AMS 325 Homework 4
(Python Mini-projects)

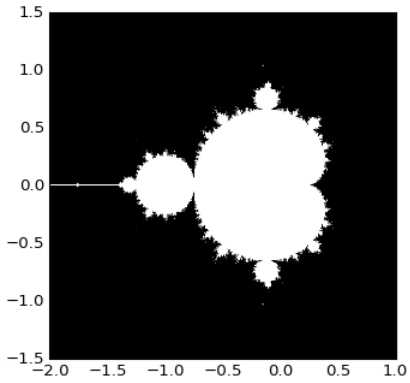
Due: June. 20th, 2022

In this homework, you will write Python scripts (.py files, instead of Jupyter Notebook) using NumPy, SciPy, and Matplotlib. As in Homework #2, you will also need to use github for the version control and submission of your source code (bonus).

1 Tasks

There are two tasks (mini-projects) for this assignment.

1.1 Task 1: Mandelbrot Sets



The Mandelbrot set is the set of complex numbers c for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from $z = 0$. In other words, the sequence $f_c(0)$, $f_c(f_c(0))$, etc. remains bounded in absolute value. In this task, you will write a Python script mandelbrot.py to compute the Mandelbrot fractal with the following Mandelbrot iteration on each point:

```
N_max = 50
threshold = 50
c = x + 1j*y

z = 0
for j in range(N_max):
    z = z**2 + c
```

A point (x, y) belongs to the Mandelbrot set if $|z| <$ threshold computed from the above iteration starting with $c = x + yi$. Do this computation as follows:

1. construct an $n \times n$ grid (2D array) of points (x, y) in range $[-2, 1] \times [-1.5, 1.5]$ (e.g., using numpy.meshgrid) and corresponding complex values $c = x + yi$ (note that the imaginary unit in Python is $1j$);

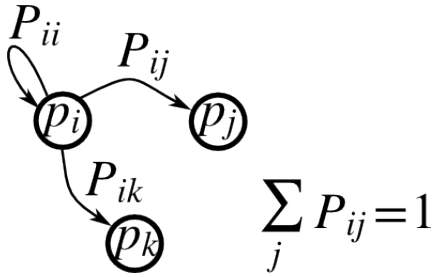
2. perform the iteration as outlined above to compute z for each complex value in the grid;
3. form a 2-D boolean array mask indicating which points are in the set (i.e., $|z| <$ threshoold);
4. save the result to an image using the commands:

```
import matplotlib.pyplot as plt
plt.imshow(mask.T, extent=[-2, 1, -1.5, 1.5])
plt.gray()
plt.savefig('mandelbrot.png')
```

5. organize the statements into a function that takes n, N_max, and threshold as input, add docstring and comments to the function, and experiment the function with different n so that you get an image resembling the one above.

1.2 Task 2: Markov Chain

The following diagram depicts a set of states and the transition between each pair of states, where p_i denotes the probability within the ith state and P_{ij} is the probability for state i to transition into state j.



Mathematically, we can represent the diagram with a vector p and a matrix P. The vector p of size n would describe the probability distribution on n states, with

$$0 \le p[i] \le 1 \qquad \text{and} \qquad \sum_i p[i] = 1.$$

Matrix P is a Markov chain transition matrix of size $n \times n$, where

$$0 \le P[i,j] \le 1,$$

with the sum of each row equal to 1 (i.e., $\sum_j P[i,j] = 1$ for all i). Given an old state p_{old}, the new state is then given by $P^T p_{old}$.

In this task, you need to write a Python script named markov_chain.py to perform the following:

1. Construct a random n-vector with non-negative entries and scale its entries so that the sum is 1. This computation gives us a probability distribution p.
2. Construct a random $n \times n$ (say $n = 5$) matrix with non-negative entries, and scale the entries so that the sum for each row is 1. This computation gives us a transition matrix P.
3. Starting from p as the initial state, compute the transition for N (say $N = 50$) steps (i.e., compute $p \leftarrow P^T p$ for N times).
4. Use the function np.linalg.eig to compute the eigenvector of P.T corresponding to the largest eigenvalue. Rescale the entries of the eigenvector so that its sum is equal to 1. Let the resulting vector be $p_{\text{stationary}}$ (or p_stationary).
5. Change the loop in step 3 to compute the norm of $p - p_{\text{stationary}}$ and plot the norms against i. Check whether the difference diminishes as the number of iterations increases.
6. Organize your code into a function so that it would take n and N as input and call the function by passing in $n = 5$ and $N = 50$. Document the function and test it with some other values of n and N.

You might find the following functions useful: np.random.rand, np.dot, np.sum, np.linalg.eig, np.argmax, etc.

2 Submission Instruction

1. Create a git repository on github.com and push mandelbrot.py and markov_chain.py to the repository (bonus). Add a README file for your repository, which should explain the files and how to run the code.

2. Share your github repository with me @kideuler on github. Your last commit should be before the deadline (bonus).

3. Write a short report and upload the PDF file of the report onto Microsoft Teams. This report should contain the URL of your github repository that you have shared with me. It should also include a brief description of the algorithm you used, a discussion of the performance of your code with the plots embedded. You can use Microsoft Word, Google Doc, or a LaTeX-based software to write your report. Note: if you decide not to use github upload all your code including the README file to blackboard

3 Grading Rubric

- 70% for the correctness of the code (35% for each min-project).
- 15% for the documentation, including the help texts, comments, and the README file.
- 15% for the report.
- 10% for the correct use of git (bonus).

It is important that you write the code on your own. Do not copy the code from any source.