Taylor Series and Floating Point Error

Morgan Monzingo

9.16.15

Math 3315

Part A:

In this project, we are using c++ to approximate a function by taylor polynomials, and then calculate the error and then we also find the error in a backward finite difference approximation. All algorithms and equations used came from the project guide and the textbook.

I began by initializing a counter matrix that increments by 0.01 on the interval -4 to 4. I named this matrix z and it will be the independent variable throughout part a. Then I initialize every future matrix and their sizes are all the same as z. Then by hand I solved for the coefficients for a 5th, 9th and 13th degree taylor polynomial of sin(x). In a four loop I calculate the 5th, 9th and 13th degree taylor polynomial using the nest algorithm function.

The nest function accepts two variables, a matrix and the independent "x" variable. I use a for loop to solve for the matrix a with my x value acting as the independent variable in the matrix. To solve the for loop acts as Horner's method, factoring out the x values and incrementing through the taylor polynomial coefficients and multiplying it by the value the previous for loop has already calculated "p". This method is solved from right to left, starting with the highest coefficient and working to the first coefficient.

When solving each p5, p9 and p13, the main for loop goes through every value of z so in the final p matrices they each have 801 solutions. I also solve for the error of each of the sin(x) polynomials. To do this I create an f matrix that holds the 801 values of sin(z). Then I find err5 which is the absolute value of sin(z) minus p5(z) , both values of z must be the same. I find the error from the 9th and 13th degree polynomial the same way. To finish the program I write each of my final matrices into their own file.

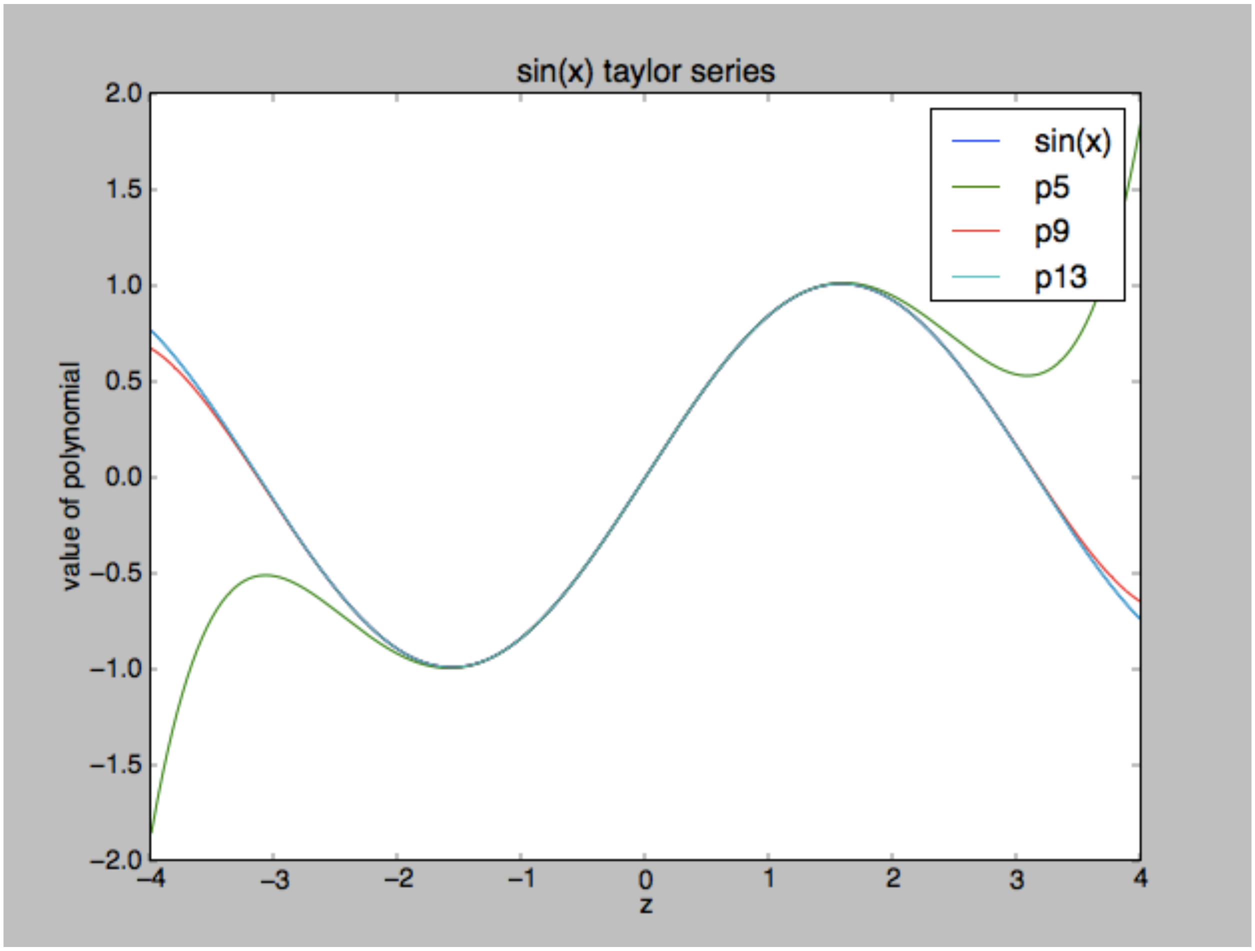
My python file for part a uploads every file created in the c++ into a matrix. The program then plots the sin(z) function, and the three taylor series vs z in a normal graph. In the second graph it's a semi-log plot that has all three error matrices vs the z values.

