Files in Bash on Collatz Sequences

Moore Macauley
University of California, Santa Cruz

October 1, 2022

Abstract

The Collatz sequence, or rather, the Collatz conjecture, is a mathematical sequence starting with the natural number n. If n is even, then the next value in the sequence is equal to $n/2$. Otherwise, it is equal to $1+3n$. This process then repeats, with the calculated value taking the place of n. The conjecture states that no matter what the inital value n is, the sequence will always end with the last term being 1. Given a program that would calculate the collatz sequence for a given number, I was tasked to create a series of four graphs comparing the various sequence lengths and maximum values in sequences for numbers between 2 and 10000.

1 Introduction

In assignment 1, I created a series of three graphs based on the collatz sequence of numbers between 2 and 10000 by writing a bash script. To begin with, I used the make command to compile the provided collatz.c. This command accesses the similarly provided Makefile and uses it to compile collatz.c into a runnable form, which will be helpful later. I then used a bash for loop to iterate over all the whole numbers from 2 to 10000, as I would need to create a collatz sequence for all of those numbers, so a for loop iterating over them would be the easiest way to do that. The aforementioned for loop was used to help create all of the graphs. Similarly, I used the now compiled ./collatz -n $i to run the now compiled collatz.c with the qualifier -n to create a collatz sequence for the current value of the iterating variable. Next, I used > to redirect the standard output of ./collatz to a file while also overwriting the collatz sequence of the previous number, which is used in the creation of all of the graphs

2 Figure 1

To create the first graph, seen below and similar to figure 2 in the assignment pdf, I primarily used echo, wc, tail, and redirections inside the for loop. For gnuplot to create this graph successfully, I needed to build a file that had “x y” and a newline for every point I wanted to produce, as that is the format required for dot plots. As this graph has the starting number of a collatz sequence plotted against the length of that sequence, this takes the form of "(iterating number) (sequence length)" followed by a newline. Echo was used to include the number and the whitespace, as could call echo -n “$i “ to print the current state of the iterating number and a whitespace. As the argument -n also removes the trailing newline that typically follows an echo call, this single command creates the first half of the necessary line. Therefore, I used » to write the output to file length_data.dat, which appends it onto the end of length_data.dat without overwriting it, allowing the file to be created slowly over multiple iterations of the loop. I then used wc -l on the

