Assignment 1:
HMM Signal Source

EQ2341 Pattern Recognition
and Machine Learning

Full Name

Haihong Yu, Hailey
Leyh, Levi

Stockholm, April 21, 2023

Contents

1 | Calculation of `P(S_t = j)` for the first infinite-duration HMM and measured relative frequencies

1.1 | Calculation of `P(S_t = j)` for the first infinite-duration HMM

Given that $q = \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}$, $A = \begin{bmatrix} 0.99 & 0.01 \\ 0.03 & 0.97 \end{bmatrix}$, the question asks us to find the probability of state equals to j at time point t, i.e. $P[S_t = j]$, where $j = 1, 2$.

Let's define vector $P_t = \begin{bmatrix} P(S_t = 1) & P(S_t = 2) \end{bmatrix}$.

$$\begin{aligned} P_t * A &= \begin{bmatrix} P(S_t = 1) & P(S_t = 2) \end{bmatrix} * \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \\ &= \begin{bmatrix} P(S_t = 1)a_{11} + P(S_t = 2)a_{21} & P(S_t = 1)a_{12} + P(S_t = 2)a_{22} \end{bmatrix} \\ &= P_{t+1} \end{aligned} \tag{1.1}$$

Therefore, for any time point t,

$$P_t = P_{t-1} * A = P_{t-2} * A^2 = P_{t-3} * A^3 = ... = P_1 * A^{t-1} = q^T * A^{t-1} \tag{1.2}$$

It can be seen that, at any time point t, $P(S_t = j) = P_{t,j}$ is actually a constant.
When $t = 10000$, it can be calculated that

$$P_t = q^T * A^{9999} = \begin{bmatrix} 0.75 & 0.25 \end{bmatrix} * \begin{bmatrix} 0.99 & 0.01 \\ 0.03 & 0.97 \end{bmatrix}^{9999} = \begin{bmatrix} 0.75 & 0.25 \end{bmatrix} \tag{1.3}$$

1.2 | Corresponding measured relative frequencies

To measure the relative frequencies of state1 and state2's presence, we generated 10000 samples using the HMM model with the given parameters.

```
# State generator
mc = MarkovChain( np.array( [ 0.75, 0.25 ] ), np.array( [ [ 0.99, 0.01 ], [ 0.03, 0.97 ]
    ] ) )

b1 = GaussD( means=[0], stdevs=[1] )   # Distribution for state = 1
b2 = GaussD( means=[3], stdevs=[2] )   # Distribution for state = 2
h  = HMM( mc, [b1, b2])                # The HMM

# Generate an output sequence
x,s = h.rand( 10000 )

re_freq_s1 = np.sum(s==0)/(np.sum(s==0) + np.sum(s==1))
re_freq_s2 = np.sum(s==1)/(np.sum(s==0) + np.sum(s==1))
print("relative frequency of state1:", re_freq_s1)
print("relative frequency of state2:", re_freq_s2)
```

The results are as follows

```
relative frequency of state1: 0.7529
relative frequency of state2: 0.2471
```

It can be seen that the measured frequency is quite close to the calculated result.

2 | Theoretically calculated $E[X_t]$ and $var[X_t]$ and measured results

2.1 | Theoretically calculated $E[X_t]$ and $var[X_t]$

The output probability distribution $B = \begin{bmatrix} b_1(x) \\ b_2(x) \end{bmatrix}$, where $b_1(x)$ is a scalar Gaussian density function with mean $\mu_1 = 0$ and standard deviation $\sigma_1 = 1$, and $b_2(x)$ is a similar distribution with mean $\mu_2 = 3$ and standard deviation $\sigma_2 = 2$.
The conditional expectation formulas $\mu_X = E[X] = E_Z[E_X[X|Z]]$ and $var[X] = E_Z[var_X[X|Z]] + var_Z[E_X[X|Z]]$. Therefore, since $P(S_t = j)$is known from last task, we can have

$$\begin{aligned} E[X_t] &= E[X_t|S_t = 1] * P(S_t = 1) + E[X_t|S_t = 2] * P(S_t = 2) \\ &= \mu_1 * P(S_t = 1) + \mu_2 * P(S_t = 2) \\ &= 0 * 0.75 + 3 * 0.25 \\ &= 0.75 \end{aligned} \tag{2.1}$$

$$\begin{aligned} var[X_t] &= \sigma_1^2 * P(S_t = 1) + \sigma_2^2 * P(S_t = 2) \\ &+ P(S_t = 1) * (\mu_1 - E[X_t])^2 + P(S_t = 2) * (\mu_2 - E[X_t])^2 \\ &= 1 * 0.75 + 4 * 0.25 + [0.75(0 - 0.75)^2 + 0.25(3 - 0.75)^2] \\ &= 3.4375 \end{aligned} \tag{2.2}$$

