1. **Second passage time.** In this problem we will consider the following Markov chain.

Note that self-loops are omitted from this figure. The transition matrix for this chain is

\[
P = \begin{bmatrix}
0.4 & 0.3 & 0 & 0.3 & 0 & 0 \\
0 & 0.4 & 0 & 0.3 & 0 & 0 \\
0.3 & 0.1 & 0 & 0.3 & 0 & 0.3 \\
0.3 & 0 & 0.4 & 0.3 & 0 & 0 \\
0 & 0.3 & 0.3 & 0.1 & 0 & 0 \\
0 & 0 & 0.3 & 0.3 & 0.4 & 0 \\
\end{bmatrix}.
\]

This matrix is given in `passage_time_data.m` so you don’t have to type it into MATLAB. We will compute several things about this Markov chain. Let the destination set be the single state \( E = \{1\} \), and let the initial state be \( i = 2 \). Recall that the first passage time to the set \( E \) is

\[
\tau_E = \min\{t > 0 \mid x_t \in E\}
\]

The \( k \)th passage time for a Markov chain is defined recursively as

\[
\tau^{(k)}_E = \min\{t > \tau^{(k-1)}_E \mid x_t \in E\},
\]

where \( \tau^{(1)}_E = \tau_E \).

(a) For the Markov chain above, compute and plot the distribution of the first passage time conditioned on the initial state, i.e., compute

\[
f(t) = \text{Prob}(\tau_E = t \mid x_0 = i)
\]

as a function of \( t \).
(b) One may calculate the distribution of the second passage time by constructing a new Markov chain, whose first passage time is equal to the second passage time of the original chain. What is the transition graph of this chain?

(c) For \( t = 1, 2, \ldots \), use your construction to compute the distribution of the second passage time, which is

\[
s(t) = \text{Prob}(\tau_E^{(2)} = t \mid x_0 = i).
\]

Use your method to compute and plot this distribution also.

(d) Plot the sum of \( s(t) \) and \( f(t) \), and also plot \( \text{Prob}(x_t \in E) \). Explain what you see.

(e) For a given state \( j \), the recurrence time is the first passage time to the state \( j \) given the initial condition \( x_0 = j \). This has distribution

\[
r(t) = \text{Prob}(\tau_{\{j\}} = t \mid x_0 = j).
\]

Let \( j = 1 \). Explain how to compute this, compute it, and plot it.

(f) Show that \( s = f * r \), where * denotes convolution. Verify this numerically, and interpret this result.

(g) Give a method to compute the distribution of the \( k \)th passage time, for any \( k \). How does your method scale with \( k \)?

2. **Class decomposition of a Markov chain.** In lecture we claimed that the states of a Markov chain can be ordered so that the probability-transition matrix has the form

\[
P = \begin{bmatrix} P_{11} & P_{12} \\ 0 & P_{22} \end{bmatrix},
\]

where \( P_{11} \) is block upper triangular with irreducible blocks on the diagonal, and \( P_{22} \) is block diagonal with irreducible blocks. Each of the blocks on the diagonal of \( P_{11} \) represents a transient class, while each of the blocks on the diagonal of \( P_{22} \) represents a recurrent class. In this problem you will write code to find an ordering of the states that puts \( P \) in this form. We will use standard graph-theory terminology throughout the problem. Feel free to consult Wikipedia or other external sources if you encounter any unfamiliar concepts. The file \texttt{class_decomposition_data.m} defines a specific probability-transition matrix that you should use throughout this problem. However, your code must work for any probability-transition matrix.

(a) Let \( G \) be a graph with adjacency matrix \( A \in \mathbb{R}^{n \times n} \). We write \( i \rightarrow j \) if there is a path from node \( i \) to node \( j \) in \( G \). We define the reachability matrix \( R \in \mathbb{R}^{n \times n} \) such that

\[
R_{ij} = \begin{cases} 
1 & i \rightarrow j, \\
0 & \text{otherwise}.
\end{cases}
\]
An algorithm for constructing $R$ is given in algorithm 1.

$$R = A \quad /* \text{A is the adjacency matrix} */$$

repeat
   for $i = 1$ to $n$ do
       /* iterate over all states $k$ we know are reachable from $i$ */
       for $k$ such that $R_{ik} = 1$ do
           /* iterate over all states $j$ we know are reachable from $k$ */
           for $j$ such that $R_{kj} = 1$ do
               /* $j$ is reachable from $i$ through $k$ */
               $R_{ij} = 1$
           end
       end
   end
until no changes are made to $R_{ij}$

Algorithm 1: Computing the reachability matrix

Write a function that implements algorithm 1; use the following function header.

function $R = \text{reachable\_states}(A)$

For reference, our implementation of $\text{reachable\_states}$ is less than twenty lines. Hint. The function $\text{find}$ if you are using MATLAB may be useful.

(b) We write $i \leftrightarrow j$ if $i \rightarrow j$ and $j \rightarrow i$, and we define the communication matrix $C \in \mathbb{R}^{n \times n}$ such that

$$C_{ij} = \begin{cases} 
1 & i = j \text{ or } i \leftrightarrow j, \\
0 & \text{otherwise}.
\end{cases}$$

Explain how to construct the communication matrix from the reachability matrix.

