CSCI 2244 – Homework 7

Out: Friday, October 25, 2019
Due: Friday, November 8, 2019, 11:59pm

This homework consists of written exercises **and** coding problems. You *must* type your solutions. See the "Assignments" section in the syllabus for advice about doing this. You should submit your homework via Canvas. In particular, you should upload a zip file called:

`FirstName_LastName_Homework7.zip`

Please use your full first name and last name, as they appear in official university records. The reason for doing so is that the TAs and I must match up these names with the entries in the gradebook.

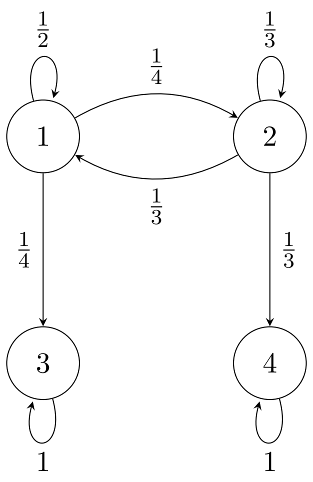
This zip file should contain 2 files:

- `written.pdf` – containing your answers to all the tasks in section 1, and the results of running your code as requested by the task in section 2.
- `shuffle.py` – containing the code you wrote for section 2.

1 Written Exercises

The problems below involve analyzing various finite Markov chains. As we saw in class, this amounts to analyzing the transition matrix and computing things like inverses and eigenvectors. See the **appendix** for an explanation of how to compute these things in Python.

Task 1.1 (12 pts). Consider the following Markov chain:



$$\begin{tikzcd} 1 \arrow[loop above, "\frac{1}{2}"] \arrow[r, bend left, "\frac{1}{4}"] \arrow[d, "\frac{1}{4}"'] & 2 \arrow[loop above, "\frac{1}{3}"] \arrow[l, bend left, "\frac{1}{3}"] \arrow[d, "\frac{1}{3}"] \\ 3 \arrow[loop below, "1"] & 4 \arrow[loop below, "1"] \end{tikzcd}$$

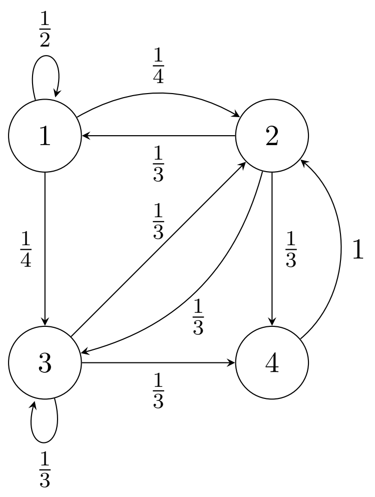
Answer the following questions assuming we start in the state specified and repeatedly transition until reaching an absorbing state:

(a) If the chain starts in state 1, what is the probability the absorbing state reached will be 4?

(b) Starting from state 1, what is the expected number of times the chain will be in state 2 before absorption?

(c) Starting from state 2, what is the expected number of transitions we need to take to reach an absorbing state?

Task 1.2 (12 pts). Consider the following Markov chain:



(a) What is the mean first passage time from state 1 to state 3?

(b) What is the mean first passage time from state 3 to state 1?

(c) What is the mean recurrence time for state 4?

Task 1.3 (6 pts). A gambler repeatedly plays a game in which she wins \$1 with probability $\frac{1}{2}$ and loses \$1 with probability $\frac{1}{2}$. She starts with \$2. She keeps playing the game until she has either \$0 or \$5.

(a) Represent the above scenario as a Markov chain, where the states correspond to how much money the gambler has, and the states corresponding to \$0 and \$5 are absorbing states. You can either draw a picture or write the transition matrix. Make it clear how much money the gambler has in each of the states.

(b) What is the probability that the gambler leaves with \$5?

2 Coding

Given a deck of cards, we would like to shuffle the cards to randomize their ordering. A standard way to do this is the so called *riffle shuffle*: we split the deck into two roughly even halves, and then interleave cards from the halves together by dropping cards from each half on top of one another. A natural question to ask is how many shuffles we have to do to get the ordering to be random enough.

In this question, we will write code to explore this question for a simpler form of shuffling called *top-to-random* shuffling. In the top-to-random shuffle, we take the top card from the deck and put it in a random position in the deck, each position being equally likely. (The random position can include leaving it at the top of the deck.).