2.2 | Measured results

We generated 10000 samples using the HMM model with the corresponding settings.

```
# State generator
mc = MarkovChain( np.array( [ 0.75, 0.25 ] ), np.array( [ [ 0.99, 0.01 ], [ 0.03, 0.97 ]
    ] ) )

b1 = GaussD( means=[0], stdevs=[1] )   # Distribution for state = 1
b2 = GaussD( means=[3], stdevs=[2] )   # Distribution for state = 2
h  = HMM( mc, [b1, b2])                # The HMM

# Generate an output sequence
x,s = h.rand( 10000 )

mean = np.mean(x)
var = np.var(x)

print("mean:", mean)
print("var:", var)
```

The result is as follows, which is quite close the the calculation.

```
mean: 0.7695832648955082
var: 3.5412433680568602
```

3 | Plot of 500 randomly generated samples and output behavior

3.1 | Plot of 500 results

Following parameter settings of $q = \begin{bmatrix} 0.75 \\ 0.25 \end{bmatrix}, A = \begin{bmatrix} 0.99 & 0.01 \\ 0.03 & 0.97 \end{bmatrix}, B = \begin{bmatrix} b_1(x) \\ b_2(x) \end{bmatrix}$, where $b_1(x)$ is a scalar Gaussian density function with mean $\mu_1 = 0$ and standard deviation $\sigma_1 = 1$, and $b_2(x)$ is a similar distribution with mean $\mu_2 = 3$ and standard deviation $\sigma_2 = 2$, we generated 500 samples output and plot them as a function of t.

From the plot, although the sample values are randomized, there seem to be 2 centers that the values are distributed around, one is about 3, and the other is about 0, which are exactly the means of the 2 output distributions.

Furthermore, the number of samples that is centered around $\mu_2 = 3$ seems smaller than the ones centered around $\mu_1 = 0$, this might be due to $P(S_t = j)$, where $P(S_t = 2) = 0.25 < P(S_t = 1) = 0.75$.

