Math 104A - Numerical Analysis I

Approximation of functions

Jea-Hyun Park

University of California Santa Barbara

Fall 2022

Introduction

Polynomial Interpolation

Problem of interest

Throughout the section, we want to answer the problem (and some important properties): given the data below, find a polynomial $y = p(x)$ of minimal degree *interpolating* it.

x	x_0	x_1	x_2	$\cdots$	x_n
y	y_0	y_1	y_2	$\cdots$	y_n

- **Subjective questions**: Does this problem make sense? What should we check to make this problem meaningful? What do your guts tell you before even start studying it?

Polynomial interpolation

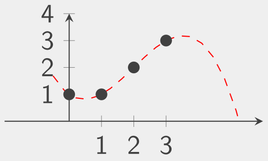
Theorem (Unique interpolation theorem)

If $x_0, x_1, \cdots, x_n$ are distinct real, for arbitrary values $y_0, y_1, \cdots, y_n$, there is a unique polynomial $p \in \Pi_n$ such that $p(x_i) = y_i$ $(0 \leq i \leq n)$.

Proof 1.

Vandermonde – next few slides . □

- Notation: $\Pi_n := \{$ polynomials of degree at most $n\}$.
- Notice that the degrees of freedom match: (n+1) values to interpolate and (n+1) coefficients we can tune in $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$.



Vandermonde matrix

Idea: Brute force.
Set $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, and require the conditions.

$$\begin{aligned}
p(x_0) &= a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0 \\
p(x_1) &= a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = y_1 \\
&\vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
p(x_n) &= a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = y_n
\end{aligned}$$

In matrix form,

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- The coefficient matrix is called **Vandermonde matrix**.

Vandermonde matrix

Theorem

$\det(V) = \prod_{0 \le i < j \le n} (x_j - x_i)$, *where V is the Vandermonde matrix:*

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}.$$

Proof.

1. $\det(V)$ is a polynomial in $x_0, x_1, \cdots, x_n$.
2. If $x_0 = x_1$, the first two rows are identical. Thus, $\det(V) = 0$. The factor theorem asserts $(x_0 - x_1)$ divides $\det(V)$. (Think of x_0 as a variable, say x, and x_1 as a number, say 1, and plug in $x = 1$.) Do the same for $x_i = x_j$ $(i \neq j)$ and conclude $(x_i - x_j)$ divides $\det(V)$. Therefore, $\det(V) = (\text{something}) \prod_{0 \le i < j \le n} (x_j - x_i)$.

Vandermonde matrix

Proof.

3. Recall Liebniz formula for the determinant: sum of $\pm$(product of entries taken from distinct columns while scanning rows from the top to the bottom). '$+$' is assigned when the order of column index chosen is an *even permutation* of $(0, 1, \cdots, n)$ and '$-$' when it is an *odd permutation*.
4. Observe that 'something' must be a constant since the order of the polynomial is $n(n+1)/2 = 0 + 1 + 2 + \cdots + n$ from both Liebniz formula and the product form.
5. Comparing the term $x_1 x_2^2 \cdots x_n^n$, we realize that the constant must be 1: this term appears only once with '$+$' in the Liebniz formula and we have (something)$x_1 x_2^2 \cdots x_n^n$ by expanding (something) $\prod_{0 \le i < j \le n} (x_j - x_i)$ choosing only x_j's.

$\square$

Polynomial interpolation

Theorem (Unique interpolation theorem)

If $x_0, x_1, \cdots, x_n$ are distinct real, for arbitrary values $y_0, y_1, \cdots, y_n$, there is a unique polynomial $p \in \Pi_n$ such that $p(x_i) = y_i$ $(0 \leq i \leq n)$.

Proof 1.