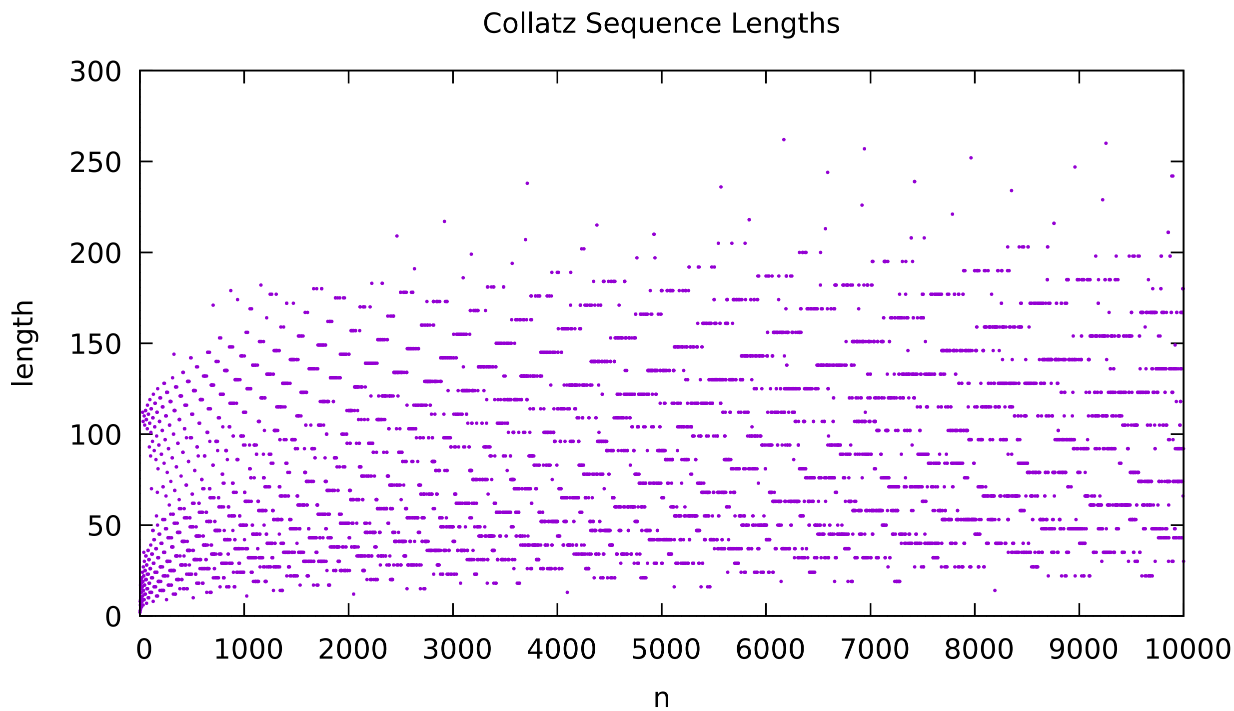

Figure 1: Interesting pattern reminiscent of root functions

file containing the collatz sequence, which counts the number of lines and nothing else, due to the argument -l. Due to how ./collatz outputs, each number in a sequence occupies a single line, so counting the lines doubles as counting the length of the sequence. As such, wc was simply the easiest way of determining the sequence's length. While I could redirect the outputs into length_data.dat, I need the length data without the number and whitespace for another graph, so I redirect it into the file raw_length_data.dat with » instead. However, I still need to add the collatz sequence length to length_data.dat, which now resides in the last line of raw_length_data.dat. So, to fetch this, I run the command designed to get the last lines of files, tail, on raw_length_data.dat. I apply the additional arguments -n to specify that tail should retrieve lines, and -1 so it only fetches the very last line, and append that to length_data.dat. With both the iterating value and the sequence length included in length_data.dat for every iteration of the for loop, this section of code will create a file that can be graphed by gnuplot at the end of the loop.

3 Figure 2

My strategy for the second graph, a graph of the maximum values of collatz sequences, is very similar to the first, although I used sort instead of wc. I needed a file in the same format, which in this case would be "(number) (maximum value)" followed by a newine, so I used echo to add the iterating value just like in graph 1. To acquire the maximum value, I used sort -n on the collatz sequence, which sorted it numerically (thanks to the -n argument) from smallest to largest, and redirected the result into ordered_sequence.dat. As the file is sorted, getting the maximum value of the sequence is as simple as getting the last line of ordered_sequence.dat, making sort the easiest way of isolating this value. As such,