For example, suppose we have a deck containing cards numbered 1 through 10. Imagine that initially the position of the cards is (from top to bottom):

$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$

Then after one top-to-random shuffle, the order might be:

$$2\ 3\ 1\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$

so that 2 is now at the top. Intuitively, the deck is still far from "random" after a single top-to-random shuffle. But if we repeat the top-to-random many, many times, then eventually it seems that the position of cards will be close to a uniformly distributed random permutation.

We can represent this process as a Markov chain, where the state is the current order of the cards in the deck, and each transition is a single round of top-to-bottom shuffle. Throughout, we will assume cards are just numbered, and do not have suits or anything like that.

Task 2.1 (1 pts). If the deck has 10 different cards, how many states does this Markov chain have?

Task 2.2 (2 pts). Write a function called `top_to_random(l, n, printing)` that takes a list `l` and performs `n` iterations of the top-to-random shuffle starting from `l`, and returns the result. If `printing` is `True`, then the routine should print the state of the list after each iteration. In your write-up, include the output of running `top_to_random([1, 2, 3, 4, 5], 10, True)`.

Task 2.3 (3 pts). Write a function `monte_carlo(l, n, k, r)` which uses Monte Carlo simulation to estimate the probability that the card with value `n` is at the top of the deck after `k` iterations of shuffling, starting from the list `l`. The procedure should do `r` Monte Carlo trials to estimate this probability.

In your write-up, for each value of k in the set $\{10, 25, 50, 75, 100\}$, include the estimated probabilities when l is the list:

$$1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$

$n = 10$, and $r = 200000$.

A Linear Algebra in Python

We can use the `numpy` library in Python to do various calculations with matrices. See `https://docs.scipy.org/doc/numpy/reference/routines.linalg.html` for the linear algebra library documentation. Below is a brief overview of the main commands. As usual, we can start by importing the `numpy` library:

```
>>> from numpy import *
```

Use `array` to define a matrix. For example, the following command defines the matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
>>> M = array([[1, 2], [3, 4]])
```

We can add together two matrices just using `+`:

```
>>> N = array([[5, 6], [7, 8]])
>>> M + N
array([[ 6,  8],
       [10, 12]])
```

Multiplication of matrices uses the `dot` function:

```
>>> dot(M, N)
array([[19, 22],
       [43, 50]])
```

We invert a matrix using `linalg.inv`:

```
>>> Minv = linalg.inv(M)
>>> dot(M, Minv)
array([[1.0000000e+00, 0.0000000e+00],
       [8.8817842e-16, 1.0000000e+00]])
```

Due to rounding errors, the product of `M` and its inverse is not exactly, the identity matrix, but it is close. `eye(n)` returns the $n \times n$ identity matrix:

```
>>> eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Finally, we can compute *right* eigenvectors using `linalg.eig`. This returns both a list of eigenvalues, and then an array of eigenvectors for those eigenvalues. Where the ith column of this array is an eigenvector for the ith eigenvalue.

```
>>> evals, evects = linalg.eig(M)
>>> evals
array([-0.37228132,  5.37228132])
>>> evects
array([[-0.82456484, -0.41597356],
       [ 0.56576746, -0.90937671]])
>>> # Get the eigenvector for the first eigenvalue:
```

```
... v1 = [vects[i][0] for i in range(2)]
>>> # Check that it's actually an eigenvector with that value
... dot(M, v1)
array([ 0.30697009, -0.21062466])
>>> dot(evals[0], v1)
array([ 0.30697009, -0.21062466])
```

How can we find the *left* row eigenvectors? The key is that the left eigenvectors of a matrix can be found by computing the *right* eigenvectors of the *transpose* of the matrix. The transpose makes the rows of a matrix into columns and vice versa. Finding the eigenvectors of the transpose will give us column vectors, so to get back to row vectors we transpose again:

```
>>> leval, levects = linalg.eig(transpose(M))
>>> leval
array([-0.37228132,  5.37228132])
>>> levects
array([[-0.90937671, -0.56576746],
       [ 0.41597356, -0.82456484]])
>>> lv1 = transpose([levects[i][0] for i in range(2)])
>>> dot(lv1, M)
array([ 0.33854396, -0.15485919])
>>> dot(leval[0], lv1)
array([ 0.33854396, -0.15485919])
```

Sometimes the eigenvectors you get back will be complex numbers with an "imaginary" component. You can extract a real number out using the function `real`.