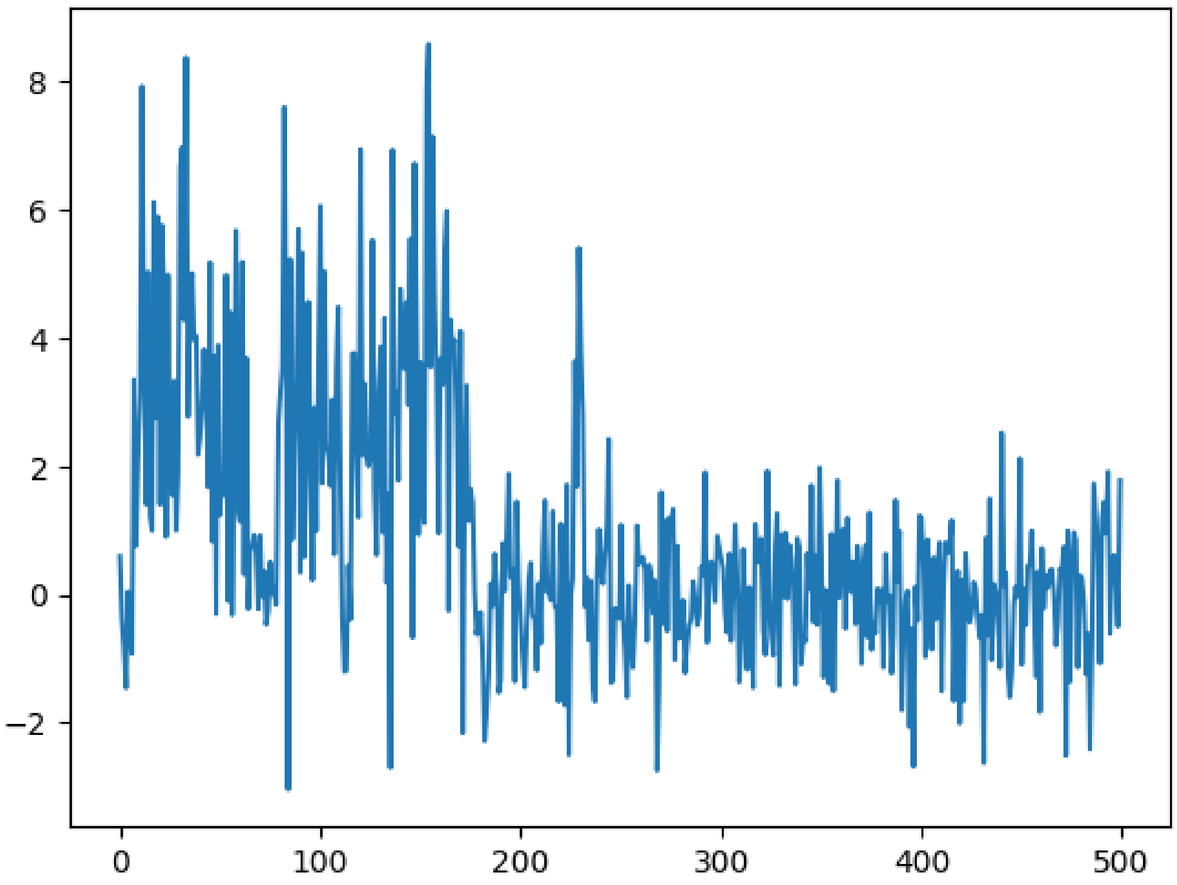


Figure 3.1: Plotted output: samples X_t as a function of t with $\mu_1 = 0, \mu_2 = 3$

Below is the code used:

```
# State generator
mc = MarkovChain( np.array( [ 0.75, 0.25 ] ), np.array( [ [ 0.99, 0.01 ], [ 0.03, 0.97 ]
    ] ) )

b1 = GaussD( means=[0], stdevs=[1] )   # Distribution for state = 1
b2 = GaussD( means=[3], stdevs=[2] )   # Distribution for state = 2
h  = HMM( mc, [b1, b2])                # The HMM

# Generate an output sequence
x,s = h.rand( 500 )
plt.plot(x[0])
```

4 | Output behaviour of the second infinite-duration HMM

associated Question:
Create a new HMM, identical to the previous one except that it has $\mu_2 = \mu_1 = 0$. Generate and plot 500 contiguous values several times using @HMM/rand for this HMM. What is similar about how the two HMMs behave? What is different with this new HMM? Is it possible to estimate the state sequence S of the underlying Markov chain from the observed output variables x in this case?

4.1 | Generation and plotting of the HMM using $\mu_2 = \mu_1 = 0$

Used code:

```
# Define the state generator
mc = MarkovChain( np.array( [ 0.75, 0.25 ] ), np.array( [ [ 0.99, 0.01 ], [ 0.03, 0.97 ]
    ] ) )

# Define the distributions for each state and the HMM
b1 = GaussD( means=[0], stdevs=[1] )    # Distribution for state = 1
b2 = GaussD( means=[0], stdevs=[2] )    # Distribution for state = 2
h  = HMM( mc, [b1, b2])                 # The HMM

# Generate an output sequence
x,s = h.rand( 500 )

# Plot the output sequence
plt.plot(x[0])
```

Plots obtained:

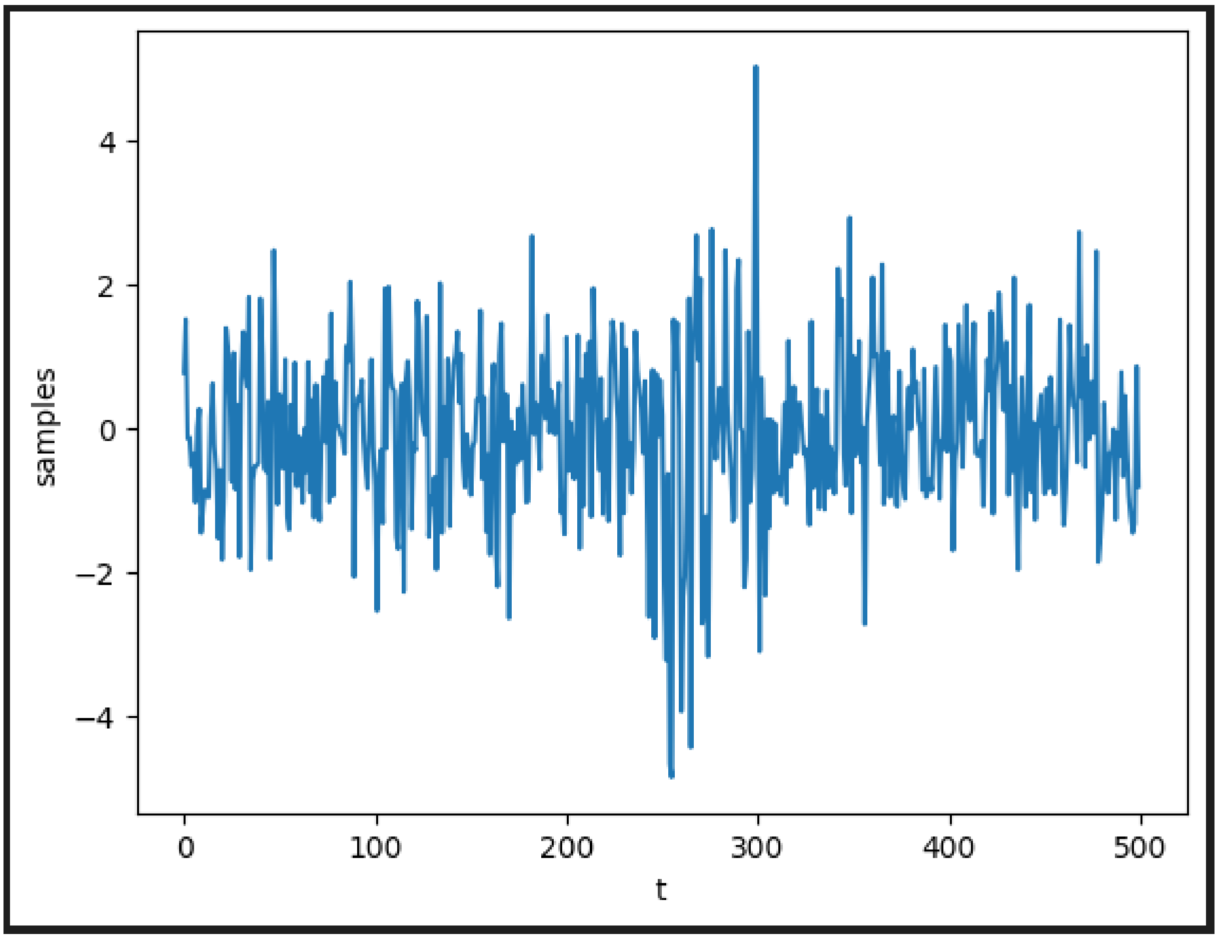


Figure 4.1: Plotted output: samples X_t as a function of t with $mu_2 = \mu_1 = 0$

4.2 | Discussion

One similarity is that the transition probabilities between the states in the transition matrix are the same for both HMMs.
The difference between the two HMMs is that in the previous HMM, the mean of the emission distribution for the states was 0 and 3, while in the new HMM, both means are 0. This means that the generated sequences from the new HMM has a mean of 0 for both states, while in the previous HMM, the mean values were different between the two states. This can be observed in the plotted output, as X_t moves around 0 for all t.

In this case, it is difficult to estimate the state sequence S of the underlying Markov chain from the observed output variables x using the Viterbi algorithm, since the emission distributions for both states have the same mean value. It is still possible, but it my be less accurate since both emission distributions are zero. Based on the observed values alone, the only way to distinguish between the two states is via the different variance.

5 | Finite-duration test HMM

associated Question:
Another aspect you must check is that your rand-function works for finite-duration HMMs. Define a new test HMM of your own and verify that your function returns reasonable results.