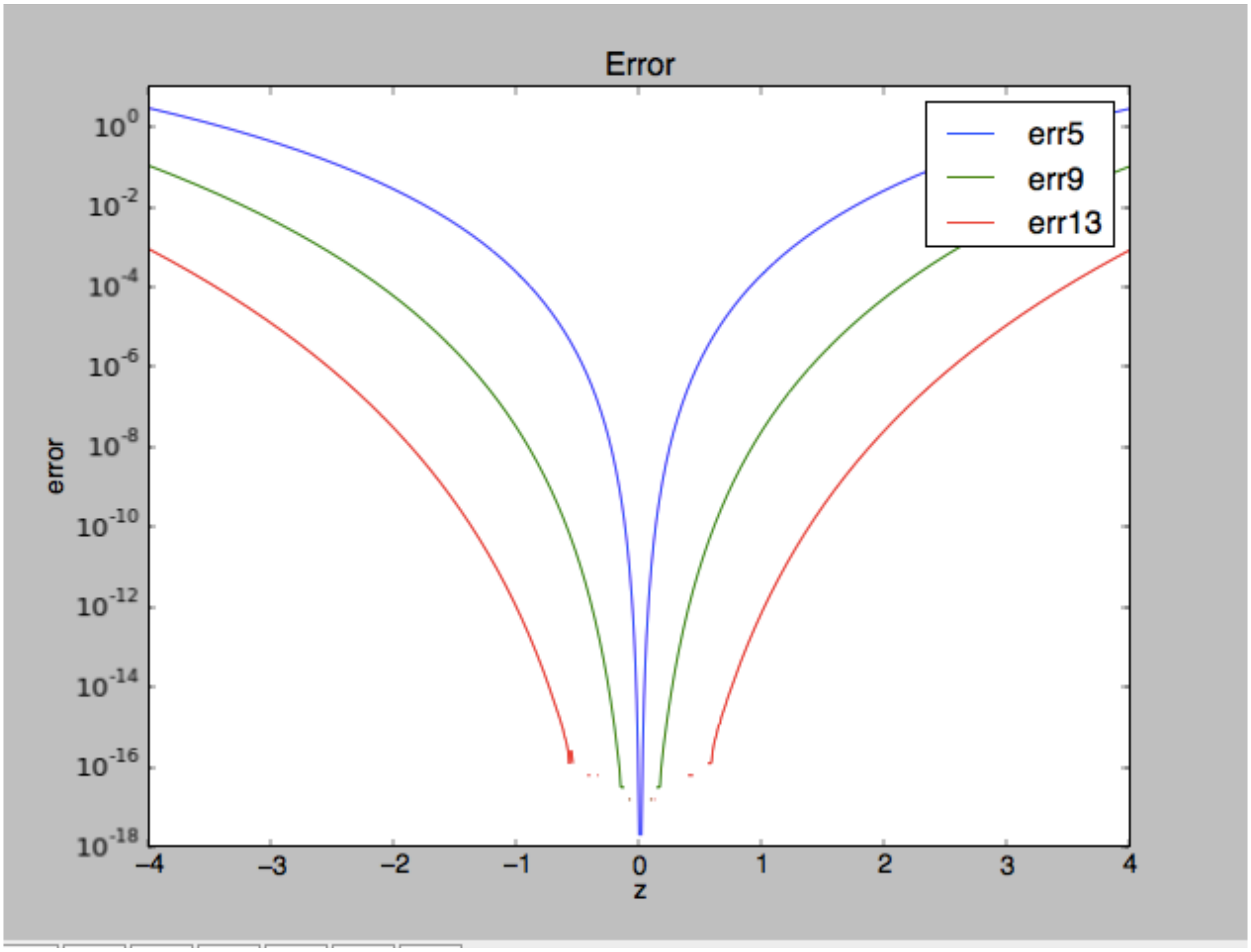
Part B:

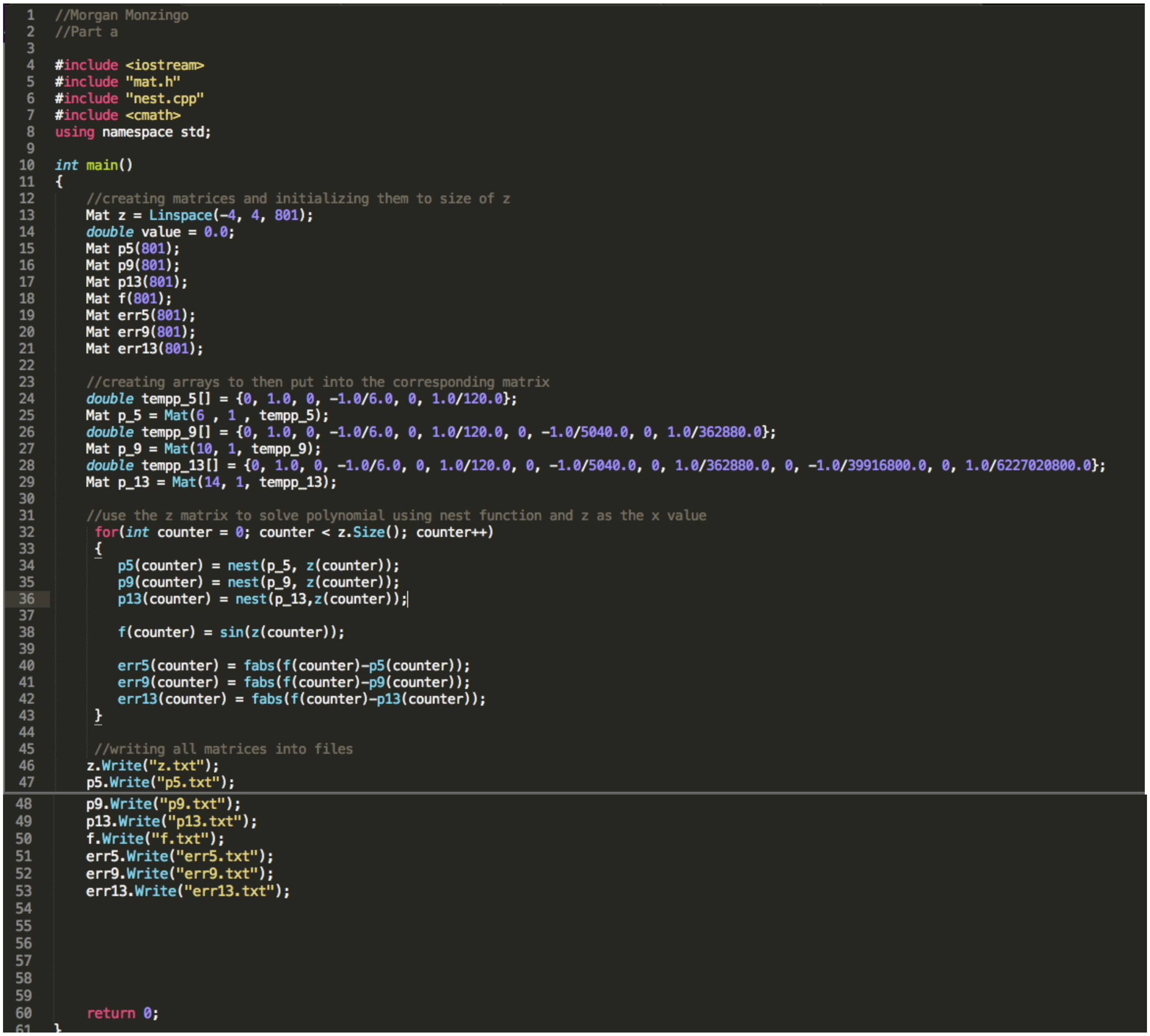
This part begins with one c++ program and then also uses python to plot the results. I began with creating a counter, 'n', that goes from 1 to 52 incrementing by 1. I then solve my f(x), f'(x) and f''(x) using the given value of 5 for the x. I was also given f(x) to be x^-2. I also solve for the future c1 and c2 values. The equations were given in the project file, they also use the recently solved for f, f', and f''.