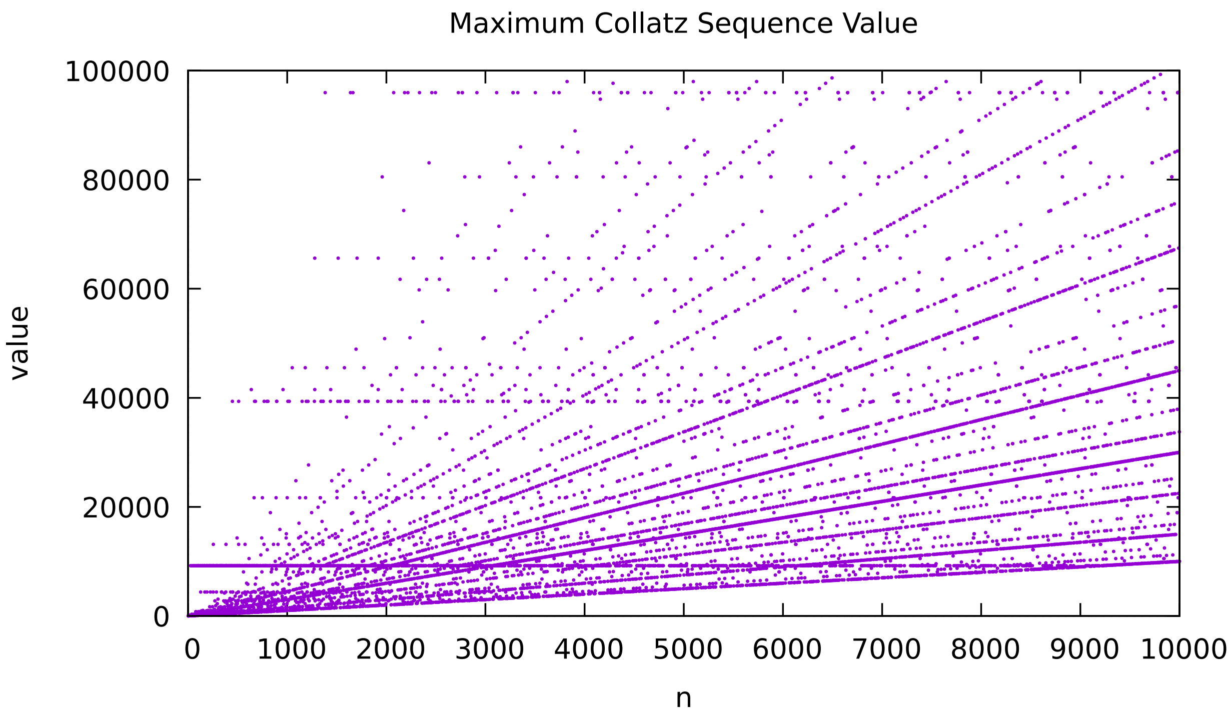

Figure 2: Almost seems to be a series of several linear funtions

I used tail the same way as I did in the first graph to get the last line of raw_length_data.dat, and redirected it into value_data.dat. Altogether, much the same as graph 1, this code constructs a line containing both the iterating value and the maximum value in the collatz sequence created with the iterative. Therefore, the file created by this code can be used by gnuplot to produce the above graph.

4 Figure 3

Graph 3 is my different graph, which consists of the length of a collatz sequence starting with a number plotted against the maximum value of that collatz sequence. I was interested to see how strong a correlation length and maximum value had, inspiring me to create this graph. As can be seen from the diagram, while the correlation is not strong, there are very distinct horizontal lines, which I found intriguing and impossible to explain.

Same as the previous graphs, gnuplot needs a line in the format "(sequence length) (max value)" for every point. Thankfully, these values already exist and have been saved while crafting the first two graphs. To begin with, the sequence length for the current iterative exists as the last line in raw_length_data.dat, so I used tail the same way as I used it in graphs 1 and 2 to retrieve it. Unfortunately, this line contains a newline character at the end, which must be removed first to maintain the required format. As I don't want to have to save this line somewhere, I use a pipe to move the output of the tail command into the input of head -c -1. Head typically outputs the first 10 lines of the input, but due to the argument -c and a number, it will instead output the first provided number of bytes. However, as the provided number is negative, it will output all of the input except for the last number of bytes. By run-

