Project 1
Taylor Series and Floating Point Error

Erik Gabrielsen

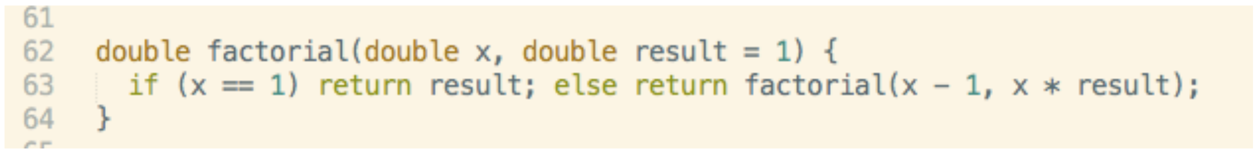
09/16/2015

MATH 3316

Part I

For part one of this project, we were to approximate the function f=cos(x) by creating a nested multiplication algorithm that will evaluate the n-degree polynomial for cos(x). This algorithm takes in a vector of coefficients (a(0) – a(n)) and a value x to compute the Taylor Series of cos(x) to the nth polynomial. In this particular project, we are required to use the nest algorithm to expand the Taylor polynomial for cos(x) at 4, 8, and 12 and compare each polynomial to cos(x) to see which would yield the most accurate approximation.

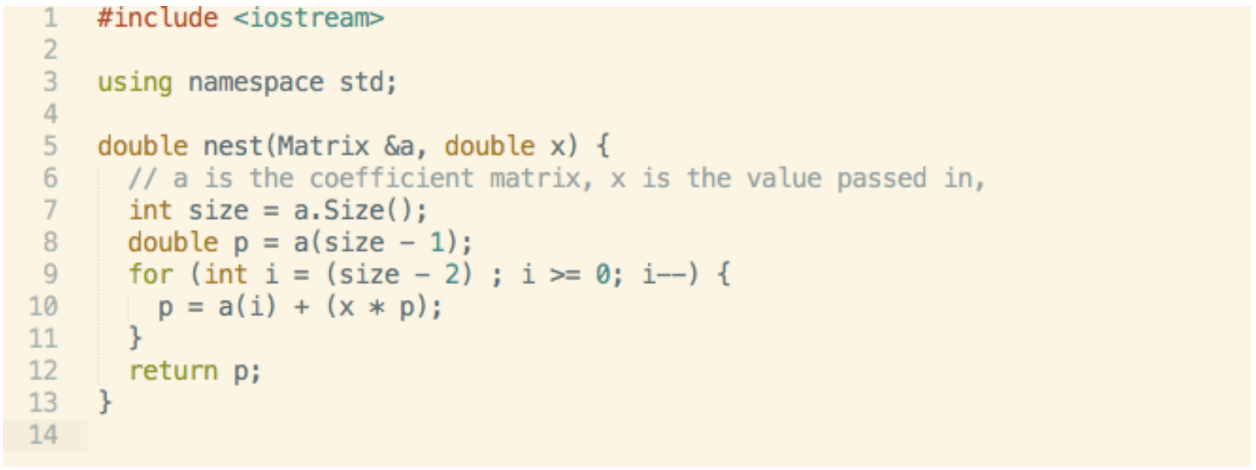
To represent the coefficient values, I decided to create my own factorial function that recursively computed the factorials in the denominator of the Taylor Series for cos(x) as follows:



```
double factorial(double x, double result = 1) {
  if (x == 1) return result; else return factorial(x - 1, x * result);
}
```

