MAT 245 Lab 2

A simple Markov chain.

Background. Suppose there is a discrete time system $\mathcal{S}$ that we want to model. We have the following information:

- at each time point, $\mathcal{S}$ must be in one of 3 states labelled by $\{1, 2, 3\}$;
- $\mathcal{S}$ may transition between two states at any time increment;
- for every state i, we know the probability of each transition $i \to 1$, $i \to 2$, $i \to 3$;
- the transition probabilities are constant in time.

Fixed time. We model $\mathcal{S}$ at time $k \in \mathbb{N}$ using a vector

$$X_k = (x_k^1, x_k^2, x_k^3).$$

Here x_k^i gives the probability that $\mathcal{S}$ is in state i at time k. Since $\mathcal{S}$ must be in one of these states,

$$x_0^1 + x_0^2 + x_0^3 = 1.$$

In other words, X_0 is a probability distribution over the states $\{1, 2, 3\}$. For example, if at time k we know with probability 1 that $\mathcal{S}$ is in state 1 then $X_k = (1, 0, 0)$.. If instead there is a 50%/50% split between states 1 and 3 then $X_k = (1/2, 0, 1/2)$.

Transitions. Next we need to model the transition process. We already know the probability p_{ij} of transitioning from state i to state j. Combining all the transition probabilities gives a matrix:

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix}.$$

Since $\mathcal{S}$ must transition at each step (even if the transition is from i to itself), so the rows of P satisfy

$$p_{i1} + p_{i2} + p_{i3} = 1.$$

That is, the rows of P are probability distributions. A matrix like this is called *stochastic.*

Forecasting. If we know X_k we can compute X_{k+1} using P:

$$X_{k+1} = X_k \cdot P.$$

Therefore, once we know the initial configuration X_0, we get a distribution representing the k^{th} time step by computing

$$\begin{aligned} X_k &= X_{k-1} \cdot P \\ &= X_{k-2} \cdot P \cdot P \\ &= X_0 \cdot P^k. \end{aligned} \tag{1}$$

Implementing the model.

Background. The model of $\mathcal{S}$ constructed above is an example of a *Markov chain*. We're now going to implement a simple 3-state Markov chain. The matrix P of transitions will be randomly generated. The main point of this exercise is to gain experience with `numpy`. You are encouraged to play around with the `numpy` functions and objects in the interpreter as you follow along.

Steps. **1.** Import `numpy`.

2. The function `numpy.random.rand(`n, m`)` is a `numpy` function of two parameters. It is used to generate an $n \times m$ matrix with random entries chosen uniformly from $[0, 1]$. Use this function to initialize a 3×3 `numpy` matrix called `P_unscaled` with random entries.

3. The matrix `P_unscaled` is not a transition matrix, since its rows do not necessarily sum to 1. To fix this:

i Compute the sum of each row of `P_unscaled`. This can be done via a call to the `sum` method of `P_unscaled`:

```
 row_sums = P_unscaled.sum(axis=1).reshape(3, 1)
```

The variable `row_sums` is a `numpy` matrix with one column and three rows. The entry in row i is the sum of the i^{th} row of `P_unscaled`. (To improve your intuition, run this command in the interpreter and investigate the `numpy` matrix returned as output.)

ii Create a new matrix `P` by dividing each row of `P_unscaled` by its sum:

```
P = P_unscaled / row_sums
```

Pay close attention to how the rows of `P_unscaled` are matched with the rows of `row_sums` during the division process. It may help to run this command in the interpreter as well.

You may want to verify that `P` is a stochastic matrix. You can do so by summing its rows:

```
print(P.sum(axis=1))

## prints array([1., 1., 1.])
```

4. Create an initial configuration corresponding to state 2 with probability 1:

```
X_0 = numpy.array([0.0, 1.0, 0.0])
```

5. Compute the distribution `X_5` that gives the evolved configuration of our system after 5 time steps. Recalling equation 1, we get `X_5` by computing compute $\texttt{X\_0} \cdot \texttt{P}^5$. The matrix power $\texttt{P}^k$ can be computed using the `numpy.linalg.matrix_power` function. The product of `X_0` and $\texttt{P}^5$ can then be computed using the `dot` method. Combined into one line, this looks like

```
X_5 = X_0.dot(numpy.linalg.matrix_power(P, 5))
```

6. Repeat steps 4 and 5 to compute the configuration of the system after 7 steps when given an initial state vector with probabilities:

$$\texttt{state } 1 \mapsto 0.25, \quad \texttt{state } 2 \mapsto 0.25, \quad \texttt{state } 3 \mapsto 0.5$$

Plotting the (p-norm) unit ball.

Goals.

(1) Generate the points of the unit ball $|x|^p + |y|^p = 1$ using `numpy` for $p = 1, 2, 3, 5, 10$. Make a 2-D plot of these points using the `matplotlib` library.
(2) Generate the points of the unit ball $|x|^p + |y|^p + |z|^p = 1$ using `numpy` for $p = 1, 2, 3, 5, 10$. Make a 3-D plot of these points using the `matplotlab` library.

Hints.

- In the 2-D case write y as a function of x, given $|x|^p + |y|^p = 1$. Then write a python function called `y_fn` that takes p and x as arguments and returns $y(x)$.
- If we have a function defined by

```
def func(x, p):
    return x ** p
```

then the `numpy.vectorize` function has the following effect:

```
vectorized_func = numpy.vectorize(func)
print(vectorized_func([1, 2, 3, 4, 5], 2))
## prints array([1, 4, 9, 16, 25])
```

- Use `matplotlib.pyplot.subplot` to show multiple plots at once. The `subplot` function arranges the individual plots into an $n \times m$ grid. It takes an argument `nmk` where `n` = *number of rows*, `m` = *number of columns*, and `k` specifies we want the k^{th} sub-plot (labelled left-to-right and top-to-bottom). Below is an example that creates an 3×2 grid of subplots:

```
plt.figure(1)
plt.subplot(321)
## do plotting for p = 1
plt.gca().set_aspect('equal', adjustable='box') ## this line scales the axes

plt.subplot(322)
## do plotting for p = 2
plt.gca().set_aspect('equal', adjustable='box')

## etc ...

plt.show()
```

- If you have time, adapt your solution for 2D unit balls for the 3D case. You will need to use `mpl_toolkits.mplot3d` In particular, try using `mpl_toolkits.mplot3d.scatter` to make a scatter plot of the points in the 3D unit ball.