In the first for loop I am solving for h which is going to increment through the equations. This equation used to solve for h is given in the assignment, it is 2^-n, n being the values 1-52. I next put my h array into a matrix. In the next for loop I am solving for another piece of the final equations. This equation finds the value of the approximation.

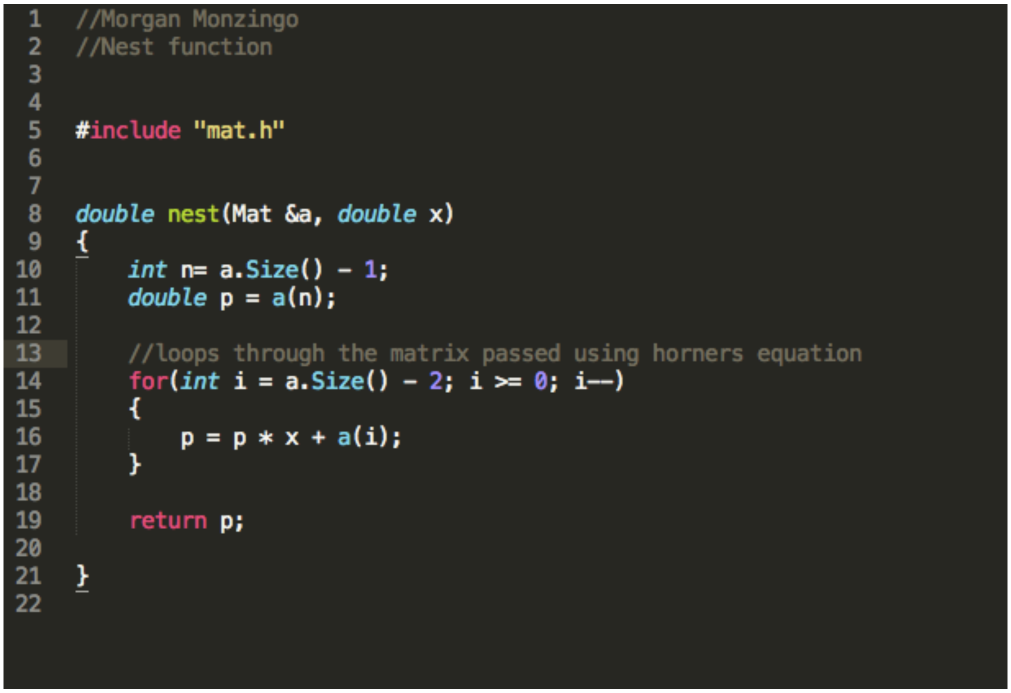
Then I solved for the relative error and the upper bound on the relative error. this is the error in approximating the f'(x). Each equation used is from the project information, all values were calculated in the beginning of the c++ program. I write the counter and error variables to text files and upload them into python. I make two graphs, the error vs n and the error vs h. The first graph is a semi-log graph and the second is a log-log graph.


sin(x) taylor series
value of polynomial
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−4
−3
−2
−1
0
1
2
3
4
z
sin(x)
p5
p9
p13

