Math 152, Exploration 2

Martin H. Weissman

1 General guidelines

You and your group should explore **one** of the following three projects using your mathematics and programming skills. At the end, you should turn in a Python notebook (ipynb) file, by putting it on GitHub, and submitting the link to the file on Canvas. The focus of the notebook should be **two visualizations**: one created with **MatPlotLib** and one with **PIL**. Your notebook should contain text cells and well-documented code cells as usual. You should describe your exploration, and explicit questions that are answered by your visualizations. Describe what the visualization means, so that it is easy to interpret. Describe your choices in making the visualization (why did you choose certain colors, shapes, layout, axis scaling, etc.?)

With MatPlotLib, you should create a more traditional data graph, e.g. composed of line or scatterplots, a histogram, etc. You might look at MatPlotLib (or Seaborn) galleries for ideas. The MatPlotLib plot can certainly be more than a single line on an axis – overlaying different data, choosing colors, shapes, etc., should be considered. There should be one central "plot object," though it may be composed of a grid of small and closely connected "subplots" if that helps get the idea across. But there should not be multiple plots – choose one method of getting your idea across, for the final draft.

The other visualization should be created by PIL – this provides much more freedom, but fewer tools to make traditional plots. In this image, you should create your image pixel by pixel (perhaps overlaying lines or rectangles or circles if you like). The position and color of every dot and shape should represent something. This plot can be more artistic, but must still allow the viewer to learn something about your exploration. The PIL image may require more information in nearby text cells, since your image will probably not have text labels, axes, etc. Your PIL image should be at least 1000 x 1000 pixels in size, and at most 3000 x 3000.

There are a lot of papers written about these topics. You shouldn't need to look at these papers (maybe one or two, if you want), but if you

do use results you must cite them. For citation, please provide complete bibliographic information in your notebook, and web links to papers online. Limit your references to published papers and preprints, by looking at (1) papers published in scholarly math journals, (2) papers published by scientific journals, e.g., in Quanta, and (3) preprints from arxiv.org. If you want to look at Wikipedia, it's fine, but please use it primarily for the references (typically given at the bottom of the Wikipedia page.

Academic integrity: If you write something without citation, you are taking credit for what you wrote. You may only take credit for ideas that you thought of. So do not write things that you found elsewhere without citation. The same goes for Python code. Do not copy code snippets without citation. Even with citation, you will probably not receive full credit if you simply copy your code from elsewhere.

Remember... just choose **one** of the following projects to work on!

Grading rubric: 40 points will be possible.

10 points for code – clarity and efficiency.

10 points for text – clear writing, easy to follow, mathematically precise and correct.

10 points for the matplotlib image – depth of information, choices of color and style, success in conveying a message

10 points for the PIL image – depth of information (information per pixel), choice of color and style, success in conveying a concept (not necessarily a single message).

Project 1: Newton's method and basins of attraction

Newton's method provides a way to quickly and accurately approximate the solutions to certain equations. The general setup is that you want to find a root of a function f, i.e., a number x such that $f(x) = 0$. You begin with a guess x_0, and then you define

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

for all $n \geq 0$. Note that f' denotes the derivative of f – we only work with nice differentiable functions.

Interesting questions about Newton's method include: does it always yield a sequence that converges to a root? If so, how quickly does it converge to the root? If there are multiple roots, which root does it converge to? How do the previous two questions depend on the starting guess x_0?

For this project, it is recommended that you limit your attention to some pretty simple kinds of functions. A good choice would be to consider cubic polynomials, e.g. $f(x) = x^3 - 2x^2 - 11x + 12$ (which has three real roots) and $f(x) = x^3 - 1$ (which has one real root and two complex roots).