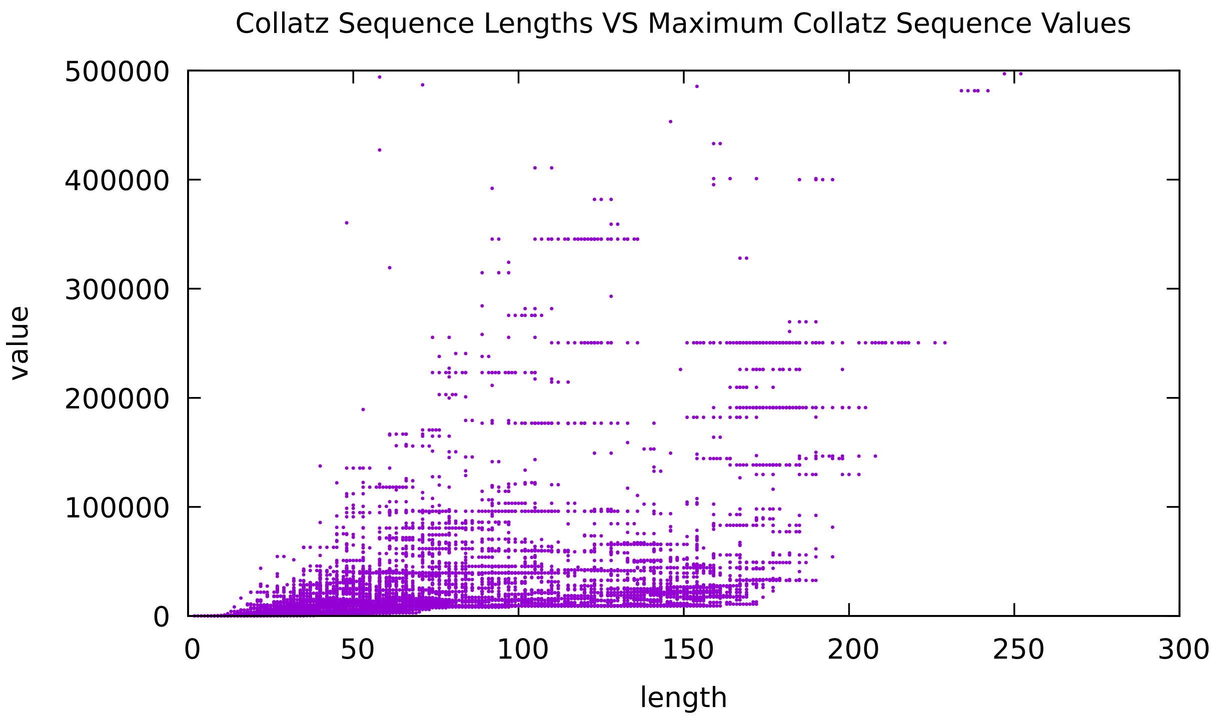

Figure 3: Odd horizontal line pattern everywhere

ning this command, head will output all except for the last byte in the input, which in this case is the newline character we want to trim, the easiest and simplest way of removing this character. Therefore, we can redirect the output of head with » to comparison_data.dat. We then use echo -n " " » comparison_data.dat to add the whitespace, before using tail to get the last line of ordered_sequence.dat, which, similar to raw_length_data.dat, is the maximum value in the collatz sequence. However, unlike before, as we now want the newline, we redirect this to comparison_data.dat without any changes. With that done, this section should create a line in the proper format for every iteration of the for loop, allowing gnuplot to produce the graph seen above.

5 Figure 4

5.1 Setup

Lastly, we have graph 4, the histogram of collatz sequence lengths. For gnuplot to create this graph, I needed to create a folder that contained the number of occurrences on the corresponding line number, i.e., line 0 holds the number of times a length of 0 occurred, line 1 holds the number of times length 1 occurred, and so on. To do this, I first needed to do quite a bit of setup on some of the previously discussed data created in the initial 2 through 10000 for loop. Firstly, I used sort -n to numerically sort raw_length_data.dat, and redirected the result to sorted_length_data.dat, as the command uniq needs a sorted file to work correctly. After that, I used both uniq -u and uniq -d on sorted_length_data.dat and redirected both outputs to a new, temporary file. uniq -u outputs all of the unique lines in the file it is used on, while uniq -d outputs all of the repeated lines in the file once. By combining these outputs, I created a list of the sequence lengths in my data set, before sorting the temporary file with sort -n and redirecting the result to lengths_that_appear.dat, to order them from least to greatest. From here, I saved the last value in lengths_that_appear.dat to the variable max_value using tail -n 1 the same way I did in the previous graphs, remembering that due to length_that_appear.dat being in order, this will be the largest length that appears. Finally, I used uniq -c on sorted_length_data.dat, which adds the number of times a line occurred in a row in the first 7 bytes before the first occurrence of that line and removes all other instances. As sorted_length_data.dat was sorted, all of the occurrences of a line will be in a row, causing all appearances of different lengths to be counted. This will create a file that lists the number of times every value in lengths_that_appear.dat occurs, in the same order.

5.2 Theory

With the setup complete, it is time to create the data. To do this, we create a for loop that iterates over all the numbers between 0 and the saved max_value. As the loop iterates, we will save a value in the line for every iteration of the loop, even if it is not a length that appears in the data set. If a value between 0 and the max value is not a length that appears in the data set and is left out, all the following values will be in the wrong place. However, we first check to see if the value in the first line of lengths_that_appear.dat, which due to the earlier sorting will be the smallest length that appears, is equal to the iterating value. If it is, this means that the sequence length did appear at least once, and therefore the corresponding number of occurrences is stored in the first 7 bits of occurrences.dat. We then retrieve that value, and add it in its own line to historgram_data.dat, creating the necessary line in historgram_data.dat to plot the bar corresponding to the iterating value. We then remove the first line from occurrences.dat and lengths_that_appear.dat, to load up the next largest appearing length to be tested against the iterating value. If, however, the iterating value and the current first line of lengths_that_appear.dat are not equal, this simply means that the length never appeared, and there was a gap between the smallest length that arose and the next largest length that appeared. Therefore, we add 0 and a newline to historgram_data.dat to plot this fact. Accordingly, this will create a file that gnuplot can use to produce the below graph.