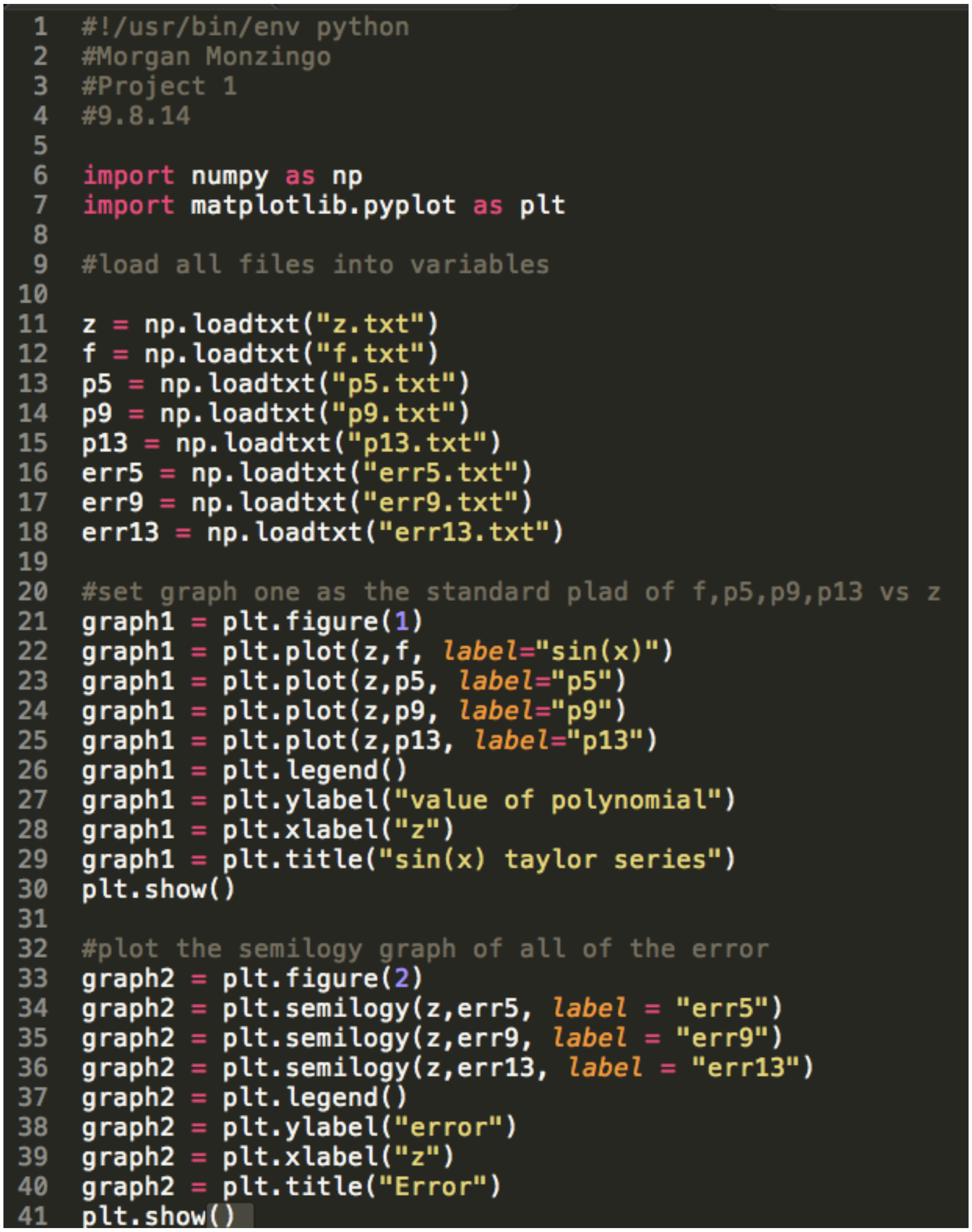

Error
error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{-18}
−4
−3
−2
−1
0
1
2
3
4
z
err5
err9
err13



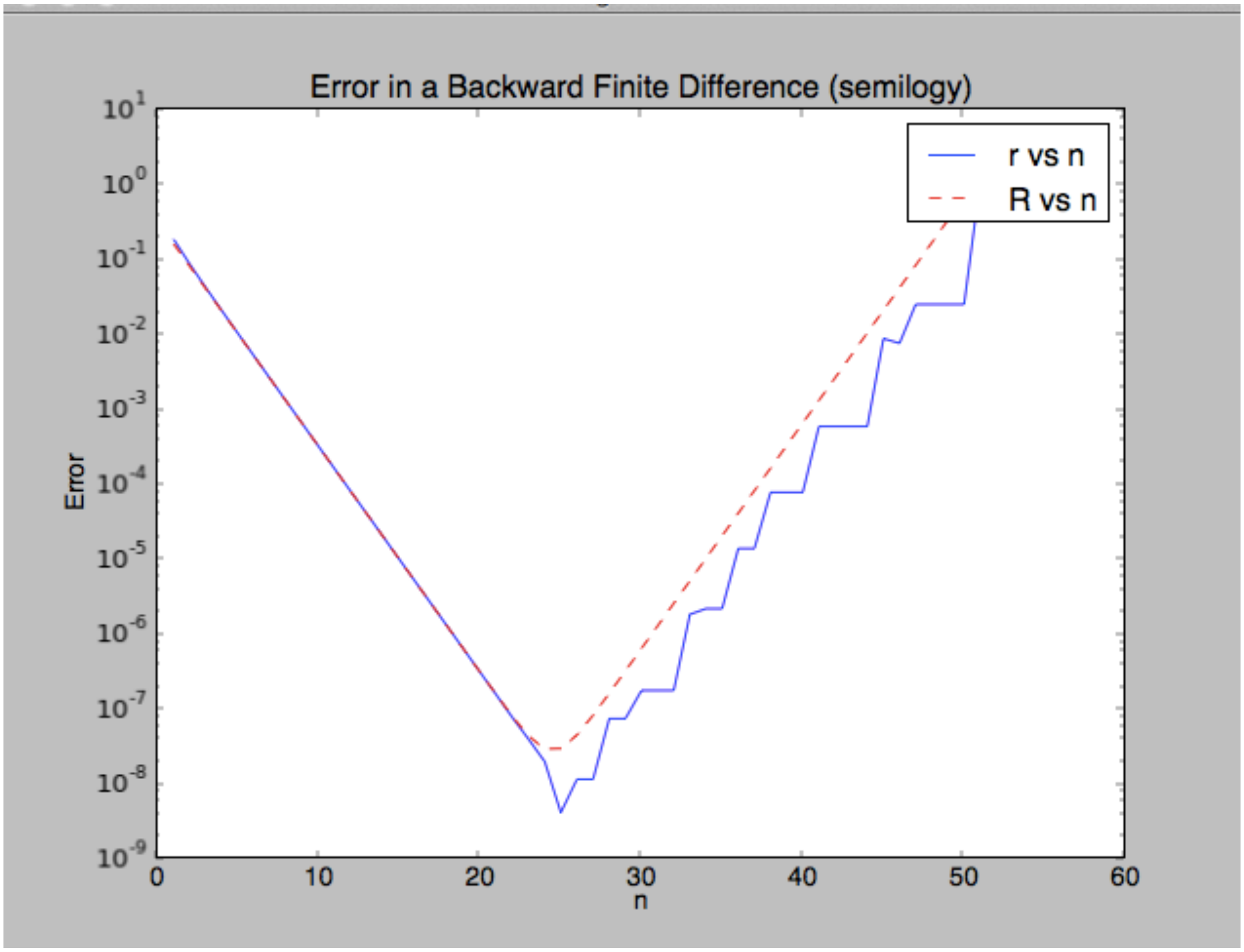
```
//Morgan Monzingo
//Part a

#include <iostream>
#include "mat.h"
#include "nest.cpp"
#include <cmath>
using namespace std;

int main()
{
    //creating matrices and initializing them to size of z
    Mat z = Linspace(-4, 4, 801);
    double value = 0.0;
    Mat p5(801);
    Mat p9(801);
    Mat p13(801);
    Mat f(801);
    Mat err5(801);
    Mat err9(801);
    Mat err13(801);

    //creating arrays to then put into the corresponding matrix
    double tempp_5[] = {0, 1.0, 0, -1.0/6.0, 0, 1.0/120.0};
    Mat p_5 = Mat(6 , 1 , tempp_5);
    double tempp_9[] = {0, 1.0, 0, -1.0/6.0, 0, 1.0/120.0, 0, -1.0/5040.0, 0, 1.0/362880.0};
    Mat p_9 = Mat(10, 1, tempp_9);
    double tempp_13[] = {0, 1.0, 0, -1.0/6.0, 0, 1.0/120.0, 0, -1.0/5040.0, 0, 1.0/362880.0, 0, -1.0/39916800.0, 0, 1.0/6227020800.0};
    Mat p_13 = Mat(14, 1, tempp_13);

    //use the z matrix to solve polynomial using nest function and z as the x value
     for(int counter = 0; counter < z.Size(); counter++)
     {
        p5(counter) = nest(p_5, z(counter));
        p9(counter) = nest(p_9, z(counter));
        p13(counter) = nest(p_13,z(counter));

        f(counter) = sin(z(counter));

        err5(counter) = fabs(f(counter)-p5(counter));
        err9(counter) = fabs(f(counter)-p9(counter));
        err13(counter) = fabs(f(counter)-p13(counter));
     }

     //writing all matrices into files
    z.Write("z.txt");
    p5.Write("p5.txt");
    p9.Write("p9.txt");
    p13.Write("p13.txt");
    f.Write("f.txt");
    err5.Write("err5.txt");
    err9.Write("err9.txt");
    err13.Write("err13.txt");






    return 0;
}
```

