence of true hidden states (the robot's actual path) and the corresponding sequence of observations (the noisy tracker reports). An example of the first 10 steps from the simulation is shown below.

```
set.seed(123)
simulation <- simHMM(hmm, length = 100)

cat("Example of first 10 true states:\n")
```

```
## Example of first 10 true states:
```

```
print(simulation$states[1:10])
```

```
##  [1]  4  5  6  7  8  8  9  9  9 10
```

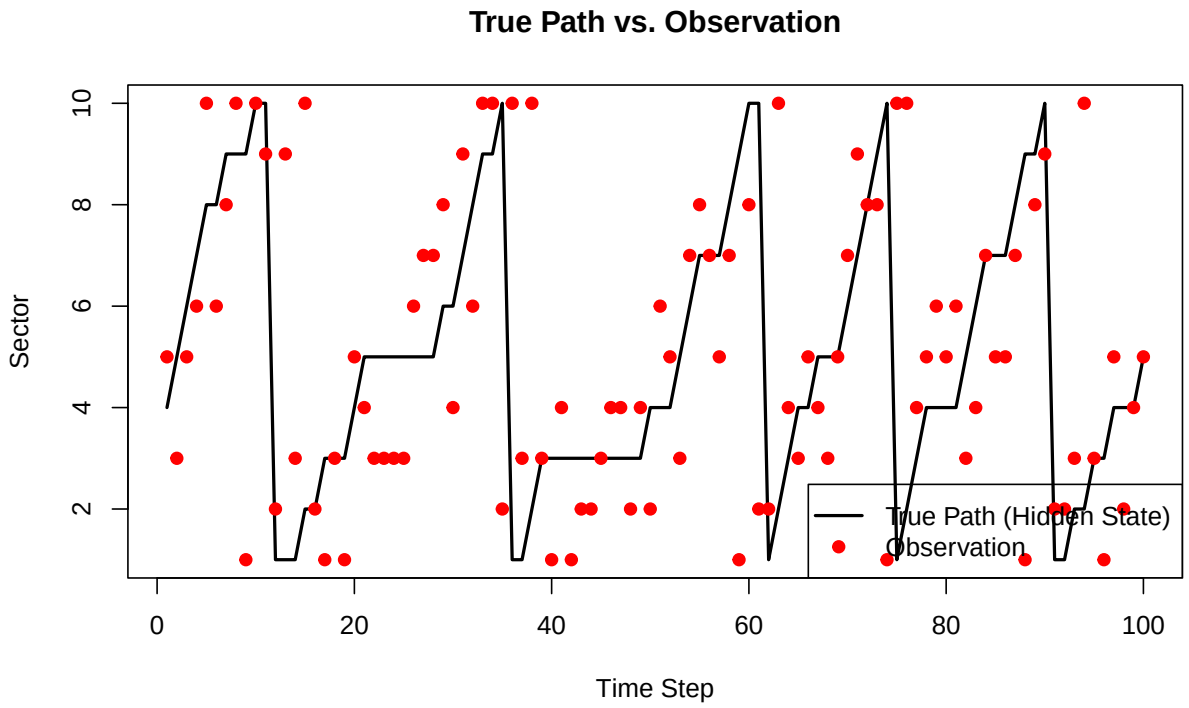
```r
cat("Corresponding first 10 observations:\n")
```

```
## Corresponding first 10 observations:
```

```r
print(simulation$observation[1:10])
```

```
##  [1]  5  3  5  6 10  6  8 10  1 10
```

**Visualization of True Path vs. Observation**




## Question 3: Perform Inference

**Answer:**
Using only the observation sequence from the simulation, we performed three types of inference:

- **Filtering:** Calculated the probability distribution of the robot's location at each time step, given observations up to that point.
- **Smoothing:** Calculated a refined probability distribution for each time step, given the entire sequence of 100 observations.
- **Most Probable Path:** Determined the single most likely sequence of states for the entire 100 steps using the Viterbi algorithm.

