## Task 1. Markov Chains

*Example: Flowers.* Consider a particular variety of flower which can take one of three colours: red, white and pink. Suppose after each year, 40% of red flowers remain red but 60% of red flowers mutate to pink. For pink flowers, 30% mutate to red, 50% remain as pink, and 20% mutate to white. For white flowers, 90% remain as white and 10% mutate to pink. We can capture these probabilities in a transition matrix 𝑃 where the columns correspond to red, pink and white flowers in the *current* year and the rows correspond to red, pink and white flowers in the *next* year.

0.4 0.3 0

𝑃 = [0.6 0.5 0.1]

0 0.2 0.9

If a certain garden initially has 80 of this variety of red flowers and 20 of this variety of white flowers then after one year:

80 32

𝑃 [ 0 ] = [50]

20 18

so we would expect to see 32 red, 50 pink and 18 white flowers in the garden. If we look at the garden after 5 years:

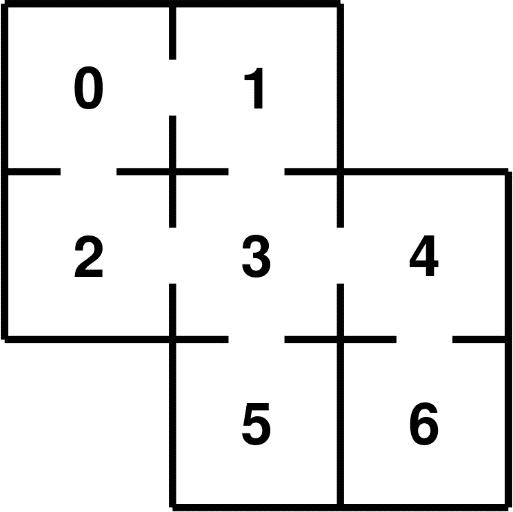
80 80 20.88

𝑃𝑃𝑃𝑃𝑃 [ 0 ] = 𝑃5 [ 0 ] = [37.08]

20 20 42.04

so we would expect to see approximately 21 red, 37 pink and 42 white flowers in the garden. *This is an example of a system called a “Markov chain”. The next state of the system depends only on its present state and does not depend on the earlier history of the system.*

*Assessment: Mice in a maze.* Consider the closed maze shown below containing seven rooms (labelled 0 to 6) with doorways connecting the rooms. A number of mice are placed in the maze and the mice then move randomly from room to room. Each time step, a mouse either stays in the same room (with probability 0.6) or leaves the room it is in by choosing at random (equally likely) one of the doors out of the room.





1. Construct a 7 × 7 transition matrix 𝑃 (as a numpy array) to model the mice moving around the maze as a Markov chain.
2. Suppose that 50 mice are placed in Room 0 and 90 mice are placed in Room 1. All other rooms are initially empty. Use numpy to calculate how many mice we would expect to see in each room at the end of each time step for each of the next 30 time steps. Use Python to plot your results on a line graph showing one line for each room, together with a legend. Briefly comment on what you observe. *You might find* numpy.linalg.matrix\_power() *helpful.*
3. Use numpy to find the eigenvectors of the transition matrix 𝑃 from part (1), and explain how one of these eigenvectors is related to the number of mice we would expect to see in each

room in the “long-run steady state”. Also, for each room 𝑗, find the expected number of time steps for a particular mouse initially located in room 𝑗 to first return to room 𝑗. *Hint: if* 𝑝𝑗 *is the probability that the particular mouse is in room* 𝑗 *in the “long-run steady state” then the expected first return time is given by* 1/𝑝𝑗*.*

1. Suppose the mouse population is in the “long run steady state” from part (3) when a large piece of poisoned cheese is placed in Room 6 so that any mouse that enters that room instantly dies.
   1. Modify the transition matrix 𝑃 from part (1) and estimate the number of time steps until 80% of the mice have died.
   2. The expected number of time steps needed for a mouse starting from room 𝑖 to reach room 𝑘 for the first time can be calculated using matrix operations and is denoted 𝜇𝑖𝑘 . For a particular destination room 𝑘, let 𝑁 be the transition matrix 𝑃 from part (4)(a) but with row 𝑘 and column 𝑘 both deleted, and let 𝐼 be the 6 × 6 identity matrix. The sum of each column of the matrix (𝐼 − 𝑁)−1 gives the values 𝜇𝑖𝑘 . Use this method to find the expected number of time steps needed for a mouse to die, for each of the possible rooms in the maze that the mouse could start from. Briefly comment on what you observe.