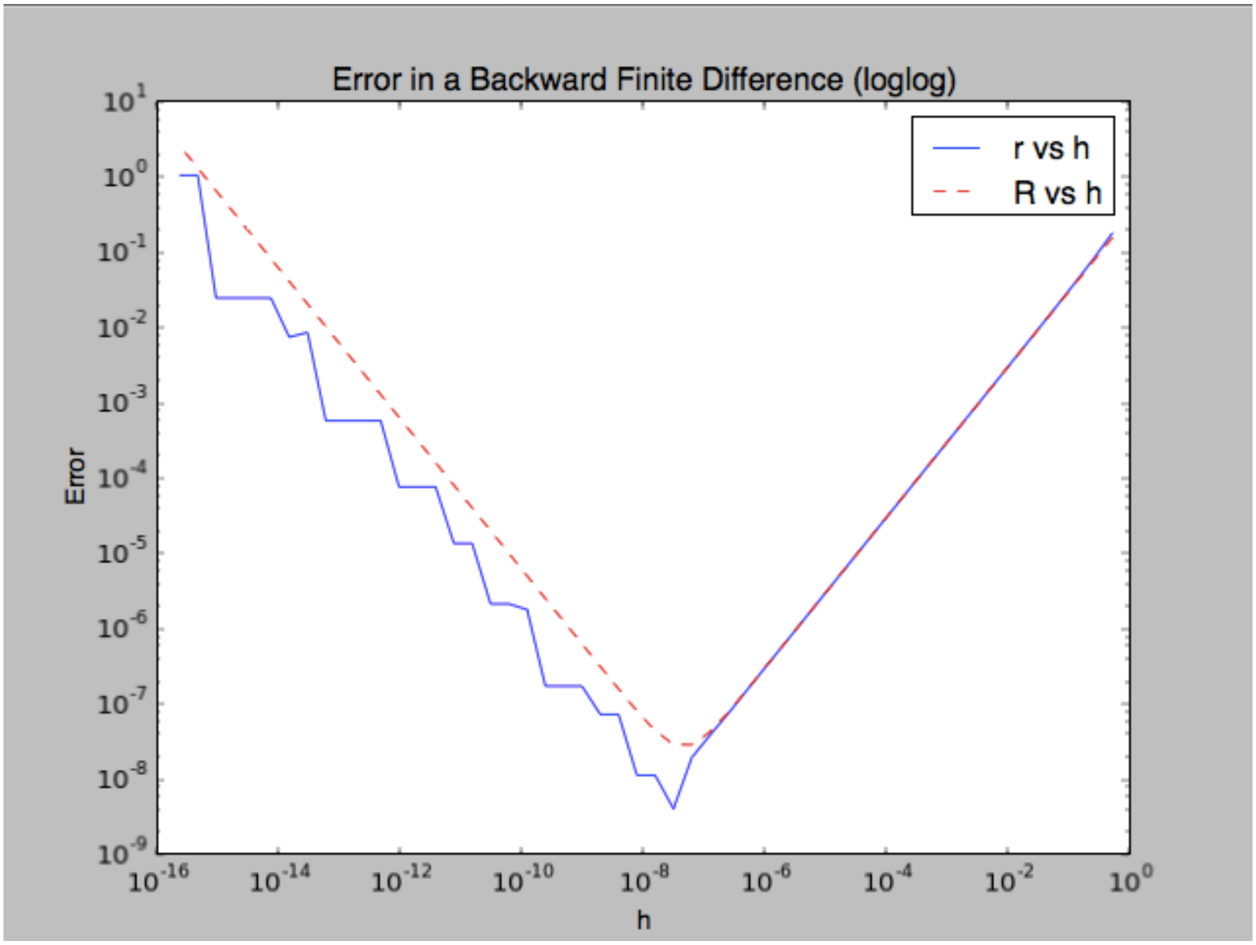


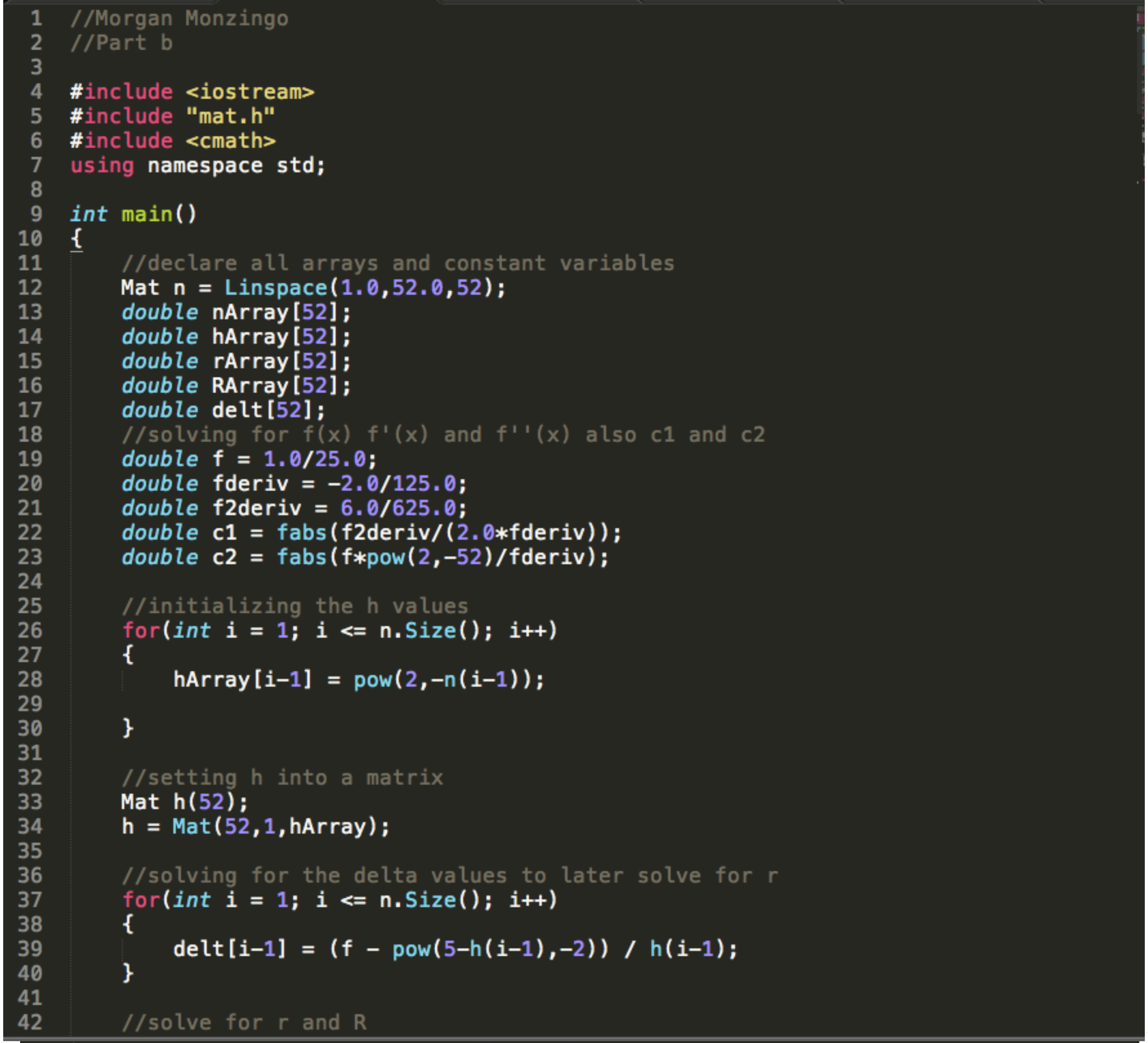
```
//Morgan Monzingo
//Nest function


#include "mat.h"


double nest(Mat &a, double x)
{
    int n= a.Size() - 1;
    double p = a(n);

    //loops through the matrix passed using horners equation
    for(int i = a.Size() - 2; i >= 0; i--)
    {
        p = p * x + a(i);
    }

    return p;

}
```



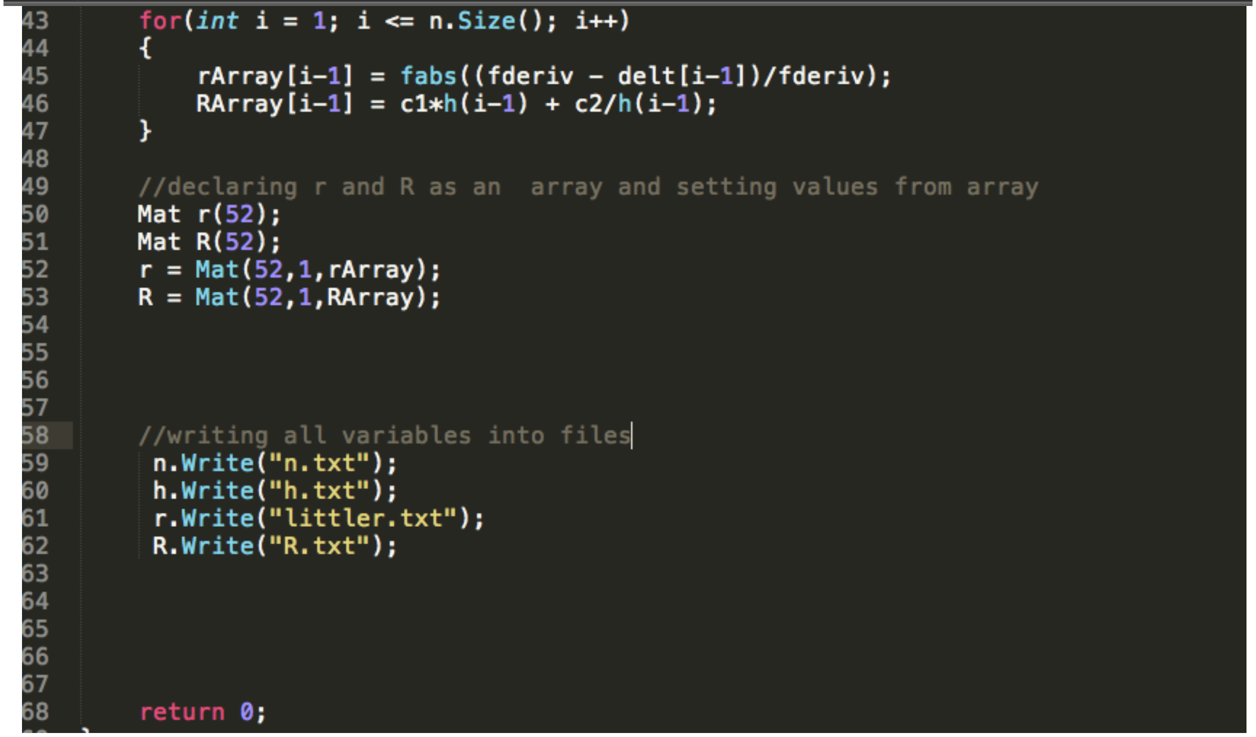
```python
#!/usr/bin/env python
#Morgan Monzingo
#Project 1
#9.8.14

import numpy as np
import matplotlib.pyplot as plt

#load all files into variables

z = np.loadtxt("z.txt")
f = np.loadtxt("f.txt")
p5 = np.loadtxt("p5.txt")
p9 = np.loadtxt("p9.txt")
p13 = np.loadtxt("p13.txt")
err5 = np.loadtxt("err5.txt")
err9 = np.loadtxt("err9.txt")
err13 = np.loadtxt("err13.txt")

#set graph one as the standard plad of f,p5,p9,p13 vs z
graph1 = plt.figure(1)
graph1 = plt.plot(z,f, label="sin(x)")
graph1 = plt.plot(z,p5, label="p5")
graph1 = plt.plot(z,p9, label="p9")
graph1 = plt.plot(z,p13, label="p13")
graph1 = plt.legend()
graph1 = plt.ylabel("value of polynomial")
graph1 = plt.xlabel("z")
graph1 = plt.title("sin(x) taylor series")
plt.show()

#plot the semilogy graph of all of the error
graph2 = plt.figure(2)
graph2 = plt.semilogy(z,err5, label = "err5")
graph2 = plt.semilogy(z,err9, label = "err9")
graph2 = plt.semilogy(z,err13, label = "err13")
graph2 = plt.legend()
graph2 = plt.ylabel("error")
graph2 = plt.xlabel("z")
graph2 = plt.title("Error")
plt.show()
```