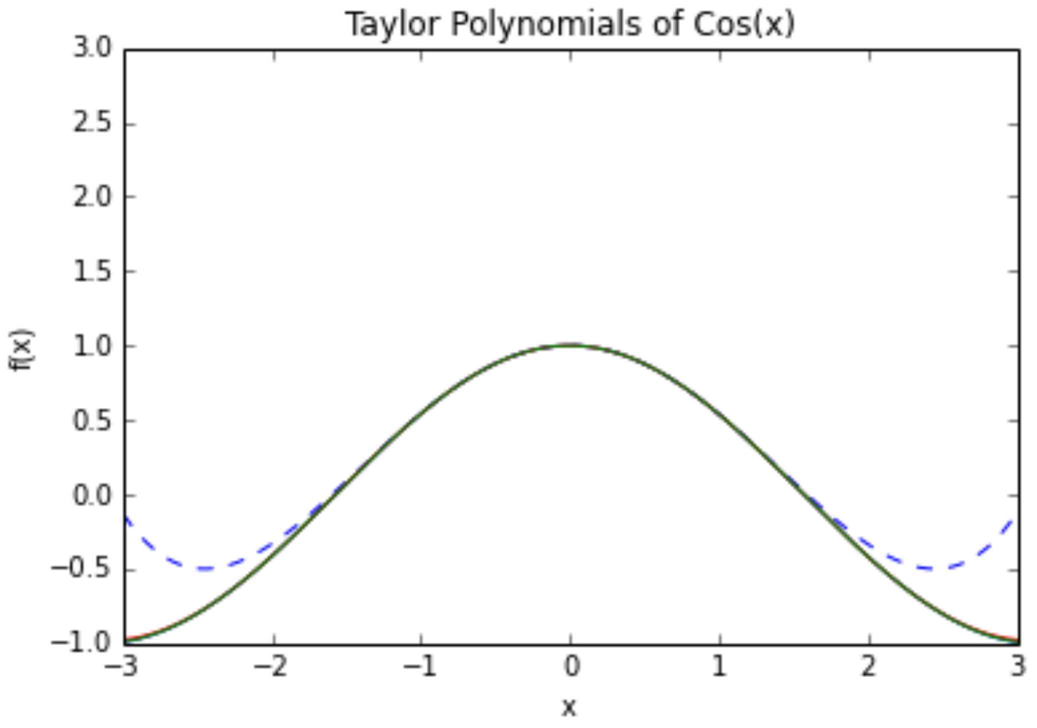
My reasoning behind this was so that it was clear where these values came from in the program and that the coefficients for cos(x) come from its Taylor Series and not from thin air. Using this factorial algorithm, I used a linear matrix to represent the coefficients for p4, p8, and p12

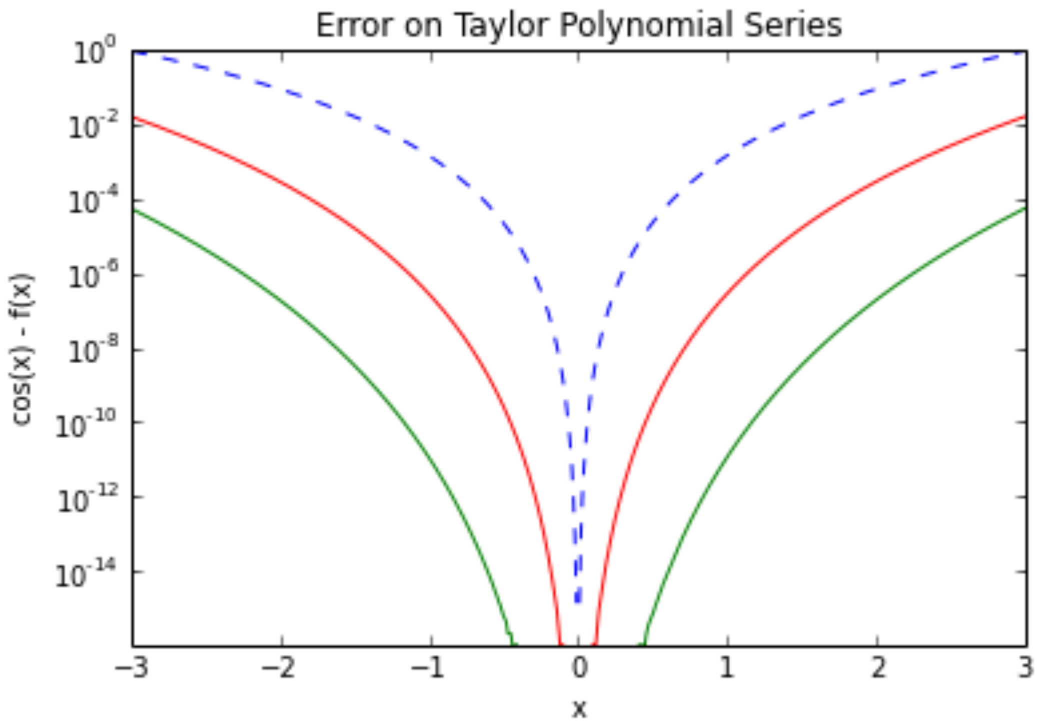
In order to show this approximation for each polynomial series, we created a row vector, z, which contained values -3, -2.99, -2.98 …, 2.98, 2.99, 3. We would then iterate through each value of the vector z (containing 601 values) to show the approximation given by the polynomial series of p4, p8, and p12, as well as the error (|cosx| - |p(4, 8, 12)|) from truncating the polynomial taylor series of cos(x). I used the Nested Multiplication pseudo code provided by the textbook to come up with the nest function.