## Task 2. Eigenfaces

The Olivetti Faces dataset consists of 400 images of human faces. Five examples are given below.



Each image is 64 pixels wide by 64 pixels high. However, the Python code below assembles the 400 images as columns of the matrix Xall. Each image is greyscale (not colour) and all pixel values are between 0 and 1. The matrix Xsub is the last 300 columns of the matrix Xall.

*The only Python libraries you may use in this task are numpy and matplotlib, except that you may use sklearn only to load the dataset as in the Python code below*

import sklearn.datasets

faces = sklearn.datasets.fetch\_olivetti\_faces()

Xall = faces.data.T print(Xall.shape) print(Xall.min(), Xall.max()) Xsub = Xall[:,100:]

1. Use numpy to find the mean of each row of Xsub and reshape as a 4096 × 1 vector (call this xbar). Adapt the Python code below to visualise this “mean face” and compare it to the image stored in the column of Xall corresponding to the last two digits of your Student ID.

plt.imshow(*\*\*something\*\**, cmap=plt.cm.gray, vmin=0, vmax=1)

1. Use numpy.cov() to calculate the covariance matrix of Xsub, giving a 4096 × 4096 matrix (call this matrix 𝐶). Then use numpy.linalg.eigh() to find the eigenvalues (𝑉) and eigenvectors (columns of the matrix 𝑃) of 𝐶. Use the Python code below to reverse the entries in 𝑉 and the columns of 𝑃. Let 𝐷 be the diagonal matrix consisting of the elements of 𝑉 along its diagonal. Use numpy to confirm that 𝑃 is an *orthogonal* matrix and that 𝐶 = 𝑃𝐷𝑃𝑇.

*Make sure you use* numpy.linalg.eigh() *not* numpy.linalg.eig() *for this part.*

V = V[::-1]

P = P[:,::-1]

1. In the following Python code, we apply 𝑃𝑇 (as a matrix transformation) to the faces in Xsub, where xbar is the “mean face” (a 4096 × 1 vector) from part (1).

Ysub = P.T@(Xsub-xbar)

* 1. The eigenvectors of 𝐶 (i.e. the columns of 𝑃) are called “eigenfaces” (Turk and Pentland, 1991). Visualise the eigenfaces corresponding to the largest 10 eigenvalues. Comment on what you discover. *You will need to remove vmin and vmax from the* plt.imshow() *example given above.*
  2. Show (by example) that if y is any one column of Ysub then P@y+xbar perfectly recreates the corresponding column of Xsub. *This shows that each “face” in Xsub is a “linear combination” of eigenfaces plus the “mean face”.*

Although 𝑃 is a 4096 × 4096 matrix, generally only those eigenfaces corresponding to the largest eigenvalues of 𝐶 are “useful”. If there are 𝑘 such eigenvalues (with 𝑘 ≥ 2), let Xpartial be the matrix formed from matrix multiplication of the first 𝑘 columns of 𝑃 and the first 𝑘 rows of Ysub, and adding xbar. Investigate how well Xpartial approximates Xsub for different values of 𝑘 by using the numpy.linalg.norm() function applied to Xpartial minus Xsub (this provides a measure for “goodness of fit”). *Build an appropriate line graph as part of your investigation*

*Reference*

Turk, M.A. and Pentland, A.P. Face recognition using Eigenfaces. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 1991, pp 586-591. <https://ieeexplore.ieee.org/document/139758>