5.3 Specific Command Details

The commands used to create this file are a little more complex. To retrieve the first line in lengths_that_appear.dat, we use head -n 1, then pipe that into head -c -1 to trim off the newline character as discussed previously. This is tested against the iterating variable with [[]] and an equals sign in an if statement. To get the number of appearances from occurrences.dat, we retrieve the first 7 bytes of occurrences.dat using head -c 7, containing the number of occurrences and several whitespaces which are ignored by gnuplot. After adding the number of occurrences to historgram_data.dat, we use echo “”, which only outputs a newline character, creating the necessary line for the current iterating value. To remove the first line from lengths_that_appear.dat, we use tail -n +2 on it. While tail -n x gets the last x lines from a file, tail -n +x outputs all lines of the file starting at line x, notably including x. By using tail -n +2, we output all but the first line of lengths_that_appear.dat, which gets redirected to a temporary file. We then redirect cat temp.dat (which simply outputs everything in temp.dat) back into lengths_that_appear.dat. This temporary file is unfortunately necessary, as attempting to directly redirect the output of tail -n +2 to the same file results in the file being erased. The result is that we remove the first line from lengths_that_appear.dat. To keep the files synced up, we do the same thing to occurrences.dat. When

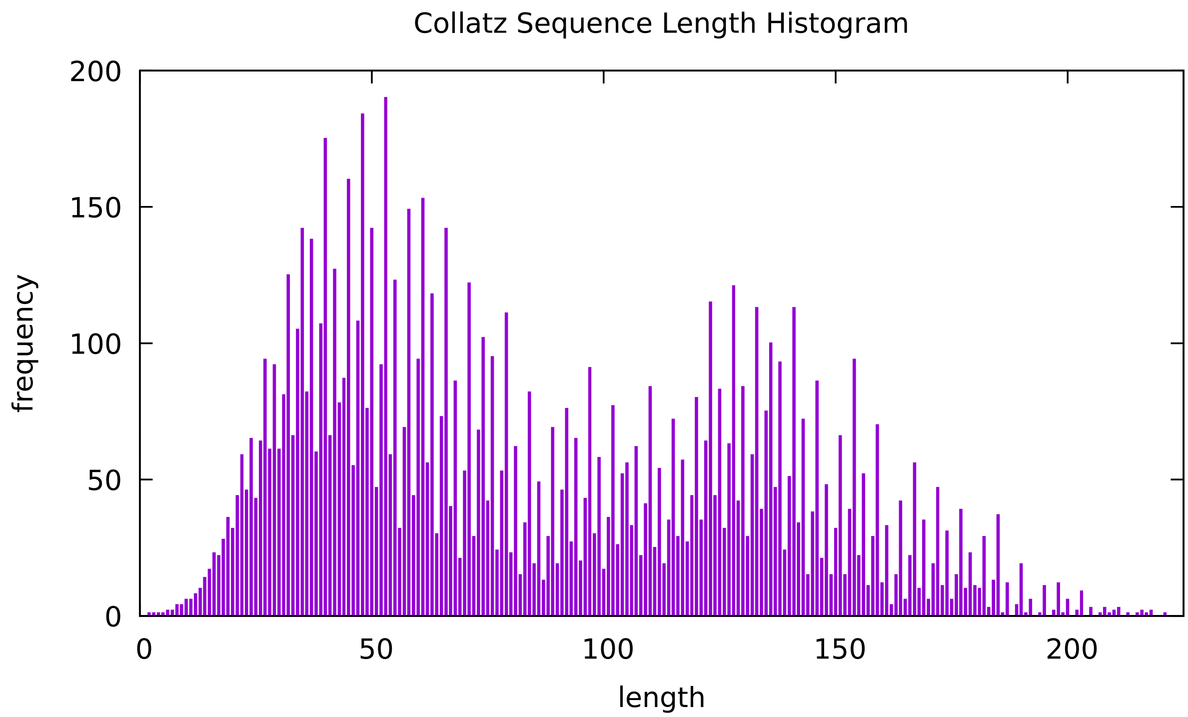

Figure 4: Seems to have a value that repeats often roughly every other length

the iterating value does not occur in lengths_that_appear.dat, things are much more simple, as we use echo 0 » historgram_data.dat to add a 0 and a newline at the corresponding line.

6 Notes on Graphing with Gnuplot

Now that all of the data has been created, the last step is to direct all of it into gnuplot itself. To accomplish this, I decided to create a heredoc. This section of code acts as a large bloc, which can all be redirected into gnuplot as a whole, essentially being a separate script for the creation of the graphs by gnuplot inside of plot.sh. Inside the heredoc all of the commands used are fairly simple. I use the set command to set various aspects of the graph, from the label of the x axis to the domain and range, all of which are fairly self explanatory. After setting the necessary parameters, I then simply call plot "(respective data file.dat)" with (graph type) title "", which tells gnuplot to graph the collected data as the appropriate graph type, without the title that appears in the top right. With that, the script has created all four necessary graphs, and all that is left to do is remove all the files created during runtime. The program expects the files to be empty at the start of the script, so by removing them we ensure that they always are

