Analysis of a discrete time Markov chain

1. Consider the markov chain illustrated by the transition graph in Fig. 1. Construct the transition matrix \( T \) in Matlab; don’t forget that \( T_{ij} \) is the probability of ending a step in state \( i \), given that the system starts the step in state \( j \).

2. Find the probability distribution after 10 steps given an initial starting position of state 2.

3. Find the stationary distribution reached in the long time limit by finding the eigenvector with eigenvalue 1 (the largest eigenvalue). Make sure it is normalised.

4. Find the expected (average) time to reach state 8, given that the system starts in state 1, by using the fundamental matrix method.

5. (Optional) Find the probability that a system starting in state 4 reaches state 2 before it reaches state 1. Again, use the fundamental matrix method.
Simulating an overdamped particle

In this exercise, we will consider an overdamped particle moving in the potential \( V(x) = 2(x + 1)^2(x - 1)^2 \), in a medium with viscosity \( \gamma = 10 \) and temperature \( k_B T = 1 \) (don’t worry about the lack of units; it simply means we’re defining all dimensional quantities relative to each other). We will generate long sample trajectories by numerically integrating the Langevin equation; if we run these simulations for long enough, the sample trajectories will sample states in a manner consistent with the stationary distribution. We can then use these samples to understand properties of the stationary distribution; this is an extremely common approach!

1. Plot the potential \( V(x) \) for the range \(-2 \leq x \leq 2\). How do you expect the particle to behave?

2. Use Euler’s method to integrate the stochastic differential equation

\[
\frac{dx}{dt} = -\frac{1}{\gamma} \frac{dV(x)}{dx} + \eta(t)
\]

for a total time of 1000, using a time step of 0.01, starting from a position of \( x = 0 \). Plot \( x \) as a function of time. You will need to use the fact that

\[
\int_{t_0}^{t_0 + \Delta t} \eta(t) dt
\]

is a Gaussian random number with mean 0 and standard deviation \( \sqrt{(2k_B T \Delta t)/\gamma} \).

3. By running long simulations (for a total time of 10000) and measuring the energy at each step, estimate the average energy of the particle in the stationary distribution. What happens if you change your time step to 0.2? The exact answer is 0.545728.

4. Construct a histogram to record the states visited in the stationary distribution as sampled from a simulation of total time 3000. How does this histogram change if you increase the number of time steps to 30000? Which is more trustworthy?

5. What is the Boltzmann distribution for \( x \) in this system? Plot this function. Are your histograms approximately consistent with this prediction?

6. (Optional) Matlab cannot actually generate truly random numbers; instead, it uses a complex formula to generate a sequence of numbers that look random (i.e., it is difficult to predict the next number knowing the current one). However, the sequence is actually fully determined by a single integer “seed”. If we set this seed to be the same, we’ll get the same sequence of random numbers. This seed is automatically set to 0 every time you restart Matlab, so you’ll always get the same “random” trajectory if you simply open Matlab and press go. Explore what happens as you change the seed using the \texttt{rng(seed)} instruction.

In this coursework you may need to use the following Matlab commands: \texttt{normrnd} and \texttt{hist} or \texttt{histogram} (depending on version). You will also find it useful to define functions analogously to \texttt{f= @(x) x^2}; \texttt{f(x)} will then return \( x^2 \). You can check the Matlab help by using \texttt{help COMMAND}. 
Coupled population processes

In this exercise, we will consider the coupled population dynamics of a simple “predator-prey” model. You will learn how to code-up your own (2d) Gillespie simulation. The model is a stochastic version of the famous Lotka-Volterra equations, which describe the interaction between a population of predators and a population of prey. In the deterministic case, the system shows limit cycle oscillations in populations: https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations.

We have two populations, rabbits ($R$) and foxes ($F$). Rabbits are born at a rate $aR$ ($a$ per rabbit), and consumed by foxes at a rate $bRF$. Foxes are born at a rate depending on the number of rabbits available, $gRF$, and die at a rate $cF$ ($c$ per fox).

1. Construct an m-file function that takes the current population of $R$ and $F$, along with the birth/death parameters, as input, and returns a new state and time for transition sampled appropriately. You will need to consider the relative rates of four transitions at each step, and pick one using a random number. You will then need to draw a time from an exponential distribution. It might be useful to build an “exception handler” that deals with the possibility that $R, F = 0$ is the current (absorbing) state.