5.1 | Generation and plotting of a finite-duration HMM

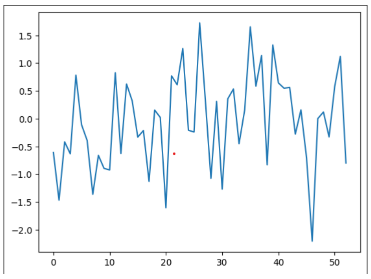
Used code:

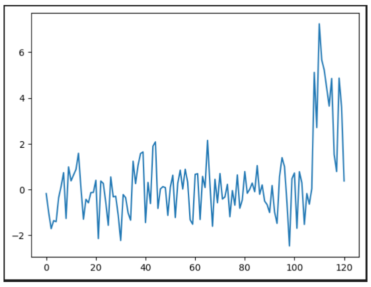
```
# Define the state generator
mc = MarkovChain( np.array( [ 0.75, 0.25 ] ), np.array( [ [ 0.98, 0.01, 0.01], [ 0.02,
    0.97, 0.01]] ) )

# Define the distributions for each state and the HMM
b1 = GaussD( means=[0], stdevs=[1] )    # Distribution for state = 1
b2 = GaussD( means=[3], stdevs=[2] )    # Distribution for state = 2
b3 = GaussD( means=[1], stdevs=[2] )    # Distribution for end state
h  = HMM( mc, [b1, b2, b3])                     # The HMM

# Generate an output sequence
x,s = h.rand( 500 )
plt.plot(x[0])
```

Plots obtained:


(a) First plot


(b) Second plot

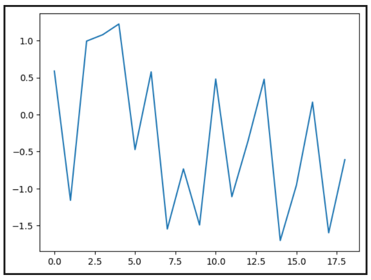

(c) Third plot

Figure 5.1: Plotted output: samples X_t as a function of t for finite-duration using a unsquare matrix

5.2 | Discussion

For finite-duration Markov Chains, there is a final state that cannot be exited, and the state transitions are restricted by the transition matrix.
Therefore, when generating a random sequence of data from a finite-duration Markov Chain, it is reasonable that the sequence may end earlier than the specified nSamples parameter because the Markov Chain may reach its final state before nSamples iterations are completed.
This is the difference to a infinite-duration Markov Chain, which can continue to transition between states indefinitely.
The number of samples, after which the HMM ends is random and depends on the random data sequence generated by the rand function.

6 | Testing the rand function with randomly generated vector outputs

6.1 | Definition of the test with Gaussian vector distributions outputs

Used code:

```
# State generator
mc = MarkovChain( np.array( [ 0.75, 0.25 ] ), np.array( [ [ 0.99, 0.01], [ 0.03, 0.97]] )
     )

b1 = GaussD( means=[0,0], stdevs=[1,1], cov=np.array([[2,1],[1,4]]) )   # Distribution
    for state = 1
b2 = GaussD( means=[3,3], stdevs=[2,2], cov=np.array([[2,1],[1,4]]) )   # Distribution
    for state = 2
h  = HMM( mc, [b1, b2])                  # The HMM

# Generate an output sequence
x,s = h.rand( 500 )

print(x)
print(len(x))
print(len(x[0]))

plt.plot(x[0])
plt.xlabel('t')
plt.ylabel('samples')
```

Plots obtained:

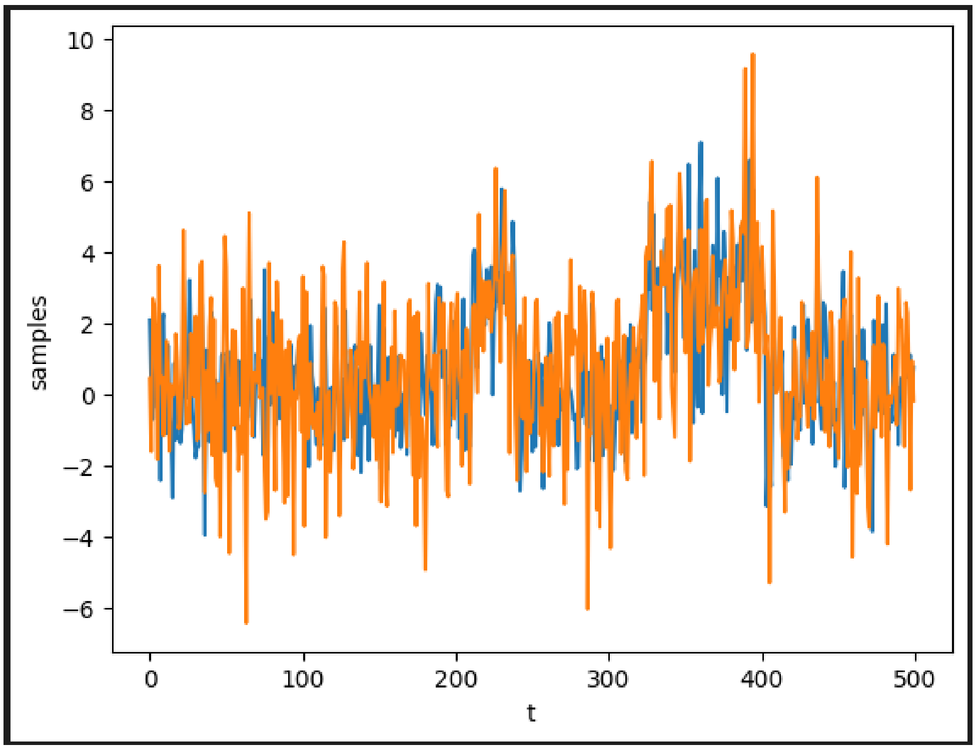


Figure 6.1: Plotted output: samples X_t as a function of t using randomly generated vector outputs

The code was changed in such a way that now additionally a non-diagonal covariance matrix were given and another dimension was added to the *mu* parameters. As can be observed by the plot, our implementation also works with vector output distributions and valid values are returned.

A | Appendix

A.1 | DiscreteD rand function

```
def rand(self, nData):
    """
    R=rand(nData) returns random scalars drawn from given Discrete Distribution.

    Input:
    nData= scalar defining number of wanted random data elements

    Result:
    R= row vector with integer random data drawn from the DiscreteD object
       (size(R)= [1, nData]
    """
    R = np.random.choice(a=range(len(self.probMass)), size=nData, p=self.probMass)

    return R
```

A.2 | MarkovChain rand function

```
def rand(self, tmax):
    """
    S=rand(self, tmax) returns a random state sequence from given MarkovChain object.

    Input:
    tmax= scalar defining maximum length of desired state sequence.
       An infinite-duration MarkovChain always generates sequence of length=tmax
       A finite-duration MarkovChain may return shorter sequence,
       if END state was reached before tmax samples.

    Result:
    S= integer row vector with random state sequence,
       NOT INCLUDING the END state,
       even if encountered within tmax samples
    If mc has INFINITE duration,
       length(S) == tmax
    If mc has FINITE duration,
       length(S) <= tmaxs
    """
    # Initialize variables
    S = np.array([])
    i = DiscreteD(self.q).rand(1)
    duration = 0

    # Generate state sequence
    while duration < tmax:
        S = np.concatenate((S,i))
        duration += 1
        if i == self.nStates and self.is_finite :    # END state
            break

        p = self.A[i, :][0]
        j = DiscreteD(p).rand(1)
        i = j

    return S.astype(int)
```

A.3 | HMM rand function

```
    def rand(self, nSamples):
        """
        [X,S]=rand(self,nSamples); generates a random sequence of data
        from a given Hidden Markov Model.

        Input:
        nSamples=  maximum no of output samples (scalars or column vectors)

        Result:
        X= matrix or row vector with output data samples
        S= row vector with corresponding integer state values
          obtained from the self.StateGen component.
          nS= length(S) == size(X,2)= number of output samples.
          If the StateGen can generate infinite-duration sequences,
              nS == nSamples
          If the StateGen is a finite-duration MarkovChain,
              nS <= nSamples
        """

        #*** Insert your own code here and remove the following error message
        S = self.stateGen.rand(nSamples)
        #print("S from HMM:",S)
        #print("outputdis", self.outputDistr)
        X = []
        for state in S:
            i = self.outputDistr[state].rand(1)
            #print("X", X)
            #print("i", i)
            if len(i)==1:
                X = np.concatenate((X,i[0]))
            else:
                X.append([i[0][0], i[1][0]])

        return [np.array([X]),S]
```