A plot.sh

```
make collatz

#iterates over all the numbers that will be plotted to create the data that will
#be used in the graphs

for i in {2..10000};
do
    ./collatz -n $i > sequence.dat;        #Creates collatz sequence for number i

    #Creates the data on collatz sequence lengths in the format of "(starthing number i)
    #(sequence length)" to length_data, and saves all sequence lengths in order to
    #raw_length_data

    echo -n "$i " >> length_data.dat
    cat sequence.dat | wc -l >> raw_length_data.dat;
    tail raw_length_data.dat -n 1 >> length_data.dat;

    #Creates data on maximum collatz sequence values in the format "(starting number i)
    #(max value)" to value_data

    echo -n "$i " >> value_data.dat ;
    sort -n sequence.dat > ordered_sequence.dat;
    tail ordered_sequence.dat -n 1 >> value_data.dat;

    #Creates data on maximum collatz values and collatz sequence lengths in the format
    #"(length) (maximum value)"

    tail raw_length_data.dat -n 1 | head -c -1 >> comparison_data.dat;
    echo -n " " >> comparison_data.dat
    tail ordered_sequence.dat -n 1 >> comparison_data.dat;

done

#Produces a list of all the lengths that appear ordered from least to greatest,
#the maximum length that appears, and a list of the number of times all lengths
#occur, also ordered from least to greatest

sort -n raw_length_data.dat > sorted_length_data.dat;
uniq -u sorted_length_data.dat > temp.dat;
uniq -d sorted_length_data.dat >> temp.dat
sort -n temp.dat > lengths_that_appear.dat;
#I googled how to save the output of a command to a variable, specifically from the first
#few paragraphs of https://linuxhint.com/bash_command_output_variable/
```

```
max_value=$(tail -n 1 lengths_that_appear.dat);
uniq -c sorted_length_data.dat > ocurrences.dat;

#Iterates from 0 to the maximum sequence length

for (( i=0; i <= $max_value; i++ ))
do
    #We take the next largest sequence length that apears

    #Same note as last time this appeared
    current_value=$(head -n 1 lengths_that_appear.dat | head -c -1)
    if [[ $current_value = $i ]]
    then
        #If it is equal to the current iterative, then that length does appear,
        #so we get the number of occurences from our list of the number of occurences for
        #all lengths, before removing that sequence length and the number of occurences
        #from their respective lists

        head -c 7 ocurrences.dat >> historgram_data.dat
        echo "" >> historgram_data.dat
        tail -n +2 lengths_that_appear.dat > temp.dat
        cat temp.dat > lengths_that_appear.dat
        tail -n +2 ocurrences.dat > temp.dat
        cat temp.dat > ocurrences.dat
    else
        #If this is not equal, then the length does not appear, so we add a 0 to the
        #data to be plotted

        echo 0 >> historgram_data.dat
    fi
done

gnuplot <<END
    #plots length data into figure 1.pdf

    set terminal pdf
    set output "figure 1.pdf"
    set title "Collatz Sequence Lengths"
    set xlabel "n"
    set ylabel "length"
    plot "length_data.dat" with dots title ""

    #plots comparison data into figure 3.pdf

    set output "figure 3.pdf"
```

```
    set title "Collatz Sequence Lengths VS Maximum Collatz Sequence Values"
    set xlabel "length"
    set ylabel "value"
    set yrange [0:500000]
    plot "comparison_data.dat" with dots title ""

    #plots value data into figure 2.pdf

    set output "figure 2.pdf"
    set title "Maximum Collatz Sequence Value"
    set xlabel "n"
    set ylabel "value"
    set yrange [0:100000]
    plot "value_data.dat" with dots title ""

    #plots histogram data into figure 4.pdf

    set output "figure 4.pdf"
    set title "Collatz Sequence Length Histogram"
    set xlabel "length"
    set ylabel "frequency"
    set xrange [0:225]
    set yrange [0:200]
    plot "historgram_data.dat" with histogram title ""

    quit
END

#Remove all txt files created during runtime, because some of the files don't get
#overwritten, just added on to, so having them around would break things, and clutter is bad

rm raw_length_data.dat
rm comparison_data.dat
rm sequence.dat
rm ordered_sequence.dat
rm value_data.dat
rm temp.dat
rm lengths_that_appear.dat
rm ocurrences.dat
rm historgram_data.dat
rm length_data.dat
rm sorted_length_data.dat
```