Since the nodes are distinct, the determinant of the Vandermonde matrix $\det(V) = \prod_{0 \leq i < j \leq n} (x_j - x_i)$ nonzero, hence the matrix is invertible. Therefore, we have a unique solution $[a_0, a_1, \cdots, a_n]^T$ in the Vandermonde system for any prescribed $[y_0, y_1, \cdots, y_n]^T$. That is, $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \in \Pi_n$ is the unique polynomial we want. $\square$

Newton form interpolation

Theorem (Unique interpolation theorem - duplicate)

If $x_0, x_1, \cdots, x_n$ are distinct real, for arbitrary values $y_0, y_1, \cdots, y_n$, there is a unique polynomial $p \in \Pi_n$ such that $p(x_i) = y_i$ $(0 \le i \le n)$.

Proof 2.

Board work. $\square$

Example

Following the previous proof, find a polynomial (of minimal degree) interpolating

x	0	1	2	3
y	1	1	2	3

- The way the polynomial organized in the proof is called **Newton form**.

Horner's algorithm: evaluating polynomials

By storing coefficients, we can only *encode* a polynomial

$$\begin{aligned} p(x) &= 1 + 0 \cdot x + \frac{1}{2}x(x-1) - \frac{1}{6}x(x-1)(x-2) \\ &= -\frac{1}{6}x^3 + x^2 - \frac{5}{6}x + 1. \end{aligned}$$

We need to compute the output just to know one function value.

For practical reasons, the following **nested multiplication** or **Horner's algorithm** is better than following the math expression.

$$\left(\left(-\frac{1}{6}(x-2) + \frac{1}{2}\right)(x-1)\right)x + 1$$

In algorithm form, this reads much nicer:

```
$u \leftarrow c_k$;
for $i \leftarrow k-1$ to $0$ do
    $u \leftarrow (t - x_i)u + c_k$;
end
```

- This is **only for evaluating** a polynomial after finding an interpolation. Don't mix this with how to find Newton form interpolations.
- Multiplications are more expensive than additions in computing. Count the multiplications to see the difference.
- This is purely computational. Mathematically, they are the same.

Lagrange interpolation

Idea: Find a basis of Π_n that makes interpolating procedure simple.
In particular, if we can find $\ell_i(x) \in \Pi_n$ such that

$$\ell_i(x_j) = \delta_{ij}, \tag{1}$$

then, (we will call the way it's written **Lagrange form**)

$$p(x) = \sum_{i=0}^{n} y_i \ell_i(x).$$

Definition (Lagrange basis or cardinal functions)

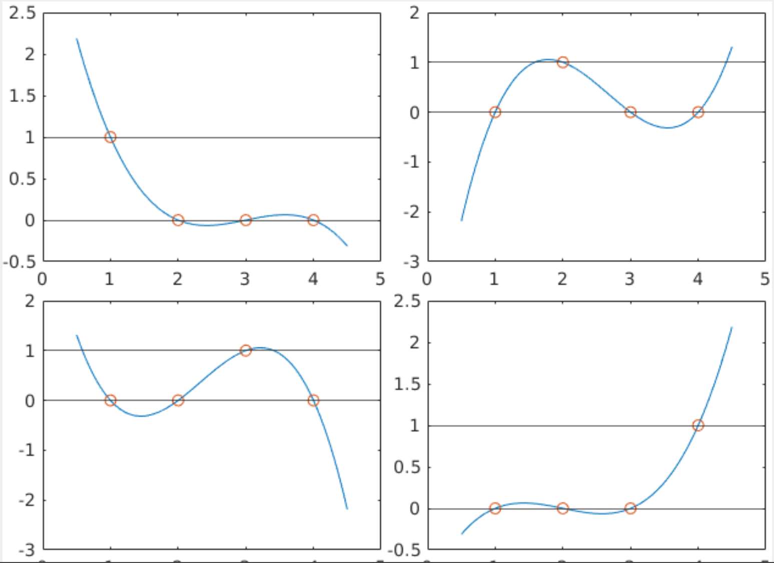
For a given set of distinct abscissas $\{x_i\}_{i=0}^n$,

$$\ell_i = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \qquad (0 \leq i \leq n) \tag{2}$$

are called **Lagrange basis** or **cardinal functions** associated to/subordinate to $\{x_i\}_{i=0}^n$.

