Lab2-Personal report

Arash

2023-09-16

Attaching The Packages and Loading the Data

First, the required packages are attached.

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

Question 1

First, I created the transition and emission probability of Hidden Markov Model and at the end we create HMM model:

```
n_states <- 10
states <- 1:n_states
symbols <- letters[states]
start_probs <- rep(1/n_states, n_states)

transition_probs <- diag(0.5, 10)
for(i in seq_len(n_states)){
  j <- ifelse(i+1 <= n_states, i+1, 1)
  transition_probs[i, j] <- 0.5
}


emission_probs <- matrix(0, nrow = 10, ncol = 10)
for(i in seq_len(n_states)){
  j <- (i-2):(i+2)
  j[j <= 0] <- 10 + j[j <= 0]
  j[j > 10] <- j[j > 10] - 10
  emission_probs[i, j] <- 0.2
}





robot_HMM <- initHMM(states, symbols, start_probs, transition_probs, emission_probs)
```

Question 2

Now, we sample 100 observations from the HMM.

```
set.seed(1234)
robot_sim <- simHMM(robot_HMM, 100)
```

Question 3

To calculate filtering and smoothing probability distributions, we need alpha and beta tables where we can get them with `forward()` and `backward()`.

$$Filtering = \ p(z_t|x_{0:t}) = \frac{\alpha(z_t)}{\sum_{z_t}\alpha(z_t)} Smoothing = \ p(z_t|x_{0:T}) = \frac{\alpha(z_t)\beta(z_t)}{\sum_{z_t}\alpha(z_t)\beta(z_t)}$$

For getting the most probable path, we can use `viterbi()` algorithm.

```
robot_observations <- robot_sim$observation

alpha <- exp(forward(robot_HMM, robot_observations))
beta <- exp(backward(robot_HMM, robot_observations))

smoothing_dist <- prop.table(alpha * beta, 2)
filtering_dist <- prop.table(alpha, 2)

likely_path <- viterbi(robot_HMM, robot_observations)
```

Question 4

To get the most likely path using filtering and smoothing distributions, we can get the index of the variable that has the highest probability at each time step.

```
smoothing_path <- apply(smoothing_dist, 2, which.max)
filtering_path <- apply(filtering_dist, 2, which.max)

table(smoothing_path, robot_sim$states)
```

```
##
## smoothing_path  1  2  3  4  5  6  7  8  9 10
##             1   6  0  0  0  0  0  0  0  0  2
##             2   2  5  1  0  0  0  0  0  0  0
##             3   0  0 12  3  0  0  0  0  0  0
##             4   0  0  0  7  4  0  0  0  0  0
##             5   0  0  0  2  7  2  0  0  0  0
##             6   0  0  0  0  1  8  1  0  0  0
##             7   0  0  0  0  0  0  8  1  0  0
##             8   0  0  0  0  0  0  0  8  1  0
##             9   0  0  0  0  0  0  0  3  7  0
##             10  0  0  0  0  0  0  0  0  2  7
```

```
mean(smoothing_path == robot_sim$states)
```

```
## [1] 0.75
```

```
table(filtering_path, robot_sim$states)
```

```
##
## filtering_path 1 2 3 4 5 6 7 8 9 10
##             1  6 2 1 0 0 0 0 0 0  0
##             2  1 3 3 1 0 0 0 0 0  0
##             3  0 0 4 4 1 0 0 0 0  0
##             4  0 0 2 7 3 0 0 0 0  0
##             5  0 0 3 0 7 2 0 0 0  0
##             6  0 0 0 0 1 8 2 0 0  0
```

```
##             7  0 0 0 0 0 0 7 2 0  0
##             8  0 0 0 0 0 0 0 8 1  0
##             9  0 0 0 0 0 0 0 2 6  2
##             10 1 0 0 0 0 0 0 0 3  7
```

```r
mean(filtering_path == robot_sim$states)
```

```
## [1] 0.63
```

```r
table(likely_path, robot_sim$states)
```

```
##
## likely_path 1 2 3 4 5 6 7 8 9 10
##          1  8 3 0 0 0 0 0 0 1  7
##          2  0 2 6 0 0 0 0 0 0  0
##          3  0 0 7 6 3 0 0 0 0  0
##          4  0 0 0 6 4 0 0 0 0  0
##          5  0 0 0 0 5 7 0 0 0  0
##          6  0 0 0 0 0 3 5 1 0  0
##          7  0 0 0 0 0 0 4 1 0  0
##          8  0 0 0 0 0 0 0 8 2  0
##          9  0 0 0 0 0 0 0 2 4  0
##          10 0 0 0 0 0 0 0 0 3  2
```

```r
mean(likely_path == robot_sim$states)
```

```
## [1] 0.49
```

The accuracy of the smoothing is 0.75, The accuracy of the filtering is 0.63 and the The accuracy of the viterbi is 0.49.