Error in a Backward Finite Difference (semilogy)
r vs n
R vs n
Error
n
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
0
10
20
30
40
50
60

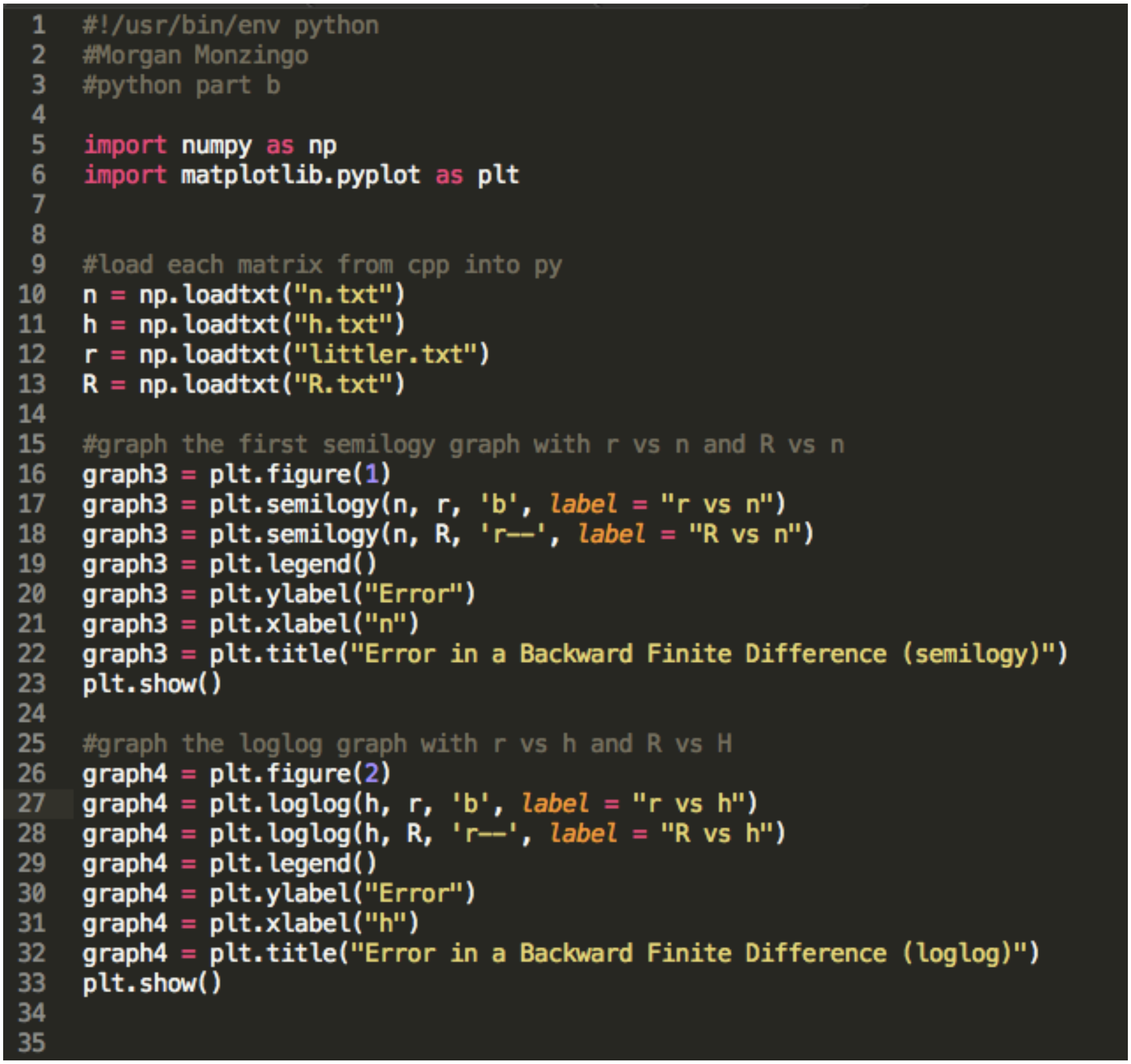

Error in a Backward Finite Difference (loglog)
r vs h
R vs h
Error
h
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-16}
10^{-14}
10^{-12}
10^{-10}
10^{-8}
10^{-6}
10^{-4}
10^{-2}
10^{0}