2. Use this function to explore the possible behaviour of the ecosystem. Taking $a = 1$, $b = 0.01$, $c = 0.4$ and $g = 0.002$, and initial conditions of $R = 100$ and $F = 10$, simulate for 100000 transitions or until both populations become extinct, whichever occurs first. Plot the populations as a function of time (note: not number of transitions). Run the code several times; what types of behaviour do you observe? What has happened to the oscillations?

3. By running 100 simulations, estimate the average time (note: not number of transitions) until the foxes become extinct.

In this coursework you may need to use the following Matlab commands: 
**rnd** and **exprnd**. You will also find it helpful to define a separate “.m” file defining a function that actually calculates the next step. The first line of the file (in my case named gillespie_ex.m) should look something like: **function**

```
[ Rnew,Fnew,deltat ] = gillespie_ex(Rold,Fold,a,b,c,g).
```

The quantities Rnew, Fnew and deltat are then calculated in the file, and automatically returned. You can then call this function from your main code each time you need to take a step. You can check the Matlab help by using **help** **COMMAND**.
Disease propagation on networks

In this exercise, we will construct some simple networks and analyse the propagation of diseases on those networks. We will use the file ‘SocialNetwork.mat’, available from blackboard, to define a realistic example.

1. Construct an adjacency matrix for an undirected Erdős-Rényi network. The network should have 276 nodes, and each possible link should exist with a probability 0.054 (don’t add self-links). Use the `spy` command to visualise the adjacency matrix, and calculate the total number of connections.

2. The command `load('SocialNetwork.mat');` will create a matrix `A` of 276 nodes, corresponding to a social network. Visualize it using `spy`, calculate the total number of connections and comment on the difference from your random graph.

3. A disease spreads through a population by infecting susceptible population members that are adjacent to an infected member. Each day, susceptible members that are adjacent to a single infected member become infected with a probability \( p = 0.2 \) (if a susceptible member is adjacent to two infected members, each independently infects it with probability 0.2). Infected population members are infectious for one day, then die (if this is to gruesome, you could call them “recovered and immune”). Define a vector “susceptible” of length 276; susceptible(n) is 1 if the nth population member is currently susceptible, and 0 otherwise. Initialise all entries to 1. Similarly, define “infected” and “dead” vectors initialised to 0.

   • Randomly infect one individual “i” (set susceptible(i)=0 and infected(i)=1). Evolve the system for 100 days, infecting susceptible neighbours of infected nodes with a probability of 0.2, using your Erdős-Rényi adjacency matrix. Plot the total number of victims (as a fraction of the total population) over this period. Repeat 100 times and plot on the same graph. What tends to happen?
   • Repeat for the social network represented by adjacency matrix `A`. Do these networks behave similarly? Is there any evidence of a difference?

4. The Government implements a vaccination programme, which reaches 75% of people. Restart the simulation for matrix `A`, but this time set susceptible(i)=0 with a probability of 75% for each individual i. Pick an initially susceptible victim, infect them, and simulate for 100 days. Repeat for 100 runs and plot the results. What fraction of the initially susceptible people are now typically infected? In what way could this be described as “herd immunity”?

5. Certain communities in the social network described by `A` oppose the vaccination. Rather than setting people to be immune randomly, set the final 75% of individuals in the list to be immune (70 to 276). You can confirm, using `spy`, that the first 69 individuals are indeed well connected within their community, but poorly connected outside. Repeat the above test; how are the results different? Why?
In this coursework you may need to use the following Matlab commands: `rand` and `randi`. You will also find it helpful to define a separate “.m” file defining a function that stochastically generates next step of the disease propagation. The first line of the file (in my case named `infect_new.m`) should look something like: `function [ susceptible_new,infected_new,dead_new ] = infect_new(adjacency,susceptible,infected,dead, ptrans).` In other words, you should feed in an adjacency matrix, and vectors listing the current set of susceptible, infected and dead population members, and the infection probability. The function should stochastically infect those that are connected to infected population members, and then update previously infected members to the dead list. You can check the Matlab help by using `help COMMAND`. 