Depending on the runs, the accuracy of the smoothing is higher than filtering and viterbi.

```r
set.seed(12345)
robot_sim <- simHMM(robot_HMM, 100)
robot_observations <- robot_sim$observation

alpha <- exp(forward(robot_HMM, robot_observations))
beta <- exp(backward(robot_HMM, robot_observations))

smoothing_dist <- prop.table(alpha * beta, 2)
filtering_dist <- prop.table(alpha, 2)
likely_path <- viterbi(robot_HMM, robot_observations)

smoothing_path <- apply(smoothing_dist, 2, which.max)
filtering_path <- apply(filtering_dist, 2, which.max)

table(smoothing_path, robot_sim$states)
```

```
##
## smoothing_path  1  2  3  4  5  6  7  8  9 10
##             1   9  0  0  0  0  0  0  0  0  1
##             2   1 10  1  0  0  0  0  0  0  0
##             3   0  3  7  0  0  0  0  0  0  0
##             4   0  0  2  9  1  0  0  0  0  0
##             5   0  0  0  3  5  0  0  0  0  0
##             6   0  0  0  0  3  5  0  0  0  0
##             7   0  0  0  0  0  1  5  0  0  0
```

```
##             8    0   0   0   0   0   0   1   9   3   0
##             9    0   0   0   0   0   0   0   2   6   2
##             10   1   0   0   0   0   0   0   0   1   9
```

```r
mean(smoothing_path == robot_sim$states)
```

```
## [1] 0.74
```

```r
table(filtering_path, robot_sim$states)
```

```
##
## filtering_path 1 2 3 4 5 6 7 8 9 10
##             1  8 1 0 0 0 0 0 0 0  4
##             2  2 7 1 0 0 0 0 0 0  0
##             3  0 4 3 1 0 0 0 0 0  0
##             4  0 1 5 7 2 0 0 0 0  0
##             5  0 0 1 3 5 2 0 0 1  0
##             6  0 0 0 1 2 3 3 0 0  0
##             7  0 0 0 0 0 1 3 1 0  0
##             8  0 0 0 0 0 0 0 8 4  0
##             9  0 0 0 0 0 0 0 2 4  3
##             10 1 0 0 0 0 0 0 0 1  5
```

```r
mean(filtering_path == robot_sim$states)
```

```
## [1] 0.53
```

```r
table(likely_path, robot_sim$states)
```

```
##
## likely_path  1  2  3  4  5  6  7  8  9 10
##          1  11  6  1  0  0  0  0  0  2  8
##          2   0  7  2  0  0  0  0  0  0  0
##          3   0  0  7  6  1  0  0  0  0  0
##          4   0  0  0  6  4  1  0  0  0  0
##          5   0  0  0  0  4  1  0  0  0  0
##          6   0  0  0  0  0  3  1  0  0  0
##          7   0  0  0  0  0  1  3  1  0  0
##          8   0  0  0  0  0  0  2  8  1  0
##          9   0  0  0  0  0  0  0  2  3  0
##          10  0  0  0  0  0  0  0  0  4  4
```