- Π_n is a vector space.
- **Notation**: (**Kronecker delta**)
  $$\delta_{ij} = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases}$$
- 'abscissas' means horizontal coordinates $\{x_i\}_{i=0}^n$, .

Lagrange interpolation



- ℓ_0 (top left), ℓ_1 (top right), ℓ_2 (bottom left), ℓ_3 (bottom right)
- **Notation**: ℓ_i's will be reserved to be Lagrange basis functions from now on.

Lagrange interpolation

Theorem

If a set of functions $\{f_i(x)\}_{i=0}^n$ satisfies $f_i(x_j) = \delta_{ij}$, then it is linearly independent.

Proof.

Board work. □

Corollary

Lagrange basis is indeed a basis.

Proof.

Since $\dim(\Pi_n) = \#\{\ell_i(x)\}_{i=0}^n = n+1$, it suffices to show linear independence. Observe $\ell_i(x_j) = \prod_{k\neq i} \frac{x_j - x_k}{x_i - x_k} = 0$ if $j \neq i$ (one of the numerator is zero) and, if $j = i$, $\ell_i(x_i) = \prod_{k\neq i} \frac{x_i - x_k}{x_i - x_k} = 1$. Linear independence follows from the previous theorem. □

- $p(x) = \sum_{i=0}^n y_i \ell_i(x)$ means that just putting the data as coordinates (or coefficients) of the Lagrange basis, you have the interpolation.
- This result can be considered 3rd proof of the polynomial interpolation theorem.

Lagrange interpolation

Example

Following the previous proof, find a polynomial (of minimal degree) interpolating

x	0	1	2	3
y	1	1	2	3

Polynomial interpolation comparison

	Vandermonde	Lagrange	Newton
Basis	$1, x, x^2 \cdots$	$\ell_i = \prod_{j\neq i} \frac{x-x_j}{x_i-x_j}$	$1, (x-x_0), (x-x_0)(x-x_1), \cdots$
Theory	Some algebraic beauty	Convenient for **Lagrange interpolation** (i.e., only function values involved)	Effective for **Hermite interpolation** (i.e., also derivatives involved)
Numerical	Inaccurate and inefficient: the matrix is *ill-conditioned* and inverting a matrix is among expensive computations	Efficient when nodes are fixed but possibly the data to fit changes (Lagrange basis depends only on the nodes)	Efficient when nodes gets added (a newly added term does not affect the previous interpolations). Also, finding coefficients can be efficient when equipped with **divided difference**.[1]
Evaluation algorithm	Horner	Some algorithms exist	Horner

[1] Text in blue: next topics.

Polynomial interpolation error

Theorem

Let $x_0, x_1, \cdots, x_n \in [a, b]$ be distinct nodes, $f \in C^{n+1}[a, b]$, and $p \in \Pi_n$ interpolating f at the nodes. For each $x \in [a, b]$, there is $\xi_x \in (a, b)$ such that

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) \prod_{i=0}^{n} (x - x_i)$$

Proof.

Board work. $\square$

- This theorem is of great importance later on when we analyze numerical methods. There will be no waste of time appreciating detailed aspects of this theorem.

Polynomial interpolation error

Example

Find a bound on errors made by the polynomial interpolation of $f(x) = \sin(x)$ at 11 distinct nodes on $[0, 1]$. What if we require the nodes to be equally spaced?

- **Subjective question**: Can you conjecture how good interpolations are depending on the choice of nodes? Feel free to say what your guts tell you, then modify it if needed.

Runge's phenomenon

Question: For a very smooth function, say, $f \in C^\infty[-1,1]$, imagine what polynomial interpolations will be like if you use equally spaced nodes? What will happen as we increase the nodes?

Example (Runge's phenomenon)

Dynamic example of

$$\frac{1}{1+25x^2}$$

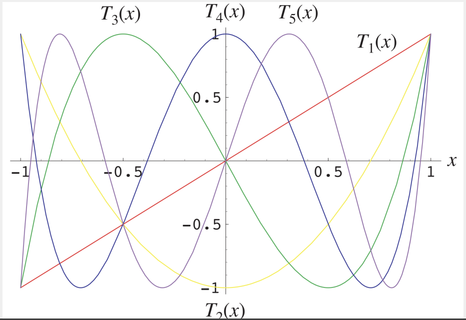
- Thank you, Cleve Moler, for this example.