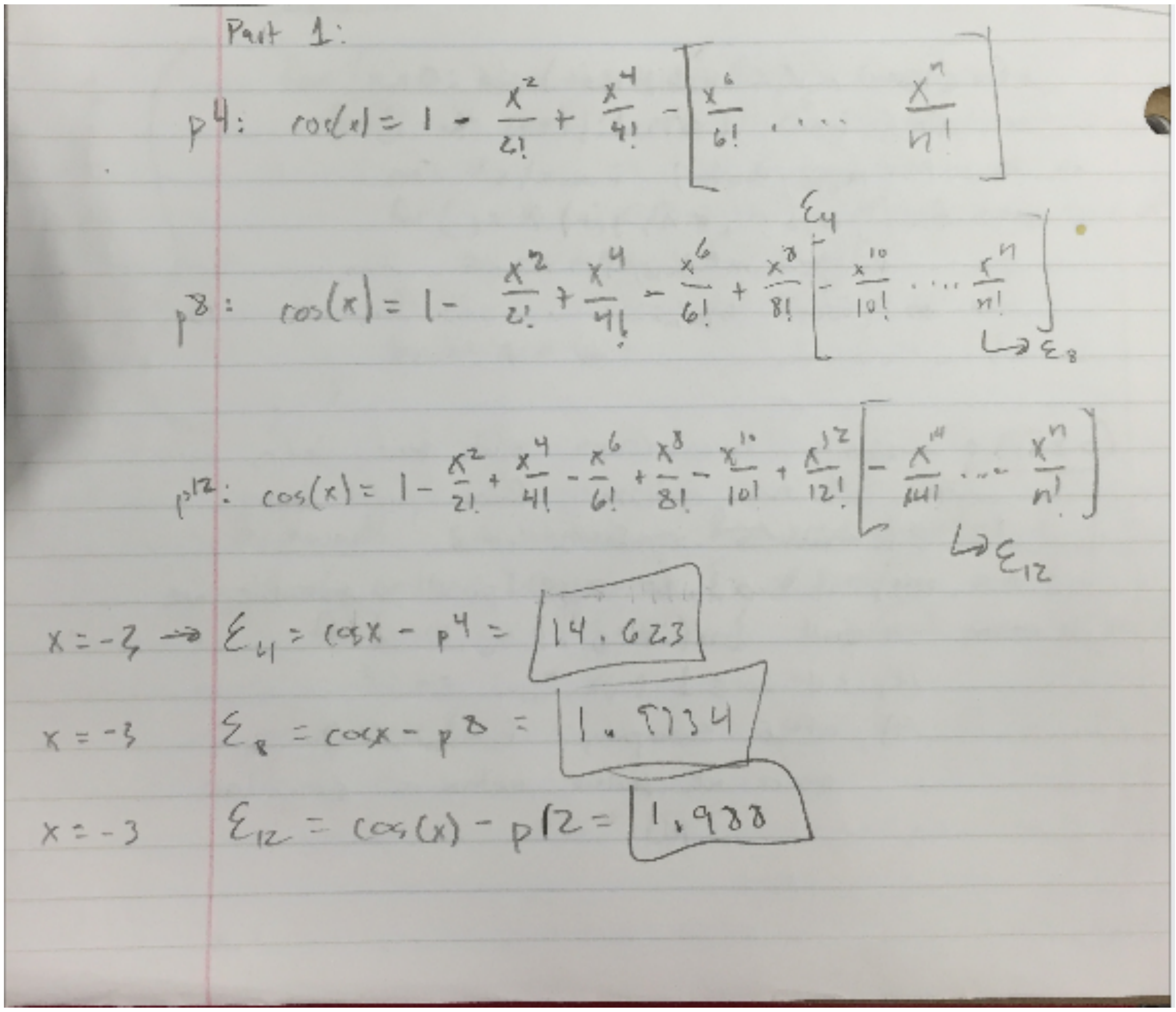
```
#include <iostream>

using namespace std;

double nest(Matrix &a, double x) {
  // a is the coefficient matrix, x is the value passed in,
  int size = a.Size();
  double p = a(size - 1);
  for (int i = (size - 2) ; i >= 0; i--) {
    p = a(i) + (x * p);
  }
  return p;
}
```

After computing and writing out the results of the approximation and error I wrote a short python script in iPython that would put the data into a graph.