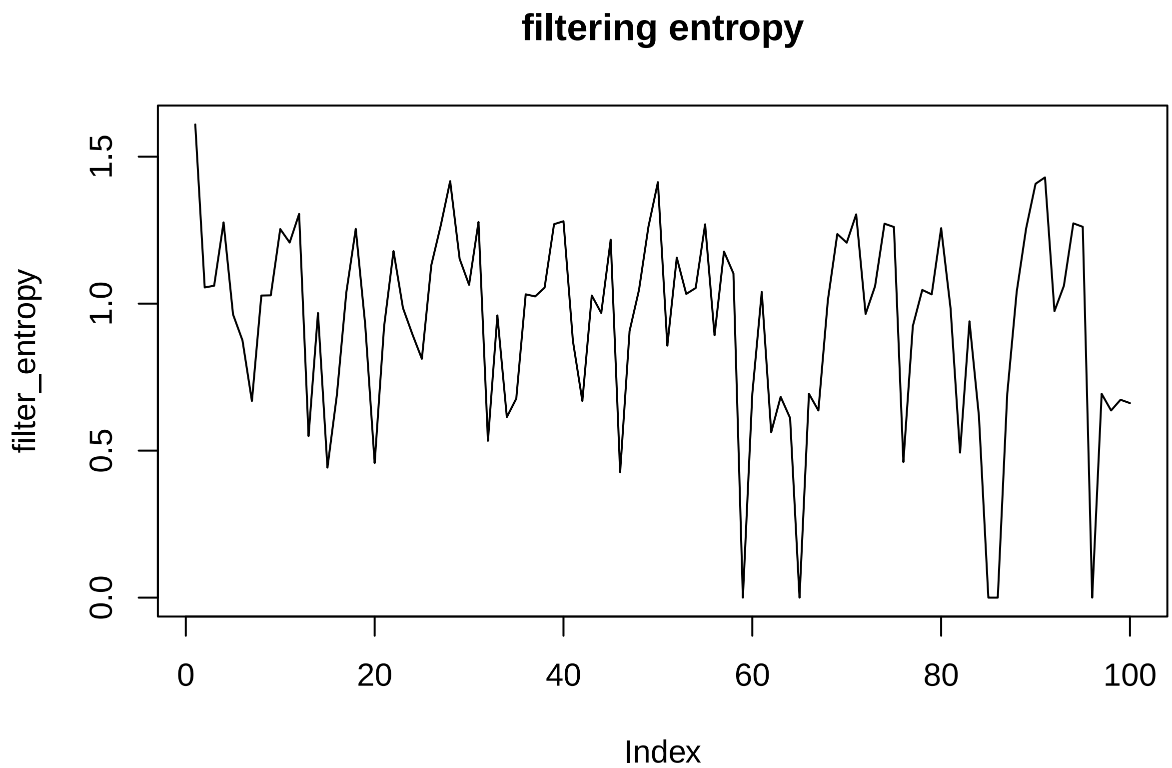
```r
mean(likely_path == robot_sim$states)
```