Chebyshev polynomials

Motivation: Though we will not be able to discuss the full picture, some “best” interpolation is related to **Chebyshev polynomials**.

Definition (Chebyshev polynomials - 1st kind)

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad (n \geq 1)$$



- When we talk about Chebyshev polynomials, we are interested in the domain $[-1, 1]$ though they are defined everywhere.
- First few are:

$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 4x^3 - 3x$$
$$T_4(x) = 8x^4 - 8x^2 + 1$$

- We omit “1st kind” from now on.

Chebyshev polynomials

Theorem

For $x \in [-1, 1]$, we have

$$T_n(x) = \cos\left(n \cos^{-1} x\right) \quad (n \geq 0)$$

Proof.

Board work. □

Corollary

$$\begin{aligned} |T_n(x)| &\leq 1 && (-1 \leq x \leq 1) \\ T_n\left(\cos \frac{j\pi}{n}\right) &= (-1)^j && (0 \leq j \leq n) \\ T_n\left(\cos \frac{2j-1}{2n}\pi\right) &= 0 && (1 \leq j \leq n) \end{aligned}$$

Chebyshev polynomials

Theorem

Monic polynomials of degree n satisfy,

$$\|p\|_\infty = \max_{-1\le x\le 1} |p(x)| \ge 2^{1-n},$$

where equality holds with $p(x) = 2^{1-n} T_n(x)$.

Proof.

Board work. □

Theorem

If the nodes are zeros of T_{n+1}, then we have, for $|x| \le 1$,

$$|f(x) - p(x)| \le \frac{1}{2^n (n+1)!} \max_{|t|\le 1} \left| f^{(n+1)}(t) \right|$$

Proof.

Board work. □

Runge's phenomenon revisited

Question: We saw a bad interpolating result for the Runge's function. What if we use Chebyshev nodes?

Example (Runge's phenomenon)

Dynamic example of

$$\frac{1}{1+25x^2}$$

- As the name suggests, **Chebyshev nodes** are the ones consist of the zeros of Chebyshev polynomials.

Summary of polynomial interpolation

Here is some high level summary, which I believe is good enough for the very first course of numerical analysis.

- If a function is very well-behaving (like $\sin(x)$), reasonable polynomial interpolation (e.g., equally spaced ones) works well.
- Even if a function looks well-behaving (like $1/(1+25x^2)$), equally spaced nodes may not work. (To distinguish these two, we need to look through complex analysis lens.)
- If we choose a good set of nodes, the interpolation can be very satisfying (e.g., Runge's function with Chebyshev nodes)
- (**Weierstrass Approximation Theorem**) For any continuous function, we can find as good polynomial approximations as we please. (But it does not tell us how.) That is, let $f \in C[a,b]$, then, for $\forall \epsilon > 0$, there is a polynomial p such that $\|f - p\|_\infty < \epsilon$.

- As you have seen, polynomial interpolation is subtle and requires a deep dive for a better picture.
- Noticed $\frac{1}{1+25x^2}$ has singularities at $\pm\frac{\sqrt{-1}}{5}$. (We don't pursue this any further.)
- "Fix nodes first, then you can always find a bad function. Conversely, fix a function, then you can always find good nodes."

Polynomial Interpolation

Divided Differences

Newton form and divided differences

Setting: given a function f and nodes $x_0, x_1, \cdots, x_n$, find $p \in \Pi_n$ interpolating f, i.e., $p(x_i) = f(x_i)$, $(0 \le i \le n)$.

Example

Given nodes x_0, x_1, x_2 find the interpolating polynomial (of minimal degree) in Newton form. What does each coefficient depends on.