The first graph is the approximation of cos(x) using the polynomials 4, 8, and 12. It becomes clear after looking at the graph that as the polynomial n increases, the approximation gets more accurate and closer to the actual value of cosine for that value of x. Additionally, using the semilogy() function of iPython we are able to see that this is indeed the case. As the polynomial increases for cos(x) we see that the error is reduced. This is because of the truncation error discussed in chapter 1 of the book. Its impossible for a computer to represent cos(x) with absolute precision because the polynomial representation will continue to infinity, but using a bigger polynomial number will get the approximation closer to the actual number. Therefore, its obvious that p12 is the best approximation of the function cosine. Here are my manual derivations of the upper bound on the error for each approximation: p4, p8, and p12 respectively:



Part 1:

$$p4: \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \left[\frac{x^6}{6!} \cdots \frac{x^n}{n!}\right]$$

$$\varepsilon_4$$

$$p8: \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \left[- \frac{x^{10}}{10!} \cdots \frac{x^n}{n!}\right] \hookrightarrow \varepsilon_8$$

$$p12: \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} \left[- \frac{x^{14}}{14!} \cdots \frac{x^n}{n!}\right] \hookrightarrow \varepsilon_{12}$$

$x = -3 \rightarrow \varepsilon_4 = \cos x - p4 = \boxed{14.623}$

$x = -3 \quad \varepsilon_8 = \cos x - p8 = \boxed{1.5734}$

$x = -3 \quad \varepsilon_{12} = \cos(x) - p12 = \boxed{1.988}$

Based off of my computations one can see that the error decreases as the n polynomial increases from p4 to p12 which is consistent with the data I received from the graphs.

Part II

In Part 2 of the lab we are responsible for writing a program to compute the forward difference estimates δ +f(3) of the derivative f'(3) for the function f(x) = x ^(−3) with the sequence of increments h = 2−n (n = 1, 2, 3, . . . , 52). For each of these values we also need to find the relative error (r) and the upper bound on the relative error (R) and save all of this data to disk.

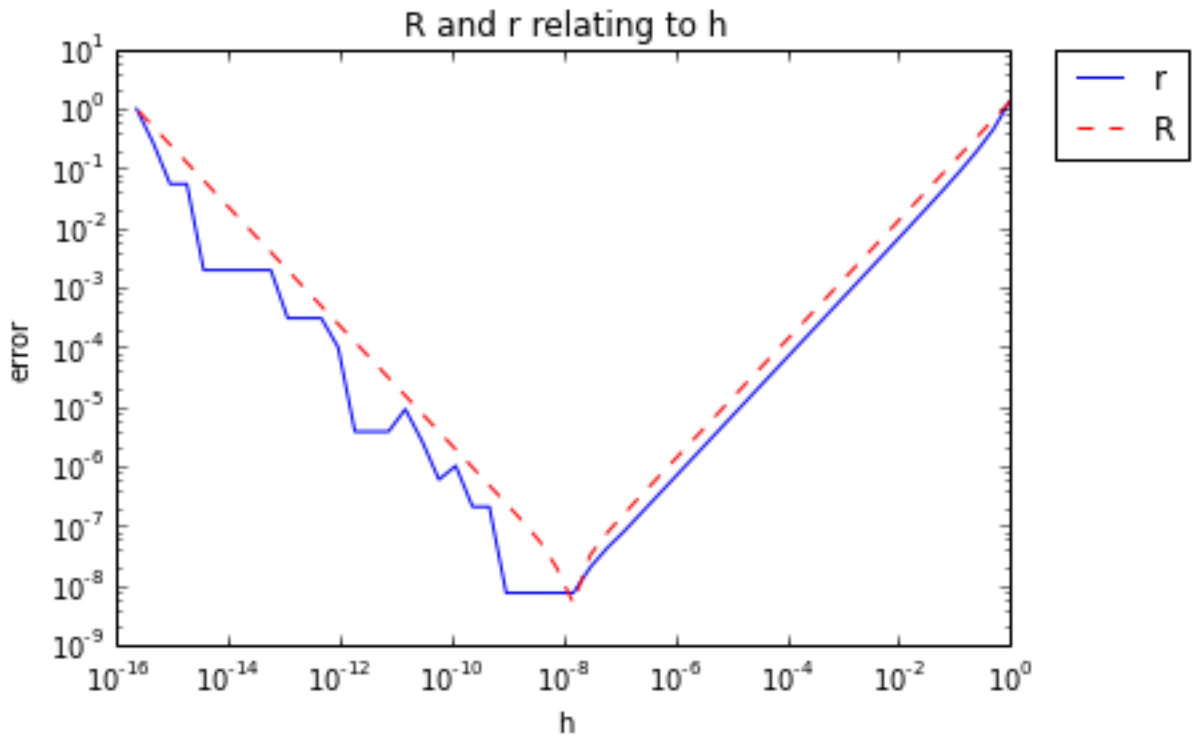
The relative error is calculated as r = |(f'(5) – df(a)) / f'(5) where df(a) is