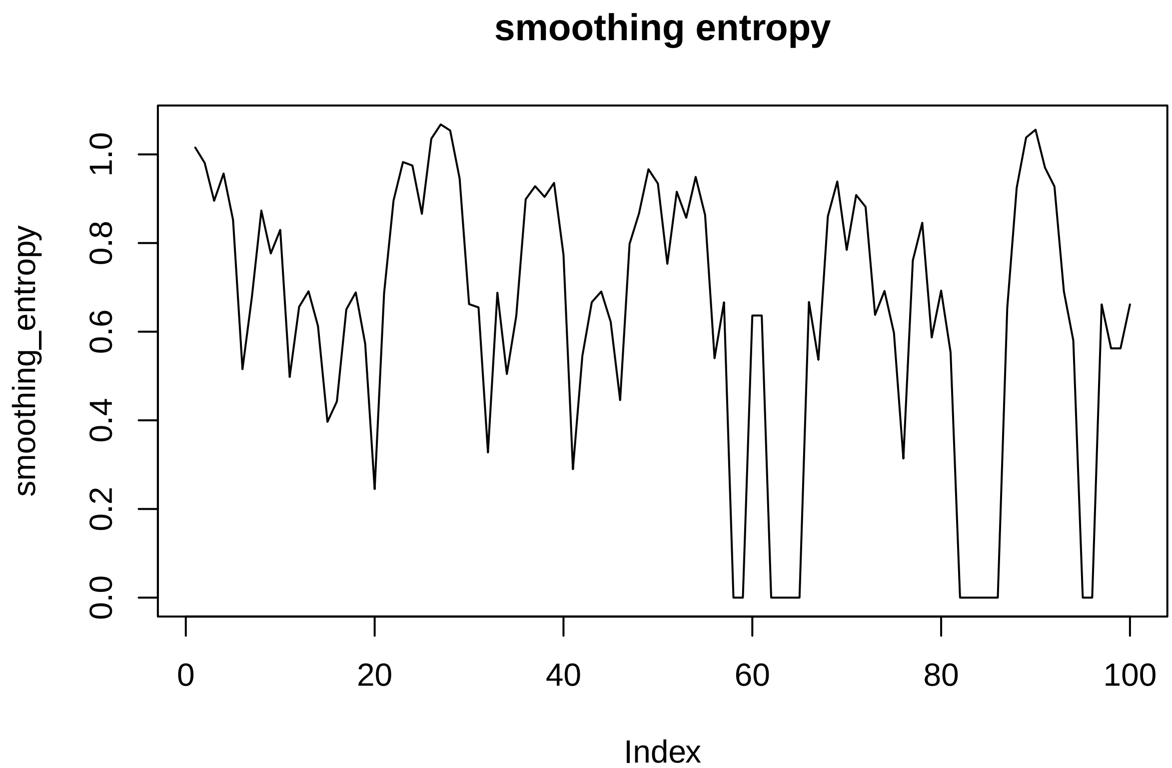
```
## [1] 0.56
```

Smoothing has the highest accuracy since it has the information both from the past and the future of z_t to do inference on value of z_t. The case is also is true for Viterbi result. The algorithm maximizes the value of z_t given all the past values of $z_{1:t-1}$ and $z_{1:t-1}$.

Question 6

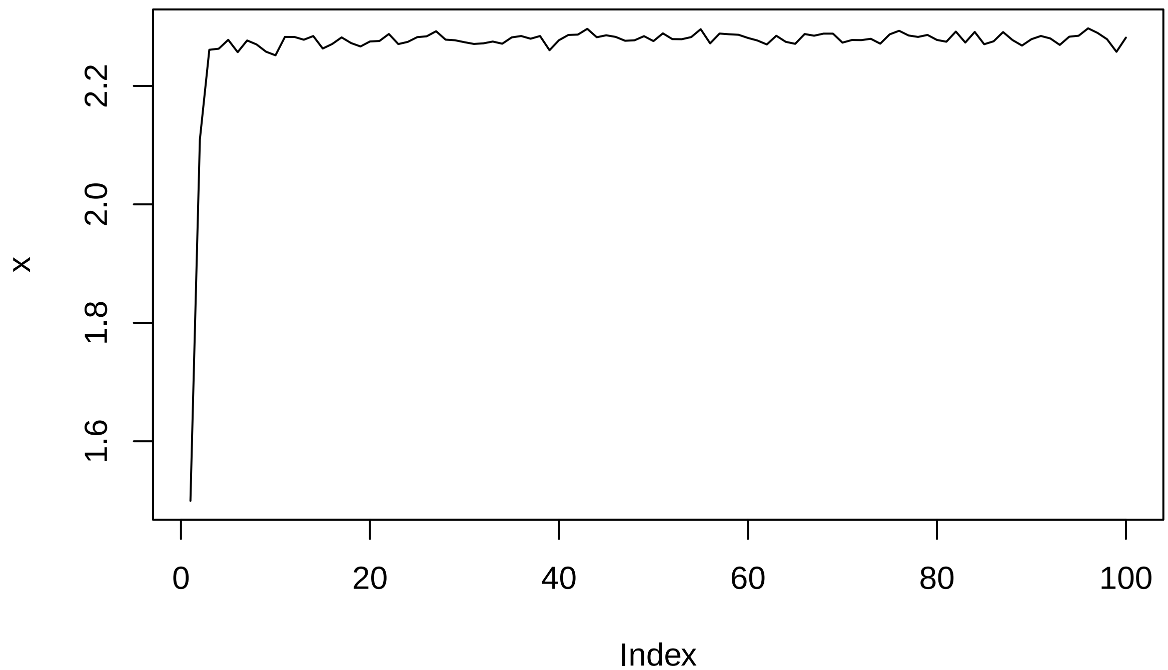
```r
filter_entropy <- apply(filtering_dist, 2, entropy.empirical)
plot(filter_entropy, type = "l", main = "filtering entropy")

smoothing_entropy <- apply(smoothing_dist, 2, entropy.empirical)
plot(smoothing_entropy, type = "l", main = "smoothing entropy")
```