Given such a function f, and a root r (so $f(r) = 0$), the *basin of attraction* for r is the set of complex numbers z such that Newton's method – beginning with the guess z – converges to r. For example, if you are looking for a root of $f(x) = x^3 - 2x^2 - 11x + 12$, and you start with the guess 2.35287527, then Newton's method will converge to the root $r = 4$. So we say that 2.35287527 lies in the basin of attraction of 4. On the other hand, if you start with the guess 2.35284172, then Newton's method will converge to the root $r = -3$. So 2.35284172 lies in the basin of attraction of -3. Wild!

Create two visualizations (one with MatPlotLib and one with PIL) that show something about the interesting questions mentioned above. For example, you might make plots with MatPlotLib which illustrate the rate of convergence of Newton's method. And you might make a plot with PIL that shows all the "basins of attraction" for Newton's method for a single f. Numpy handles arrays of complex numbers very nicely, and such basins of attraction are sources of lots of cool fractal pictures.

If you want to explore something of your own invention, contact Marty – he'll tell you if you're heading in a promising direction, or to a dead end.

Please read the general guidelines for more!

Project 2: Seeing the primes

The prime numbers are an infinite set of positive integers (as proven by Euclid), and there are many unsolved questions about their distribution. How many primes are there in a typical interval, say from 1 up to a large number X, or between two large numbers X and Y? Which gaps are most common between consecutive prime numbers (e.g., the gap of 2 between 3 and 5, or the gap of 6 between 23 and 29)? How do the last digits of primes behave?

The number of primes between 1 and X is approximately $Li(X) = \int_2^X \frac{1}{\log(t)} dt$ – this is called the prime number theorem, and the function $Li(X)$ is called the "logarithmic integral" function. It is implemented in mpmath, for example. But what is the error in this approximation? How close is this approximation, when X grows large? This is related to the Riemann Hypothesis, an open math conjecture with a million-dollar prize attached!

Create two visualizations (one with MatPlotLib and one with PIL) that show something about the interesting questions above, or other properties of the prime numbers that you explore. Contact Marty with your creative ideas, and he'll tell you if you're heading towards something promising, or a dead end. Please feel free to reuse code from earlier notebooks, e.g., the sieve of Eratosthenes to produce a long list of prime numbers. But go (far) beyond the analysis of primes that we performed in the earlier notebook.

Please read the general guidelines for more!

Project 3: The random walk

The simplest random walk is the following: start at 0. At every step of the walk, flip a coin. If it's "heads" then go right by one unit. If it's "tails" then go left by one unit. Repeat. There are many things known about random walks. For example, after N steps, you can expect (i.e., on average) to be about $\sqrt{N}$ units away from the origin.

In this project, you should explore variants of random walks and resulting statistical properties. Here are some interesting variants:

1. Try the random walk in two dimensions. You might choose a random vector (of length 1) for each step, or just a random direction: up, down, left, or right. 2. Put up a wall, or two: in the one-dimensional case, imagine there's a wall and you can never go into the negative numbers. If you are at zero and flip tails, you just stay where you are. 3. Put in a floor, and a bit of gravity. To model the atmosphere, you might imagine molecules of gas. They walk randomly up or down, but have a slight preference for moving down. And they can't move below the ground – they just bounce off. 4. Stay in a box – a rectangle, for example. If you walk into the edge, you bounce off. 5. What if there are two "people" walking at the same time. They start at the different places, and are not allowed to pass through each other (they bounce off).

Interesting things happen if you run a random walk experiment many times, e.g., 1000 steps of your walk, repeated 1000 times over (using numpy to make things run quickly!). How are the endpoints distributed (along the line, in space, in a box, etc.)? What is the typical distance between starting point and ending point? Do you ever return to the place where you started? These kind of questions are mathematically interesting, and applicable in the sciences.

Explore some of these variants and questions to find something interesting. Create two visualizations (one in MatPlotLib and one with PIL) that exhibit your findings. If you want to explore something of your own invention, contact Marty – he'll tell you if you're heading in a promising direction, or to a dead end.

Please read the general guidelines for more!