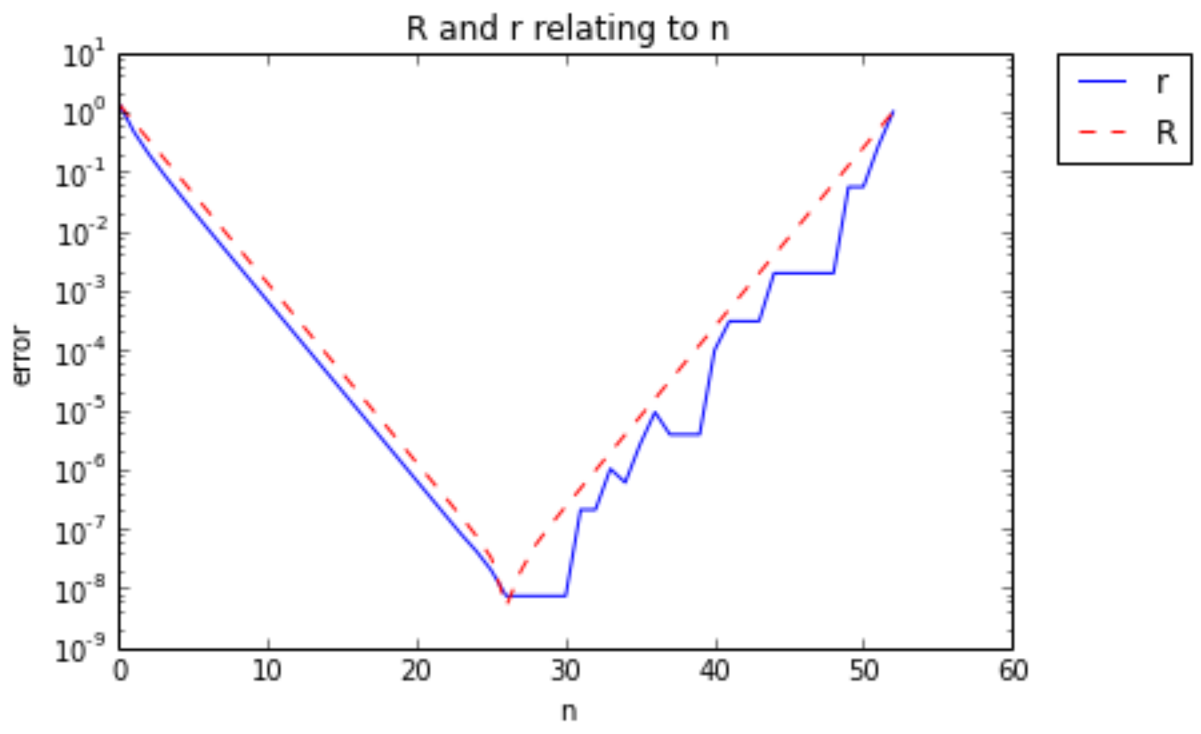
(f(a) – f(a – h)) / h.

The upper bound (R) is calculated as R = (c1 * h) + (c2 / h) where the constants c1 and c2 are:

C1 = | f''(a) / (2 * f'(a)) |
C2 = | (f(a) * eDP) / f'(a) |

I used both of these sequence of equations to determine the vectors r and R and save them to disk. In iPython I uploaded each files and showed the error data in the form of 2 graphs:



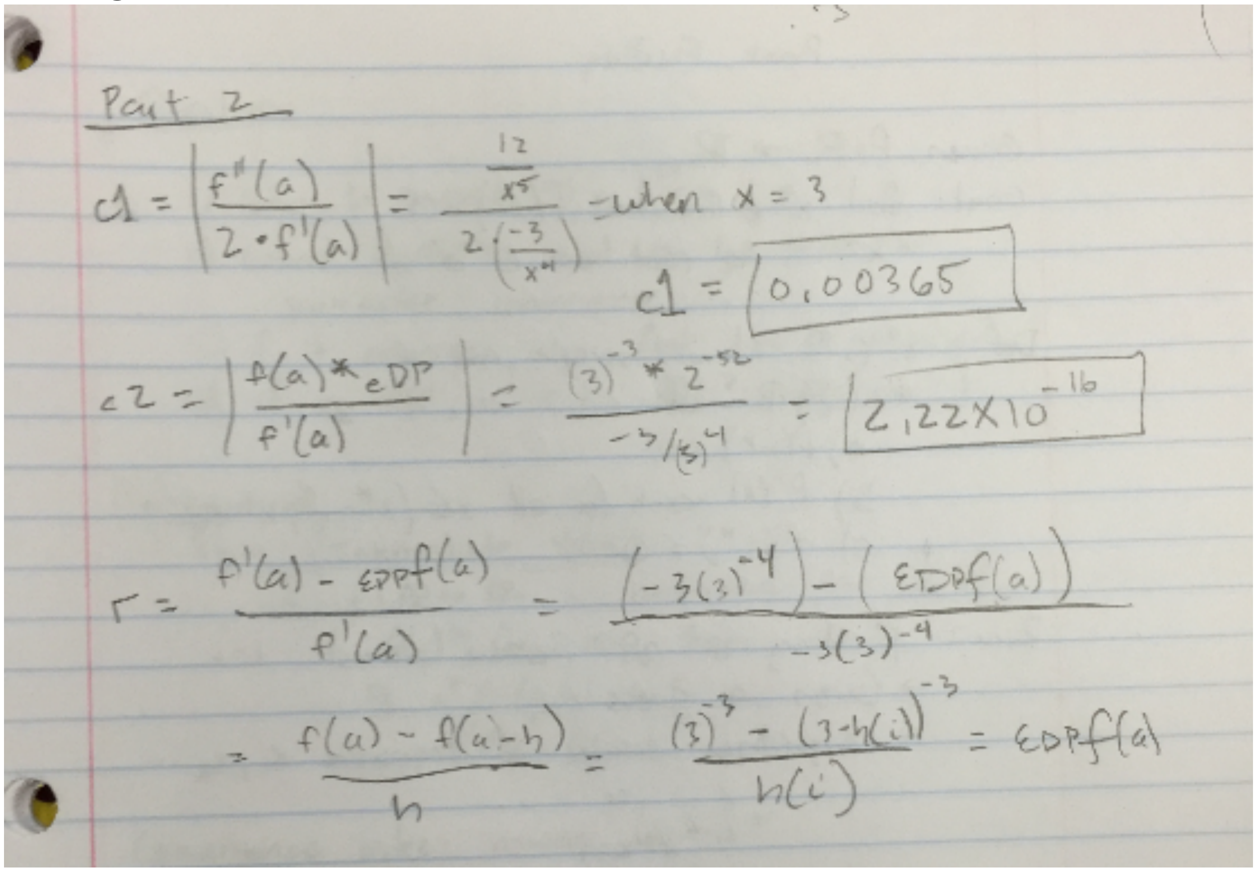


The first graph is a comparison of n versus r and n versus R. I used a semilogy() plot to put a logarithmic scale in the y-axis. In the second graph, each error (r and R) are graphed versus h using a loglog() command that puts both the x and y axis in a logarithmic scale.

For each of these graphs the relative error falls on or below the upper bound of relative error which is what is expected seeing as how R is the upper bound. It also appears that the error is reduced the most when n is close to about 26. When n > 26 the graph changes and r fluctuates closer and further away from the upper bound R. This is most likely due to precision floating point error as h begins to get exponentially smaller making the numbers stored in the computer increasingly larger. A similar phenomenon occurs with the second graph. We notice that when h = 10^(-8) both errors are at their lowest point which confirms my belief that as h gets smaller than 10^(-8) the errors vary and begin to increase due to floating point precision error.

Relevant Code/ Work

Solving for R and r:



Part 2

$$c1 = \left|\frac{f''(a)}{2\cdot f'(a)}\right| = \frac{\frac{12}{x^5}}{2\left(\frac{-3}{x^4}\right)} = \text{when } x = 3$$