filtering entropy
filter_entropy
0.0
0.5
1.0
1.5
0
20
40
60
80
100
Index

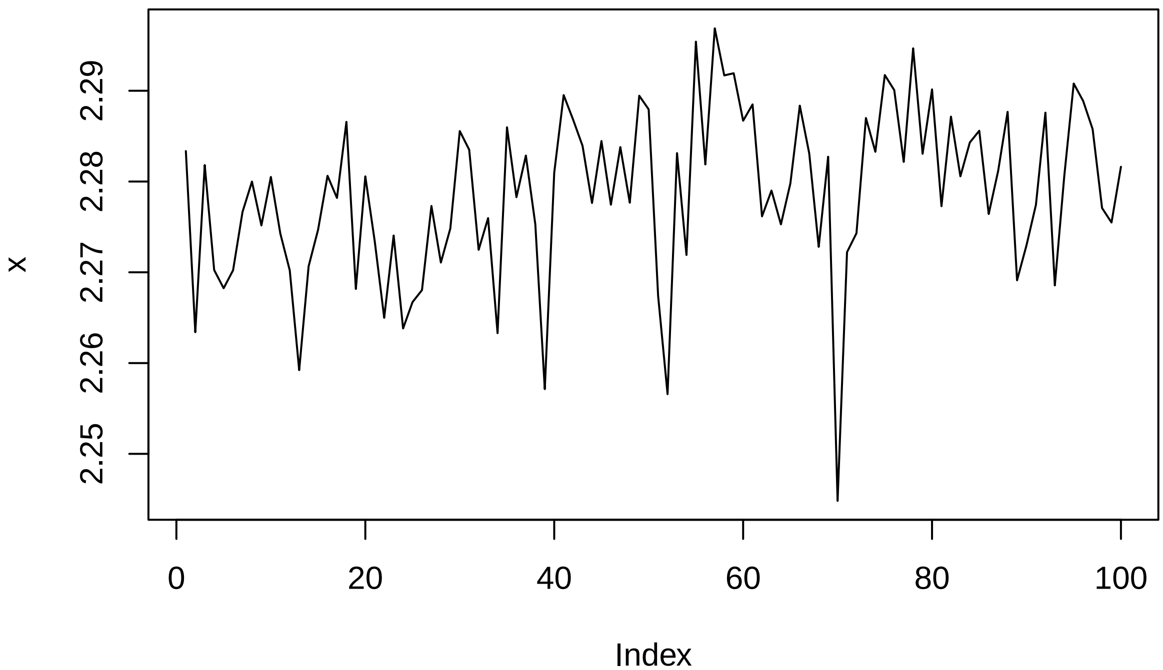


Based on the plot, the entropy of the filtering shows that we are more uncertain about the location of the robot while the smoothing have lower entropy on average. Therefore, by time we do not get more information.

```
filter_path <- matrix(nrow = 200, ncol =100)
smooth_path <- matrix(nrow = 200, ncol =100)
for (i in seq_len(200)){
  simulated_HMM <- simHMM(robot_HMM, 100)
  simulated_HMM <- simulated_HMM$observation

  alpha <- exp(forward(robot_HMM, simulated_HMM))
  beta <- exp(backward(robot_HMM, simulated_HMM))

  smooth_probs <- prop.table(alpha * beta, 2)
  filter_probs <- prop.table(alpha, 2)

  filter_path[i, ] <- apply(filter_probs, 2, which.max)
  smooth_path[i, ] <- apply(smooth_probs, 2, which.max)
}

filter_count <- apply(filter_path, 2, function(x)table(factor(x, levels = 1:10)))
smooth_count <- apply(smooth_path, 2, table)

x <- apply(filter_count, 2, entropy.empirical)
plot(x, type ="l")
x <- apply(smooth_count, 2, entropy.empirical)
plot(x, type ="l")
```



x
1.6
1.8
2.0
2.2
0
20
40
60
80
100
Index



Question 7

$p(z_{t+1}|z_{0:T}, x_{0:T}) = p(z_{t+1}|z_T).p(z_T|x_{0:T})\ where\ z_{t+1}\ is\ independent\ of\ z_{0:t-1}\ given\ z_t p(z_{t+1}|z_T)\ is\ transition\ probability p(z_t$

Based on the formula, we can get the prediction:

```
filtering_dist[, 100] %*% transition_probs
```

```
##      [,1]   [,2] [,3]   [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0 0.1875  0.5 0.3125    0    0    0    0    0     0
```