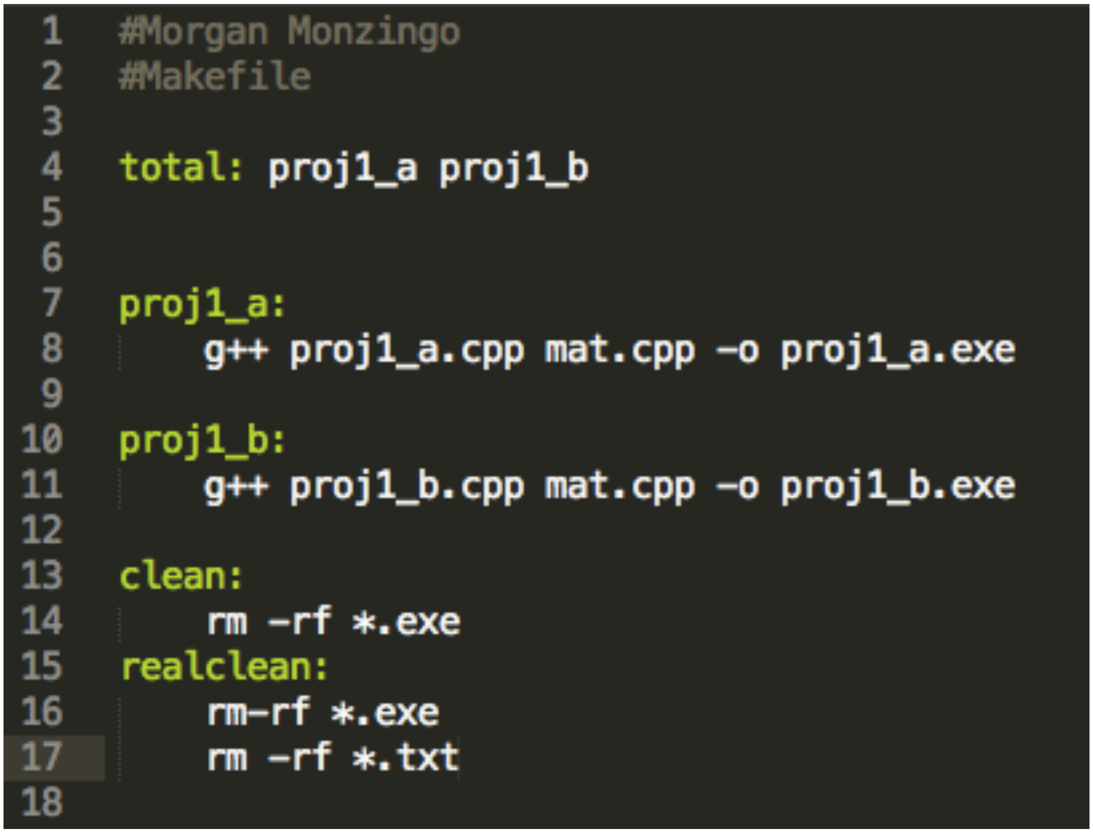
```cpp
//Morgan Monzingo
//Part b

#include <iostream>
#include "mat.h"
#include <cmath>
using namespace std;

int main()
{
    //declare all arrays and constant variables
    Mat n = Linspace(1.0,52.0,52);
    double nArray[52];
    double hArray[52];
    double rArray[52];
    double RArray[52];
    double delt[52];
    //solving for f(x) f'(x) and f''(x) also c1 and c2
    double f = 1.0/25.0;
    double fderiv = -2.0/125.0;
    double f2deriv = 6.0/625.0;
    double c1 = fabs(f2deriv/(2.0*fderiv));
    double c2 = fabs(f*pow(2,-52)/fderiv);

    //initializing the h values
    for(int i = 1; i <= n.Size(); i++)
    {
        hArray[i-1] = pow(2,-n(i-1));

    }

    //setting h into a matrix
    Mat h(52);
    h = Mat(52,1,hArray);

    //solving for the delta values to later solve for r
    for(int i = 1; i <= n.Size(); i++)
    {
        delt[i-1] = (f - pow(5-h(i-1),-2)) / h(i-1);
    }

    //solve for r and R
```



```cpp
    for(int i = 1; i <= n.Size(); i++)
    {
        rArray[i-1] = fabs((fderiv - delt[i-1])/fderiv);
        RArray[i-1] = c1*h(i-1) + c2/h(i-1);
    }

    //declaring r and R as an  array and setting values from array
    Mat r(52);
    Mat R(52);
    r = Mat(52,1,rArray);
    R = Mat(52,1,RArray);




    //writing all variables into files
     n.Write("n.txt");
     h.Write("h.txt");
     r.Write("littler.txt");
     R.Write("R.txt");





    return 0;
```



```python
#!/usr/bin/env python
#Morgan Monzingo
#python part b

import numpy as np
import matplotlib.pyplot as plt


#load each matrix from cpp into py
n = np.loadtxt("n.txt")
h = np.loadtxt("h.txt")
r = np.loadtxt("littler.txt")
R = np.loadtxt("R.txt")

#graph the first semilogy graph with r vs n and R vs n
graph3 = plt.figure(1)
graph3 = plt.semilogy(n, r, 'b', label = "r vs n")
graph3 = plt.semilogy(n, R, 'r--', label = "R vs n")
graph3 = plt.legend()
graph3 = plt.ylabel("Error")
graph3 = plt.xlabel("n")
graph3 = plt.title("Error in a Backward Finite Difference (semilogy)")
plt.show()

#graph the loglog graph with r vs h and R vs H
graph4 = plt.figure(2)
graph4 = plt.loglog(h, r, 'b', label = "r vs h")
graph4 = plt.loglog(h, R, 'r--', label = "R vs h")
graph4 = plt.legend()
graph4 = plt.ylabel("Error")
graph4 = plt.xlabel("h")
graph4 = plt.title("Error in a Backward Finite Difference (loglog)")
plt.show()


```



```
#Morgan Monzingo
#Makefile

total: proj1_a proj1_b


proj1_a:
	g++ proj1_a.cpp mat.cpp -o proj1_a.exe

proj1_b:
	g++ proj1_b.cpp mat.cpp -o proj1_b.exe

clean:
	rm -rf *.exe
realclean:
	rm-rf *.exe
	rm -rf *.txt

```