Question 1: Theme Park of Doom

Luckily our Leslie-modelling has paid off, and the theme park is mosquito-free!

As part of an investigation into which rides are the most popular, you have been tasked with modelling a theme park with rides labelled A, B, C, D, E, F, G, H, I, and J. The layout of the park is given on the following page, with the labelled circles representing the rides (see part (f) below), and the brown lines representing the paths between the rides.

The behaviour of a typical park visitor is as follows: A visitor will either go on the same ride again, or they will go to a ride that is adjacent (connected by a path) to the one they just rode. The choice will be made at random with the probability that the visitor will go to an adjacent ride 3 times the probability that they will go on the same ride again. For example, if the visitor just went on ride A, the next ride would be A with probability $\frac{1}{7}$, B with probability $\frac{3}{7}$, or F with probability $\frac{3}{7}$.

(a) What is the long-term probability distribution for the locations of the visitors? Give your answer in terms of percentages.

(b) The visitors won't spend infinite time at the theme park, so perhaps the long-term distribution isn't the important thing. If all the visitors start at ride A, where are they in the following steps? How many steps does it take to get the relative error with respect to the long-term distribution (measured with the infinity norm) down below 1%? Present a plot of your results; you may find `plt.stackplot` useful.

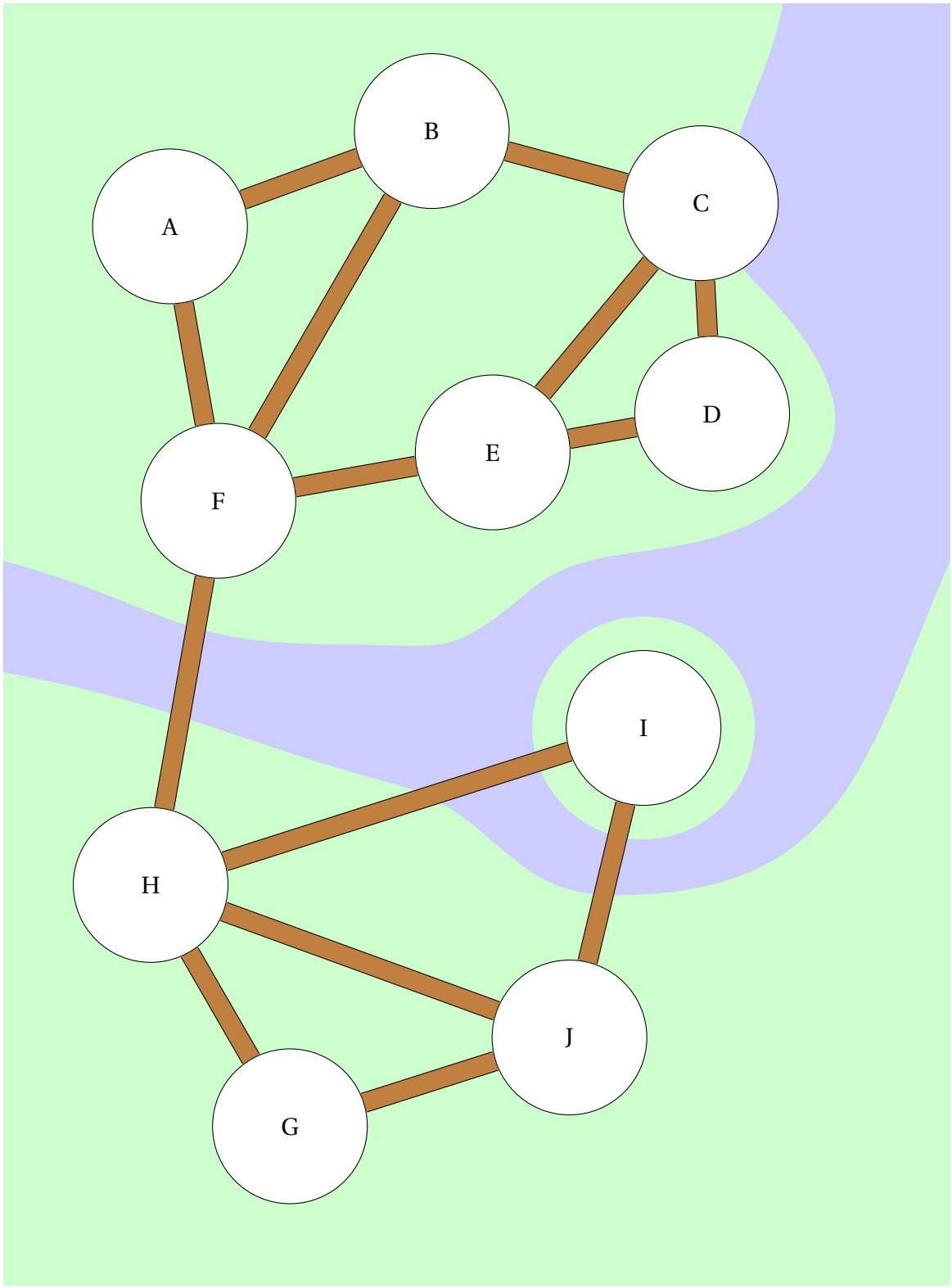
(c) What would change if the visitors were 5 times as likely to go to adjacent rides instead of 3 times as likely? What would happen if the park disallowed repeating rides immediately? Present predicted results.