- **Feedback question**: (1) How many minutes do you think subjective questions are worth out 75 min? (0: worthless, 5: quite helpful) (2) Share the aspects/reasons you do/don't like subjective questions.
- Refresher: There is a unique interpolating polynomial given nodes and data. But the way it is written makes a huge (practical) difference.

Newton form and divided differences

Definition (Divided differences)

Given a function f and nodes $x_0, x_1, \cdots, x_n$, suppose $p(x) = \sum_{k=0}^{n} c_k q_k(x)$ is the polynomial interpolating f at the nodes in Newton form, where $q_k(x) = \prod_{j=0}^{k-1}(x - x_j)$, $(0 \le k \le n)$ is the basis of Newton form. Then, **divided differences** are defined to be the coefficients

$$f[x_0, x_1, \cdots, x_k] := c_k.$$

Corollary

Under the same assumptions as above,

$$\begin{aligned} p(x) &= \sum_{k=0}^{n} f[x_0, x_1, \cdots, x_k] q_k(x) \\ &= \sum_{k=0}^{n} f[x_0, x_1, \cdots, x_k] \prod_{j=0}^{k-1}(x - x_j). \end{aligned}$$

- **Convention**: $\sum_{k=0}^{-1} a_k = 0$ and $\prod_{k=0}^{-1} a_k = 1$. In words, "if a product or a sum does not make sense, assign it a value that has the same effect of doing nothing."
- The notation $f[x_0, x_1, \cdots, x_k]$ emphasizes it depend on f and the nodes only up to index k.

Newton form and divided differences

Theorem (Recursive relation of divided differences)

$$f\left[x_0, x_1, \ldots, x_n\right] = \frac{f\left[x_1, x_2, \ldots, x_n\right] - f\left[x_0, x_1, \ldots, x_{n-1}\right]}{x_n - x_0}$$

Proof.

Board work. □

Example

Use the above formula to find a polynomial (of minimal degree) interpolating

x	0	1	2	3
y	1	1	2	3

- Mnemonic device: (a) looks similar to finite difference approximation of derivatives (in fact, this is exactly true for $f[x_0, x_1]$), (b) numerator index – last n's minus first n's; denominator index – last one minus first one.

Newton form and divided differences

Algorithm for divided differences is very efficient.

Algorithm 1: Divided differences

```
for $i = 0$ to $n$ do
    $d_i \leftarrow f(x_i)$;
end
for $j = 0$ to $n$ do
    for $i = n$ to $j$ do
        $d_i \leftarrow (d_i - d_{i-1})/(x_i - x_{i-j})$;
    end
end
```

Then, the interpolating polynomial is

$$p(x) = \sum_{i=0}^{n} d_i \prod_{j=0}^{i-1} (x - x_j).$$

- See the textbook pp. 331-332 for details of algorithm. We focus on other properties and applications of divided differences.

Properties of divided differences

Theorem (Symmetry of divided differences)

If $(z_0, z_1, \cdots, z_n)$ is a permutation of $(x_0, x_1, \cdots, x_n)$, then

$$f[z_0, z_1, \cdots, z_n] = f[x_0, x_1, \cdots, x_n]$$

Proof.

Board work. $\square$

- Permutation means a shuffle.

Properties of divided differences

Theorem (Error of polynomial interpolation)

Let $x_0, x_1, \cdots, x_n \in [a, b]$ be distinct nodes and be $p \in \Pi_n$ interpolating f at the nodes. For each $t \in [a, b]$ different from the nodes, we have

$$f(t) - p(t) = f[x_0, x_1, \cdots, x_n, t] \prod_{i=0}^{n} (t - x_i)$$

Proof.

Board work. $\square$

- Permutation means a shuffle.

Properties of divided differences

Theorem (Discrete derivatives)

Let $x_0, x_1, \cdots, x_n \in [a, b]$ be distinct nodes. If $f \in C^n[a, b]$, there is $\xi \in (a, b)$ such that

$$f[x_0, x_1, \cdots, x_n] = \frac{1}{n!} f^{(n)}(\xi)$$

- Permutation means a shuffle.