The most likely states inferred by each method for the first 10 time steps are displayed below for comparison.

```r
observations <- simulation$observation
true_states <- simulation$states

fw <- forward(hmm, observations)
filteredProbs <- apply(fw, 2, function(x) {
  probs <- exp(x - max(x))
  probs / sum(probs)
})

smoothedProbs <- posterior(hmm, observations)
viterbiPath <- viterbi(hmm, observations)

cat("Most likely states from filtered estimates (first 10 steps):\n")
```

```
## Most likely states from filtered estimates (first 10 steps):
```

```r
print(apply(filteredProbs[, 1:10], 2, which.max))
```

```
##  1  2  3  4  5  6  7  8  9 10
##  3  4  5  5  8  8  8  9  9 10
```

```r
cat("Most likely states from smoothed estimates (first 10 steps):\n")
```

```
## Most likely states from smoothed estimates (first 10 steps):
```

```r
print(apply(smoothedProbs[, 1:10], 2, which.max))
```

```
##  1  2  3  4  5  6  7  8  9 10
##  4  5  6  7  8  8  8  9  9 10
```

```r
cat("Viterbi path (first 10 steps):\n")
```

```
## Viterbi path (first 10 steps):
```

```r
print(viterbiPath[1:10])
```

```
##  [1]  4  5  6  7  8  8  9 10  1  1
```

## Question 4: Calculate Accuracy

**Answer:**
The accuracy of each inference method was calculated by comparing its guessed state at each time point to the true hidden state. For filtering and smoothing, the guessed state was taken as the one with the highest probability at each time step. The Viterbi path provides a direct sequence of guessed states. The results for the first simulation are as follows.

```r
filteredStates <- apply(filteredProbs, 2, which.max)
acc_filtered <- mean(filteredStates == true_states)

smoothedStates <- apply(smoothedProbs, 2, which.max)
acc_smoothed <- mean(smoothedStates == true_states)

acc_viterbi <- mean(viterbiPath == true_states)

cat("Q4: Accuracy Results\n")
```

```
## Q4: Accuracy Results
```

```r
cat(sprintf("Filtered distribution accuracy: %.2f%%\n", acc_filtered * 100))
```

```
## Filtered distribution accuracy: 53.00%
```

```r
cat(sprintf("Smoothed distribution accuracy: %.2f%%\n", acc_smoothed * 100))
```

```
## Smoothed distribution accuracy: 64.00%
```

```r
cat(sprintf("Viterbi path accuracy: %.2f%%\n", acc_viterbi * 100))
```

```
## Viterbi path accuracy: 36.00%
```

## Question 5: Repeat with a New Sample and Explain

**Answer:**
The exercise was repeated with a new, different simulated sample to ensure the conclusions were not based on a single, potentially anomalous result. The accuracies for the new sample are shown below, followed by a theoretical explanation of the expected performance differences.

```r
set.seed(321)
simulation2 <- simHMM(hmm, length = 100)
observations2 <- simulation2$observation
true_states2 <- simulation2$states

fw2 <- forward(hmm, observations2)
filteredProbs2 <- apply(fw2, 2, function(x) {
  probs <- exp(x - max(x))
  probs / sum(probs)
})

smoothedProbs2 <- posterior(hmm, observations2)
viterbiPath2 <- viterbi(hmm, observations2)

filteredStates2 <- apply(filteredProbs2, 2, which.max)
smoothedStates2 <- apply(smoothedProbs2, 2, which.max)

acc_filtered2 <- mean(filteredStates2 == true_states2)
```

```r
acc_smoothed2 <- mean(smoothedStates2 == true_states2)
acc_viterbi2 <- mean(viterbiPath2 == true_states2)

cat("Accuracy on a new simulated sample\n")
```

```
## Accuracy on a new simulated sample
```

```r
cat(sprintf("Filtered distribution accuracy: %.2f%%\n", acc_filtered2 * 100))
```

```
## Filtered distribution accuracy: 61.00%
```

```r
cat(sprintf("Smoothed distribution accuracy: %.2f%%\n", acc_smoothed2 * 100))
```

```
## Smoothed distribution accuracy: 66.00%
```

```r
cat(sprintf("Viterbi path accuracy: %.2f%%\n", acc_viterbi2 * 100))
```

```
## Viterbi path accuracy: 46.00%
```

**Theoretical Explanation**

**Why is smoothing more accurate than filtering?**
As is illustrated by the accuracies above, the smoothed distributions are generally more accurate than the filtered distributions. This is because the smoothed distributions is based on the whole time series (future and past), providing a more comprehensive view of the hidden states. In contrast, the filtered distributions only consider past observations, which may lead to less accurate estimates.