(c) We define the transience vector $t \in \mathbb{R}^n$ such that

$$t_i = \begin{cases} 
1 & i \text{ is a transient state}, \\
0 & \text{otherwise}.
\end{cases}$$

Explain how to construct the transience vector from the communication and reachability matrices.

(d) Write a function that implements algorithm 2.
$L \leftarrow \text{Empty List} / \! / L \text{ will contain sorted nodes} \*/$

$S \leftarrow \text{Set of all nodes with no incoming edges}$

\textbf{while } S \text{ is non empty } \textbf{do}

- \text{Remove a node } n \text{ from } S$
- \text{Add } n \text{ to tail of } L$

\textbf{for each node } m \text{ with edge } e \text{ from } n \text{ to } m \textbf{ do}

- \text{Remove edge } e \text{ from graph}$

\textbf{if } m \text{ has no other incoming edges } \textbf{then}

- \text{Insert } m \text{ into } S$

\textbf{end}$

\textbf{end}$

\textbf{end}$

\textbf{if } \text{Graph has edges } \textbf{then}$

- \textbf{return error: graph has at least one cycle}$

\textbf{else}$

- \textbf{return } L / \! / L \text{ a topologically sorted order} \*/$

\textbf{end}$

\textbf{Algorithm 2: Computing the reachability matrix}$

Use the following function header.

\textbf{function } L = \text{topological\_sort}(A)$

For reference, our implementation of \texttt{topological\_sort} is less than twenty lines.

\textbf{(e)} Suppose that duplicate rows of } C \text{ have been removed, and the rows have been sorted so that the first } n_t \text{ rows represent the transient classes. (You need to write code to do this; the commands } \texttt{unique} \text{ and } \texttt{sort} \text{ may be useful.) Note that there is now a unique row of } C \text{ corresponding to each class. For two classes } C_i, C_j \subseteq \{1,\ldots,n\}, \text{ we write } C_i \rightarrow C_j \text{ if } x \rightarrow y \text{ for some } x \in C_i \text{ and } y \in C_j. \text{ The adjacency matrix } A_t \in \mathbb{R}^{n_t \times n_t} \text{ for the set of transient classes is defined such that}$

\[
(A_t)_{ij} = \begin{cases} 
1 & C_i \rightarrow C_j, \\
0 & \text{otherwise}. 
\end{cases}
\]

\text{Explain how to use the reachability matrix } R \text{ of the Markov chain to find the adjacency matrix } A_t \text{ for the set of transient classes.}$

\textbf{(f)} You should now have all the tools you need to construct an ordering of the states that puts the probability transition matrix in the desired form. Apply your method to the matrix in \texttt{class\_decomposition.data.m}. Attach a plot of calling \texttt{spy} on your reordered matrix (the data file generates a plot of the original matrix). For reference, our solution has thirty-five lines (not including \texttt{reachable\_states} and \texttt{topological\_sort}).
3. **Markov web surfing model.** A set of \( n \) web pages labeled 1, \ldots, \( n \) contain (directed) links to other pages. We define the link matrix \( L \in \mathbb{R}^{n \times n} \) as

\[
L_{ij} = \begin{cases} 
1 & \text{if page } i \text{ links to page } j \\
0 & \text{otherwise.}
\end{cases}
\]

We define \( o \in \mathbb{R}^n \) as \( o = L1 \), which gives the number of outgoing links from each page.

A very crude model of a web surfer is a Markov chain on the pages, with transitions described as follows. The model includes a parameter \( \theta \in (0, 1) \), which (roughly) gives the probability that the surfer follows a link from the current page. For a page with \( o_i > 0 \) (i.e., with at least one outgoing link) the surfer moves to each of the linked-to pages with probability \( \theta/o_i \), and jumps to a page not linked to \( i \) with probability \( (1 - \theta)/(n - o_i - 1) \). For a page with no outgoing links (i.e., \( o_i = 0 \)) the surfer jumps to a random page, chosen from a uniform distribution on (the other) pages.

We will assume that web surfer starts at a random page, uniformly distributed.

We earn a payment when the surfer follows (i.e., clicks on) a link, given by \( R_{ij} \geq 0 \). This payment matrix satisfies \( R_{ij} = 0 \) when \( L_{ij} = 0 \) (i.e., we are not paid for random jumps; only following links).

The following questions concern the specific instance of the problem with data given in link_matrix_data.m.

(a) What is the most likely page the surfer is on, at time \( t = 10 \)? at \( t = 100 \)?

(b) Let \( J \) denote the expected total payment over \( t = 0, \ldots, 50 \). Compute \( J \) three ways:

- Monte Carlo simulation (which gives an estimate of \( J \), not the exact value).
- Distribution propagation.
- Value iteration.

Be sure to check that the values are consistent.

**Remark.** The Markov model described in this problem leads to Google’s famous PageRank, which corresponds to the fraction of time spent at each site, when \( T \to \infty \). (The current version of PageRank is based on far more than just the link topology, but the first versions really did make heavy use of the Markov surfing model.)