For the rest of class

- HW release and deadline
  - ~~Friday~~: not ideal, too many holidays Tuesday 11:59PM seems the best if Thr OH moves to Tue.
  - HW8 will also be dropped from grades. Percentage of each HW increases to 35%/5HW = 7%/HW.
- What I intended with computational HW
  - More realistic HW than overly structured ones. (Do you remember the first day slides?)
  - Go for it! Remember you have safety nets.
  - Teachers are training wheels. My goal is to make you as more independent as each becomes.
  - Whatever you did, you are awesome! You have learned something.

(11AM) Thank you for your feedback!

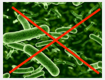
(2PM) Feedback question:

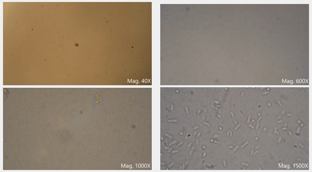
- How many minutes do you think subjective questions are worth out 75 min? (0: worthless, 5: quite helpful)
- Please, share the aspects/reasons you do/don't like subjective questions.

Some analogy and what we will learn

Goal

Tool

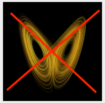




Interpret

Is it supposed to be this way? How can I tell this is what I am looking for?

$$\begin{cases} x_t = y - x, \\ y_t = -xz + \frac{1}{9}x - y, \\ z_t = xy - 2z, \\ x(0) = 1, y(0) = 1, z(0) = 1 \end{cases}$$



(x(t), y(t), z(t))=(NaN, NaN, NaN)

(x(t), y(t), z(t))=(0,0,0)

(x(t), y(t), z(t))=(0.893,1.26,2.22)

(x(t), y(t), z(t))=(1.2e10,2.5e-7, 5.332)

Which one is correct?
How can I trust the result?

- What tools are available for a specific problem? $\rightarrow$ Methods (Tool; microscopes)
- How reliable are they? Convergence, order of accuracy, etc. (Tool; knowledge on microscopes)
- What to be careful of? Wisdom, general knowledge, etc. (Interpret)
- Different aspects are waiting: abstract, beautiful, artistic, etc.

Learn from peers: friend 1

This homework is really **challenging** for me because I've **never used the "numpy"** package before and my python skill is quite weak. But **after achieving the goal, I find the result is not that complicated**. Basically, **it's just breaking the big goal into several steps** and find the corresponding **code to make each step** work. At the beginning, I'm a little stuck by the problem because it seems really complicated. Following the instruction, I break the problem into pieces like "define Newton method function", "generating matrix", "use loop to apply function to each value in the matrix", and "plotting the matrix". **The biggest struggle I've had was tons of functions that I've never seen before**. It's useful to seek some answers from **google** and asking **professor** for help.

I can find most of the functions from google and I also try to use the knowledge I've learned from CS8 to create for loops to make the code easier to read. The whole process is kind of interesting. It's **helpful to think more about the logic** of the code. Sometimes we can find a **better/easier way to achieve some goal** (I stuck at creating matrix for a while because I want to find some "easy" code to directly generate the matrix I need. After wasting a long time finding on google, in the end, I made it by combining "for loop" and "linspace"(the function to generate a sequence). These two functions are easy to use.). I think we just need to be patient and curious to the process of solving the question. It's quite fun actually.

Learn from peers: Friend 2

"I **really struggled with this assignment at first** because, though I find the math aspect easy, I **have not coded** something in a couple of **years**, and never something similar to what the assignment asked for. I did not know how to make a graph or a grid, how to color different sections – pretty much, I knew how to assign a color to a value but then I **didn't know** how to assign that color onto a **chart/graph/physical image**.
I tried to use the notes in the Tips For Plotting of the assignment document but I did not understand how to define CC and make it look like what I wanted. I also **hate using lines of code that I do not fully understand**, even if they help me achieve my goal. So I tried to find another way.
My next attempt, I tried to make a stacked bar graph so I could have a graph with numbers and I could fill in segments with color. But since there were parts where red, for example, was both in the positive and the negative, **this did not work out**.
I **looked up** an article on how to make different kinds of graphs on Python and I found that you could change the shape of the points of a scatter plot. I then looked up if you could change the size of the points of a **scatter plot** and, once learning I could, my idea to plot all the points as different colored squares of a scatter plot, blown up in size, **was born**. Once I had this idea, coding it out turned out to be fairly easy.
The funny thing is, I **eventually nixed this idea** because I made my step much smaller to make my graph more detailed. However, I **managed to stick with the scatter plot idea**.

