### A.1.1 HMM Random Source

Your task is to code and verify MatLab methods to generate an output sequence of random real numbers $\underline{x} = (x_1 \dots x_t \dots x_T)$ from an HMM with scalar Gaussian output distributions. However, your code should be general enough to handle vector random variables as well.

An HMM output sequence is always the result of *two* separate random operations: First the hidden Markov chain must generate an integer state sequence $\underline{s} = (s_1 \dots s_T)$, and then, for each element $s_t$ in the state sequence, the corresponding state-conditional output distribution generates the random observable output $X_t$. In the `PattRecClasses` code package three different functions are involved in the process: `@HMM/rand`, `@MarkovChain/rand`, and for example `@GaussD/rand`, if the output distribution is an instance of the `GaussD` class.

1. Open `@DiscreteD/rand` and finish the code precisely as specified by the predefined function interface. Save your function with the same name in the same directory. You may need to use the help system, or look into the `@DiscreteD/DiscreteD` class definition to see exactly how the class properties are stored internally.

2. Open `@MarkovChain/rand` and finish the code precisely as specified by the predefined function interface. Save your function with the same name in the same directory. You may need to use the help system, or look into the `@MarkovChain/MarkovChain` class definition to see exactly how the class properties are stored internally.

   Since the initial state of a Markov chain, and its transitions conditioned on the current state, can be seen as discrete random variables, you can use the `DiscreteD` class and the `rand` method you implemented in the previous step to simplify your work here.

   Note that your function must be able to generate output sequences for either an *infinite-duration* or a *finite-duration* Markov chain. Of course, your function should only produce sequences of finite length, even if the HMM itself could in principle continue forever.

3. Open `@HMM/rand` and finish the code precisely as specified by the predefined function interface. Save your function with the same name in the same directory. Once again you may need to look into the `@HMM/HMM` class definition to see how the class is implemented internally.

   To complete this method you need to call the `rand` method you just implemented for the state-generator component (of class `MarkovChain`) of the HMM. You also need to call another `rand` method for the `OutputDistr` array of the HMM. Just make sure to call this method for the correct `OutputDistr` element, depending on the value of the

corresponding element in the state sequence. (However, the HMM should not generate any output for the final "state" that just indicates that the end of a finite-duration state sequence was reached.)

It is very important that your implementations always follow the interface specifications, both with regards functionality and input/output conventions.

When the `OutputDistr` is a `GaussD` array, the special `@GaussD/rand` method will be invoked automatically by MatLab. This method is very easy to implement for a scalar Gaussian distribution, using the standard MatLab function `randn`. However, it is a little more complicated for a Gaussian vector distribution. Therefore, the `@GaussD/rand` method has already been implemented for you.

### A.1.2 Verify the MarkovChain and HMM Sources

To verify your code, use the following infinite-duration HMM $\lambda = \{q, A, B\}$ as a first test example:

$$q = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}; \quad A = \begin{pmatrix} 0.99 & 0.01 \\ 0.03 & 0.97 \end{pmatrix}; \quad B = \begin{pmatrix} b_1(x) \\ b_2(x) \end{pmatrix}$$

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 $\sigma_2 = 2$.

1. To verify your Markov chain code, calculate $P(S_t = j)$, $j \in \{1, 2\}$ for $t = 1, 2, 3 \ldots$ theoretically, by hand, to verify that $P(S_t = j)$ is actually constant for all $t$.

2. Use your Markov chain `rand` function to generate a sequence of $T = 10\,000$ state integer numbers from the test Markov chain. Calculate the relative frequency of occurrences of $S_t = 1$ and $S_t = 2$. The relative frequencies should of course be approximately equal to $P(S_t)$.

3. To verify your HMM `rand` method, first calculate $E[X_t]$ and $\mathrm{var}[X_t]$ theoretically. The conditional expectation formulas $\mu_X = E[X] = E_Z[E_X[X|Z]]$ and $\mathrm{var}[X] = E_Z[\mathrm{var}_X[X|Z]] + \mathrm{var}_Z[E_X[X|Z]]$ apply generally whenever some variable $X$ depends on another variable $Z$ and may be useful for the calculations. Then use your HMM `rand` function to generate a sequence of $T = 10\,000$ output scalar random numbers $\underline{x} = (x_1 \ldots x_t \ldots x_T)$ from the given HMM test example. Use the standard MatLab functions `mean` and `var` to calculate the mean and variance of your generated sequence. The result should agree approximately with your theoretical values.

4. To get an impression of how the HMM behaves, use `@HMM/rand` to generate a series of 500 contiguous samples $X_t$ from the HMM, and plot them as a function of $t$. Do this many times until you have a good idea of what characterizes typical output of this HMM, and what structure there is to the randomness. Describe the behaviour in one or two sentences in your report. Also include one such plot in the report, labelled using `title`, `xlabel`, and `ylabel` to clearly show which variable is plotted along which axis. You should do this for every plot in the course project.

5. 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 $\underline{S}$ of the underlying Markov chain from the observed output variables $\underline{x}$ in this case?

6. 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.

7. Finally, your `rand`-function should work also when the state-conditional output distributions generate random vectors. Define a new test HMM of your own where the outputs are Gaussian vector distributions, and verify that this also works with your code. (Note that a single instance of the `GaussD` class is capable of generating vector output; stacking several `GaussD`-objects is not correct.) At least one of the output distributions should have a non-diagonal covariance matrix such as

$$\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix}.$$

### Your assignment report should include

- Copies of your `@DiscreteD/rand`, `@MarkovChain/rand`, and `@HMM/rand` functions, either attached as separate m-files, or in a zip archive. Merely pasting the code into a pdf or doc file is not sufficient. Also, please avoid the proprietary rar format.

- Your theoretically calculated $P(S_t = j)$ for the first infinite-duration HMM, and your corresponding measured relative frequencies.

- Your theoretically calculated $E[X_t]$ and $\text{var}[X_t]$, and your corresponding measured results.

- A plot of 500 contiguous values randomized from the first infinite-duration HMM, with a description of typical output behaviour.

- A discussion of the output behaviour of the second infinite-duration HMM, with answers to the associated questions.
- The definition of your *finite-duration* test HMM, together with the lengths of some test sequences you obtained, and relevant code. Discuss briefly why you think those lengths are reasonable.
- The definition of your vector-valued test HMM, and the code you used to verify that vector output distributions work with your implementation.