**Why is smoothing's point-wise accuracy generally higher than Viterbi's?**
Moreover, the smoothed distributions also tend to be more accurate than the most probable paths. For each time step, during the computation with Viterbi algorithm, a single best estimation of the hidden states is forced to be chosen, even if the evidence is ambiguous. While for the smoothed distributions, they provide a full posteriro distribution over all hidden states at each time step based on all observations, allowing for a better understanding of the uncertainty in the state estimates.

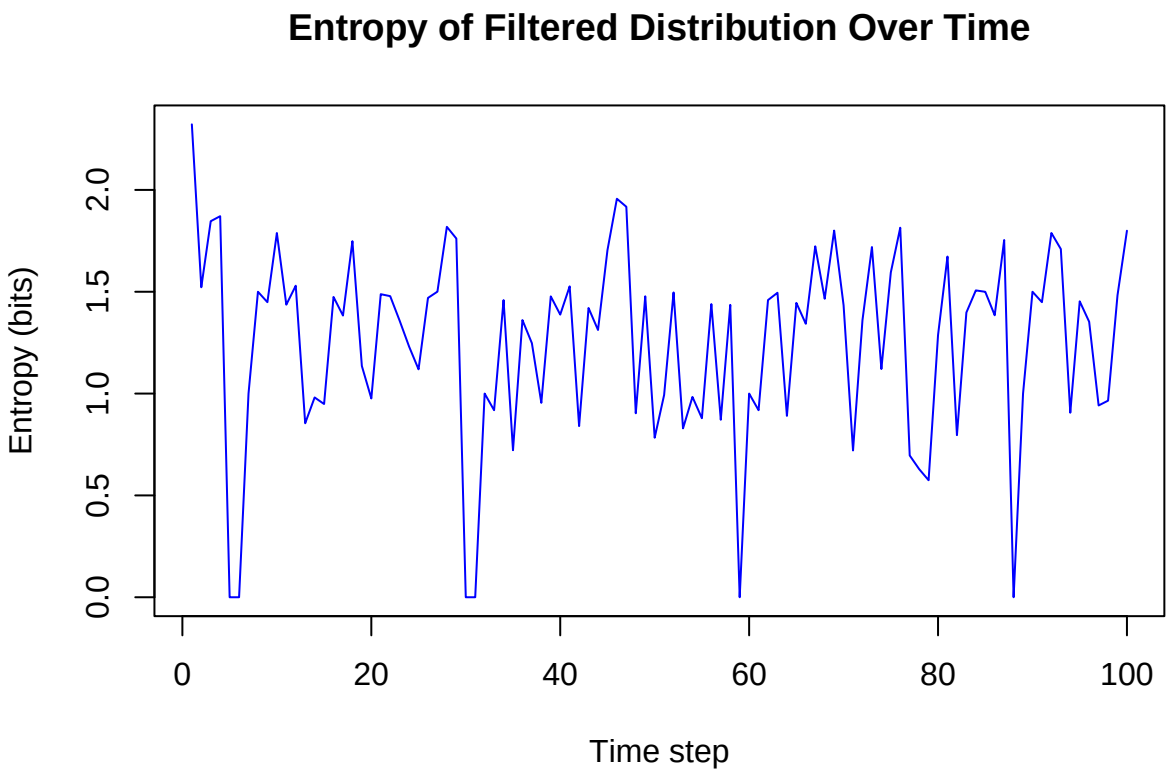
## Question 6: Uncertainty Over Time

**Answer:**
The statement "the more observations you have received, the better you know where the robot is" is generally true but not always. We can measure the model's uncertainty at each time step by calculating the entropy of the filtered probability distribution. The plot below shows this entropy over time.

```r
library(entropy)

filteredEntropy <- apply(filteredProbs, 2, function(p) {
  entropy.empirical(p, unit = "log2")
})

plot(filteredEntropy, type = "l", col = "blue",
     xlab = "Time step", ylab = "Entropy (bits)",
     main = "Entropy of Filtered Distribution Over Time")
```