(d) You have enough money to either build one bridge and one path, or three land-based paths. What could you change to improve the evenness of the ride distribution? Describe the change and present predicted results.

(e) Would you recommend to an employer that a Markov chain model be used in a situation like this? Discuss the advantages and limitations of Markov chains for this problem. This is an open-ended question, where you may want to consider things such as the Markov chain's convergence speed to long-term behaviour, human behaviour, and the effects of a real-world park layout.

(f) (Optional) Draw your best guess at what some rides might look like, where the ride names are:

A	Algebraic Abyss
B	Boolean Bumpercars
C	Covariance Cove
D	Derivative Drop
E	Exponential Exhilaration
F	Fourier Flumes
G	Gaussian Gauntlet
H	Heaviside Havoc
I	Integral Island
J	Jacobi Jousting


B
A
C
D
E
F
I
H
J
G

Question 2: Flop multiplication

How expensive is multiplying two matrices? Let's find out!

In this question you will compare real-world timings to theoretical flop-count estimates for multiplying two $n \times n$ matrices. You can use Python's built-in `time` module to time how long things take: `time.time()` will return a floating point number of how many seconds (and fractional part) have passed since 1970-01-01 00:00. The following skeleton code will help.

```
import numpy as np
import matplotlib.pyplot as plt
import time

times = []
# [... any other setup stuff ...]

for n in sizes:
    A = np.random.rand(n, n) #make a random matrix
    t0 = time.time() #start time
    # [... whatever is being timed ...]
    t1 = time.time() #stop time
    times.append(t1 - t0) #time taken, in seconds

# [... make a glorious plot! ...]
```

(a) Measure reasonably accurate time estimates for how long it takes to calculate A^2 for a broad range of matrix sizes up to and including 5000. Plot the times against the matrix size.

(Important considerations: How many matrix sizes do you need? Do you get the same timings each time you run? How much are other things running on your computer affecting the results? Do you need to repeat something multiple times and take the minimum? Are you timing just what you care about or including extra stuff?)

(b) What is the theoretical flop count of multiplying two $n \times n$ matrices? Find a scaling factor (effectively time per flop) that makes the theoretical curve go through the time for your largest matrix, and add the theoretical prediction curve to your plot.

Hint: is this scaling factor the same every time?

(c) How well do theory and practice agree in this case? If they disagree, what might be the reasons?

Question 3: Complexities of Power

Python and NumPy can work with complex numbers just as easily as real numbers. If you have already written code to do power methods, it will likely work with complex matrices without any changes.

To enter complex values in Python, put a `j` immediately after the imaginary part, e.g. `1+2j`. If you have a variable `c` with a complex value (number or array), `c.real` and `c.imag` give you the real and imaginary parts, respectively.

Let

$$A = \begin{bmatrix} -2+0.5j & 0.1 & -0.5+j & -0.1j & 0.1 \\ 0.2 & 3 & 0 & 0.1+0.1j & 0.2j \\ 0.1j & -0.1 & 3 & 2 & 0.1-0.1j \\ 1+0.2j & 1j & 0.3 & -4+j & 1+j \\ 0.1 & 0.2 & 1 & 0.1+0.1j & 3+0.1j \end{bmatrix}$$

where $j^2 = -1$. A has 5 distinct eigenvalues. There is a copy-and-paste-able code version of this on Learn.

(a) Use Gershgorin's Theorem to find approximate locations for the eigenvalues of A. Plot your discs in Python, like this:

```
fig, ax = plt.subplots()
ax.grid()
for i in range(A.shape[0]):
    ax.add_patch(plt.Circle((centres[i].real, centres[i].imag),
                            row_radii[i], color='blue', alpha=.2))
    ax.add_patch(plt.Circle((centres[i].real, centres[i].imag),
                            col_radii[i], color='green', alpha=.2))
ax.axis('equal')
```

(In some Python coding environments you may need `fig.show()` at the end to make it appear.)

(b) Use the shifted inverse power method to compute *all* the eigenvalues of A. Use a stopping condition of

$$\frac{\left|R\left(B^{-1}, \underline{\mathbf{x}}^{(k)}\right) - R\left(B^{-1}, \underline{\mathbf{x}}^{(k-1)}\right)\right|}{\left|R\left(B^{-1}, \underline{\mathbf{x}}^{(k)}\right)\right|} < 10^{-8},$$

where $B = A - qI$ is your shifted matrix. Note that the terms in this formula are available when you have just calculated $\underline{\mathbf{x}}^{(k+1)}$ (the stopping condition lags behind for efficiency).

Plot your eigenvalues on your disc plot with:

```
ax.plot(eigenvalues.real, eigenvalues.imag, 'ro')
```

Do the positions agree with your plot?

You may want to also plot NumPy's estimates of the eigenvalues (e.g. with `'kx'` instead of `'ro'`).

(c) What weaknesses of the shifted inverse power method did you need to take into account to find all the eigenvalues? Briefly describe how you worked around these weaknesses.