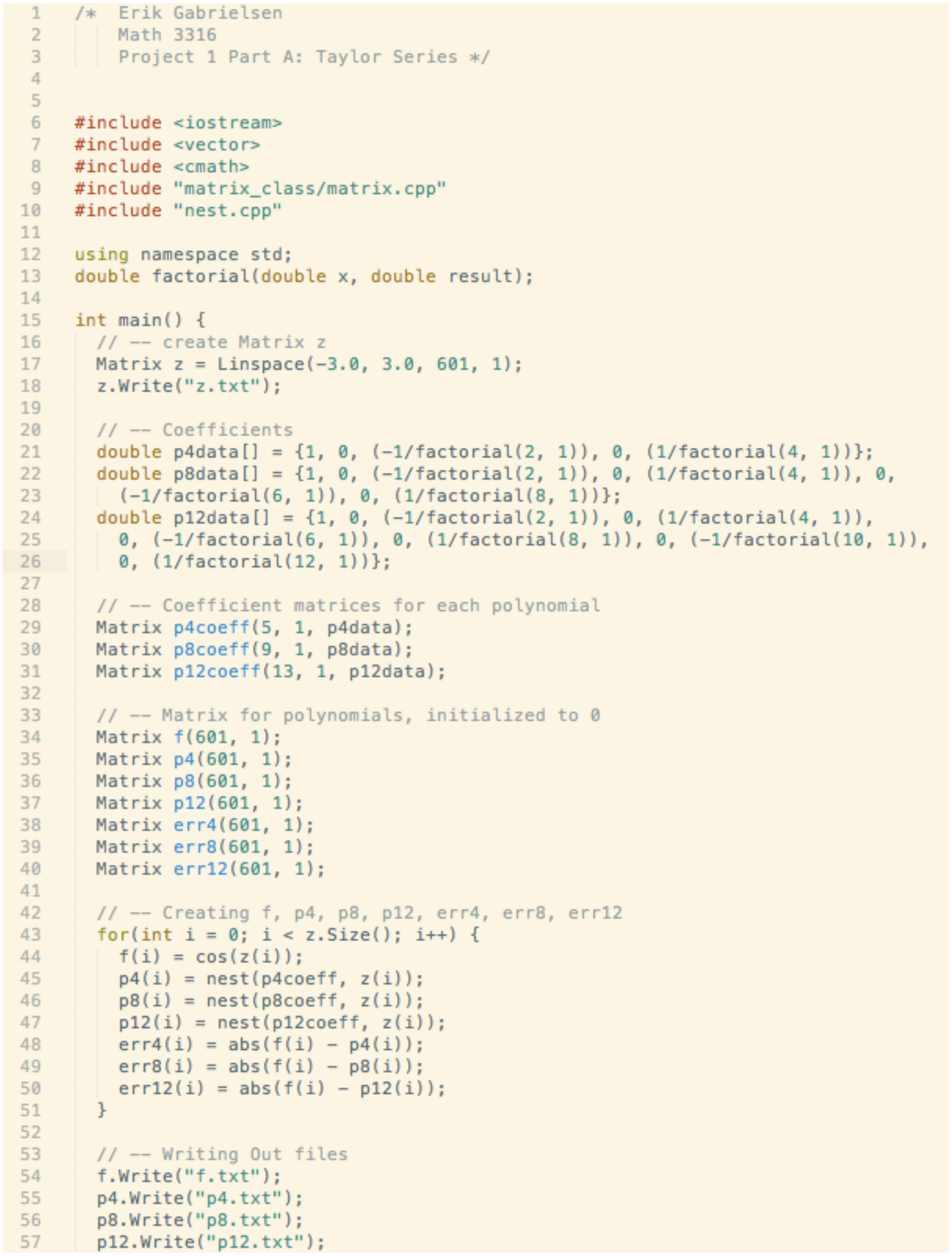
$$c1 = \boxed{0.00365}$$

$$c2 = \left|\frac{f(a) * eDP}{f'(a)}\right| = \frac{(3)^{-3} * 2^{-52}}{-3/(3)^4} = \boxed{2.22\times10^{-16}}$$

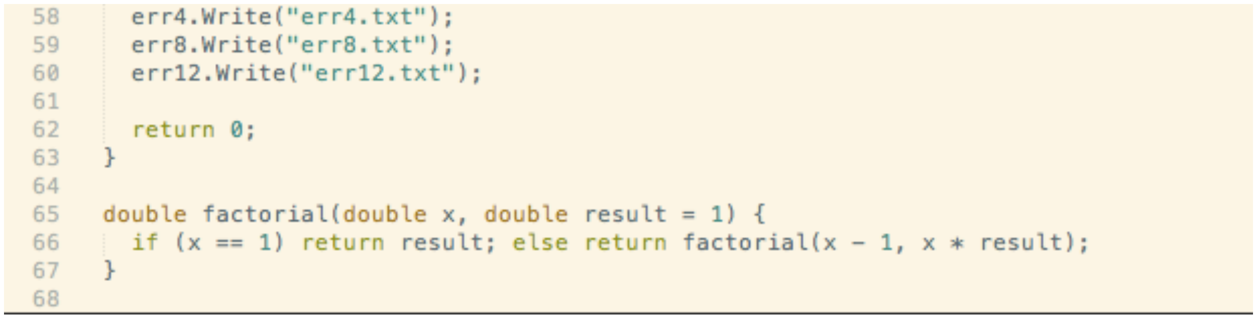
$$r = \frac{f'(a) - \varepsilon DP f(a)}{f'(a)} = \frac{\left(-3(3)^{-4}\right) - \left(\varepsilon DP f(a)\right)}{-3(3)^{-4}}$$