Empirical entropy has been proposed to measure uncertainty for the filtered distributions, where higher entropy means more uncertainty. As is shown in the plot above, the entropy of the filtered distributions fluctuates over time, indicating that it is not always true that the later in time, the better we know where the robot is. The reason is that the robot's movement is stochastic, and the observations are noisy, leading to varying levels of uncertainty at different time points.

## Question 7: Predict the Next Step

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

The predicted distribution for time 101 is given by

$$p_{101} = p_{100}\, A$$

where $p_{100}$ is the filtered distribution at time 100 and $A$ is the transition matrix.

Element-wise, for each next-state index $j$:

$$p_{101}[j] = \sum_{i=1}^{N} p_{100}[i] \cdot A_{ij}$$

```
filtered_t100 <- filteredProbs[, 100]

pred_t101 <- as.vector(filtered_t100 %*% transProbs)
```

```r
cat("Predicted state distribution for time step 101:\n")
```

```
## Predicted state distribution for time step 101:
```

```r
print(round(pred_t101, 4))
```

```
##  [1] 0.0000 0.0000 0.0488 0.2227 0.3750 0.2773 0.0762 0.0000 0.0000 0.0000
```

```r
cat(sprintf("\nThe most likely state at time step 101 is: %d\n", which.max(pred_t101)))
```

```
##
## The most likely state at time step 101 is: 5
```

## Contritbutions

This report was completed through the joint efforts of team members: Xiaochen, Linn, Qinxia, and Huaide. We collectively tackled the core concepts of Hidden Markov Models through group discussions, coded and debugged the R script as a team, and analyzed the experimental results together. The writing and final review of this document were also a result of our collaborative work, ensuring the consistency and quality of the final report.