Learn from peers: friend 2 (continued)

I also had an **issue with making my graph square**. When looking up how to do this online, I didn't understand the lines of code people were using and, again, I refused to use a single line of code I myself didn't understand. So I fiddled with the ideas I saw in how to make it square and **eventually figured it out**.
Lastly, I struggled to make a **legend** for my graph. First, this was because the legend for every one of my graphs would be different since my code was set up for four **different cases for n**. However, this was easy to get around because I just set up an if-then situation to label them. My **main problem** with it was because the **size I made my scatter plot points was a terrible size for a key**.

I **tried many ways** to get around this, including making a second plot behind my first one, but this did not look good. I was about to scrap it but then I found that you are able to change the plot marker size with a very easy command when setting up the legend.
All in all, I really enjoyed his assignment. Mostly, I had **trouble getting started** but once I did it was a breeze. The struggles I faced afterwards were more aesthetic than anything but I really wanted to figure it all out and so I did."

Learn from peers: friend 3

”This assignment was very interesting and **definitely something I did not expect myself to be able to do**. Essentially, we were to grab points on a grid and run them through Newton’s Method to check if they converge to a root. If so, we would color it corresponding to its root. **Once I had this idea** in my head, I **slowly began to make progress**. However, as I complete one part of the code, I found myself having **a new issue to fix**. There were also times when I wanted to code a function, but end up not know how I would approach it. It resulted it nested loops after nested loops and even I was unsure what the results were.

What really helped me complete this assignment was taking a step back. By hand, I planned out how I wanted each list, matrix, and function. Moreoever, I had a general idea of what lists/matrices to iterate through in order to to get my functions up and running. **With a help from others**, I discovered that placing a for loop inside of a list is able to iterate and create the desired list. This definitely helped make my code more concise and easier for not only me, but also for the reader to understand.”

Learn from peers: friend 4

"To be honest, when I heard there was going to be a programming project **I was scared**. I've only taken intro python two summers ago, so it was only 6 weeks and I knew that would make me a little under-prepared and overwhelmed. However, considering I knew what I wanted to prove for Newton's function, it was only the matter of putting it into code. My first section of code was so that I could take a function, its derivative, a guess, and the number of iterations, and have an output of where the root would be after n many iterations. That was the easiest part for me. **The second part was a lot more of a struggle**. I wanted to be able to graph the function, show the points of iteration, and also color the segments after each iteration where it got closer and closer. I had to **google a lot** and I still couldn't exactly figure it out. But I managed to find **how to create code that gave me a graph including the x and y axes**. Then it would graph the function for me. **I'm dissapointed I wasn't able to plot the points of iteration on the same graph, but I did not have enough time to work it out. If I could change what I did**, I would have put everything into one body of code, so the part that determines the roots and the part that creates the graph are together. Another thing is that I would change how my code takes input. The first part can take input by calling the function, but the part that creates a graph requires me to put in the function manually in the definition. I'm not sure how I would change that but I think **if I had more time** I could have solved it. Overall, it was a very **interesting project that made me think a lot more and utilize outside material**."

Learn from peers: friend 5

"**I am very disappointed with the way that this assignment went for me.** I feel like I really spent as much time on it as I could have, and tried various ways of doing it, but it seems that **none of the images that I generated were good enough or up to my standards. I feel like I truly failed given all of the work I put into this**.
With that being said, **I do feel like I learned a lot** on this assignment. I learned a lot about coding and I believe it got me much more familiar with Newton's method. I guess that is some sort of silver lining. I hope that I can understand things better going forward."