## Appendix: Complete R Code

This appendix contains the complete, unaltered R script used to generate all the results in this report.

```r
# install.packages('HMM')
library(HMM)

# Define model parameters
states <- 1:10
symbols <- 1:10
num_states <- length(states)
num_symbols <- length(symbols)

# Define initial state probabilities
startProbs <- rep(1/num_states, num_states)

# Define transition probability matrix
transProbs <- matrix(0, nrow = num_states, ncol = num_states)
for (i in 1:num_states) {
  transProbs[i, i] <- 0.5
  next_state <- ifelse(i == num_states, 1, i + 1)
  transProbs[i, next_state] <- 0.5
}

# Define emission probability matrix (uniform over [i-2, i+2])
emissionProbs <- matrix(0, nrow = num_states, ncol = num_symbols)
for (i in 1:num_states) {
  observation_range <- (i - 2):(i + 2)
```

```
  wrapped_range <- (observation_range - 1) %% num_symbols + 1
  for (j in wrapped_range) {
    emissionProbs[i, j] <- 0.2
  }
}

hmm <- initHMM(States = states,
               Symbols = symbols,
               startProbs = startProbs,
               transProbs = transProbs,
               emissionProbs = emissionProbs)

print(hmm)


# --- Q2: Simulate a path from the HMM ---

set.seed(123)
simulation <- simHMM(hmm, length = 100)

# View results
cat("Example of first 10 true states:\n")
print(simulation$states[1:10])
cat("Corresponding first 10 observations:\n")
print(simulation$observation[1:10])

time_steps <- 1:length(simulation$states)
plot(time_steps, simulation$states, type = "l", col = "black",lwd = 2,
     main = "True Path vs. Observation",xlab = "Time Step",
     ylab = "Sector",ylim = c(1, 10))
points(time_steps, simulation$observation, col = "red",pch = 19)
legend("bottomright", legend = c("True Path (Hidden State)", "Observation"),
       col = c("black", "red"), lty = c(1, NA),pch = c(NA, 19),lwd = c(2, NA))


# --- Q3: Inference from observations ---

observations <- simulation$observation
true_states <- simulation$states

# Filtering Distribution
fw <- forward(hmm, observations)
filteredProbs <- apply(fw, 2, function(x) {
  probs <- exp(x - max(x))
  probs / sum(probs)
})

# Smoothed Distribution
smoothedProbs <- posterior(hmm, observations)

# Most Probable Path
viterbiPath <- viterbi(hmm, observations)
```

```r
# Check of the results
cat("Most likely states from filtered estimates (first 10 steps):\n")
print(apply(filteredProbs[, 1:10], 2, which.max))

cat("Most likely states from smoothed estimates (first 10 steps):\n")
print(apply(smoothedProbs[, 1:10], 2, which.max))

cat("Viterbi path (first 10 steps):\n")
print(viterbiPath[1:10])


# --- Q4: Calculate Accuracy ---

# Filtered Accuracy
filteredStates <- apply(filteredProbs, 2, which.max)
acc_filtered <- mean(filteredStates == true_states)

# Smoothed Accuracy
smoothedStates <- apply(smoothedProbs, 2, which.max)
acc_smoothed <- mean(smoothedStates == true_states)

# Viterbi Accuracy
acc_viterbi <- mean(viterbiPath == true_states)

cat("Q4: Accuracy Results\n")
cat(sprintf("Filtered distribution accuracy: %.2f%%\n", acc_filtered * 100))
cat(sprintf("Smoothed distribution accuracy: %.2f%%\n", acc_smoothed * 100))
cat(sprintf("Viterbi path accuracy: %.2f%%\n", acc_viterbi * 100))


# --- Q5: Experiment with a new simulated sample ---

set.seed(321) # Use a different seed for a new sample

simulation2 <- simHMM(hmm, length = 100)
observations2 <- simulation2$observation
true_states2 <- simulation2$states

# Recalculate
fw2 <- forward(hmm, observations2)
filteredProbs2 <- apply(fw2, 2, function(x) {
  probs <- exp(x - max(x))
  probs / sum(probs)
})

smoothedProbs2 <- posterior(hmm, observations2)
viterbiPath2 <- viterbi(hmm, observations2)

# Accuracy for the three methods
filteredStates2 <- apply(filteredProbs2, 2, which.max)
smoothedStates2 <- apply(smoothedProbs2, 2, which.max)

acc_filtered2 <- mean(filteredStates2 == true_states2)
```

```r
acc_smoothed2 <- mean(smoothedStates2 == true_states2)
acc_viterbi2 <- mean(viterbiPath2 == true_states2)

cat("Accuracy on a new simulated sample\n")
cat(sprintf("Filtered distribution accuracy: %.2f%%\n", acc_filtered2 * 100))
cat(sprintf("Smoothed distribution accuracy: %.2f%%\n", acc_smoothed2 * 100))
cat(sprintf("Viterbi path accuracy: %.2f%%\n", acc_viterbi2 * 100))



# --- Q6: Entropy over time ---
# install.packages("entropy")
library(entropy)

# Calculate the entropy for each time step's filtered distribution
filteredEntropy <- apply(filteredProbs, 2, function(p) {
  entropy.empirical(p, unit = "log2")
})

# Plot entropy over time
plot(filteredEntropy, type = "l", col = "blue",
     xlab = "Time step", ylab = "Entropy (bits)",
     main = "Entropy of Filtered Distribution Over Time")


# --- Q7: Predict state for time step 101 ---
# Get the filtered distribution at the last step (t=100)
filtered_t100 <- filteredProbs[, 100]

# Predict the state distribution for t=101 using the transition matrix
pred_t101 <- as.vector(filtered_t100 %*% transProbs)

# View the result
cat("Predicted state distribution for time step 101:\n")
print(round(pred_t101, 4))
cat(sprintf("\nThe most likely state at time step 101 is: %d\n", which.max(pred_t101)))
```