$$= \frac{f(a) - f(a-h)}{h} = \frac{(3)^{-3} - (3-h(i))^{-3}}{h(i)} = \varepsilon DP f(a)$$

proj1_a.cpp

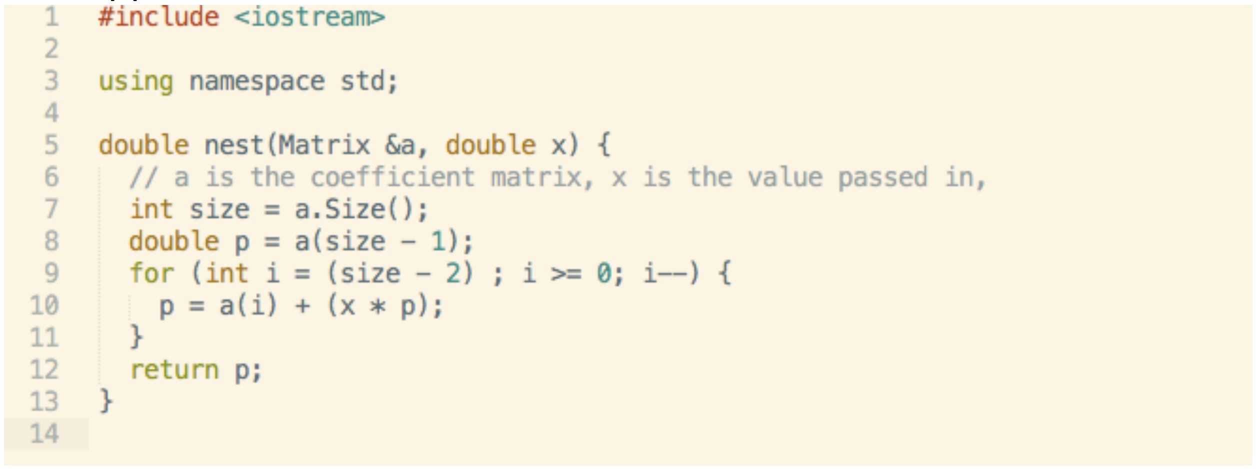


```
/*  Erik Gabrielsen
    Math 3316
    Project 1 Part A: Taylor Series */


#include <iostream>
#include <vector>
#include <cmath>
#include "matrix_class/matrix.cpp"
#include "nest.cpp"

using namespace std;
double factorial(double x, double result);

int main() {
  // -- create Matrix z
  Matrix z = Linspace(-3.0, 3.0, 601, 1);
  z.Write("z.txt");

  // -- Coefficients
  double p4data[] = {1, 0, (-1/factorial(2, 1)), 0, (1/factorial(4, 1))};
  double p8data[] = {1, 0, (-1/factorial(2, 1)), 0, (1/factorial(4, 1)), 0,
    (-1/factorial(6, 1)), 0, (1/factorial(8, 1))};
  double p12data[] = {1, 0, (-1/factorial(2, 1)), 0, (1/factorial(4, 1)),
    0, (-1/factorial(6, 1)), 0, (1/factorial(8, 1)), 0, (-1/factorial(10, 1)),
    0, (1/factorial(12, 1))};

  // -- Coefficient matrices for each polynomial
  Matrix p4coeff(5, 1, p4data);
  Matrix p8coeff(9, 1, p8data);
  Matrix p12coeff(13, 1, p12data);

  // -- Matrix for polynomials, initialized to 0
  Matrix f(601, 1);
  Matrix p4(601, 1);
  Matrix p8(601, 1);
  Matrix p12(601, 1);
  Matrix err4(601, 1);
  Matrix err8(601, 1);
  Matrix err12(601, 1);

  // -- Creating f, p4, p8, p12, err4, err8, err12
  for(int i = 0; i < z.Size(); i++) {
    f(i) = cos(z(i));
    p4(i) = nest(p4coeff, z(i));
    p8(i) = nest(p8coeff, z(i));
    p12(i) = nest(p12coeff, z(i));
    err4(i) = abs(f(i) - p4(i));
    err8(i) = abs(f(i) - p8(i));
    err12(i) = abs(f(i) - p12(i));
  }

  // -- Writing Out files
  f.Write("f.txt");
  p4.Write("p4.txt");
  p8.Write("p8.txt");
  p12.Write("p12.txt");
```



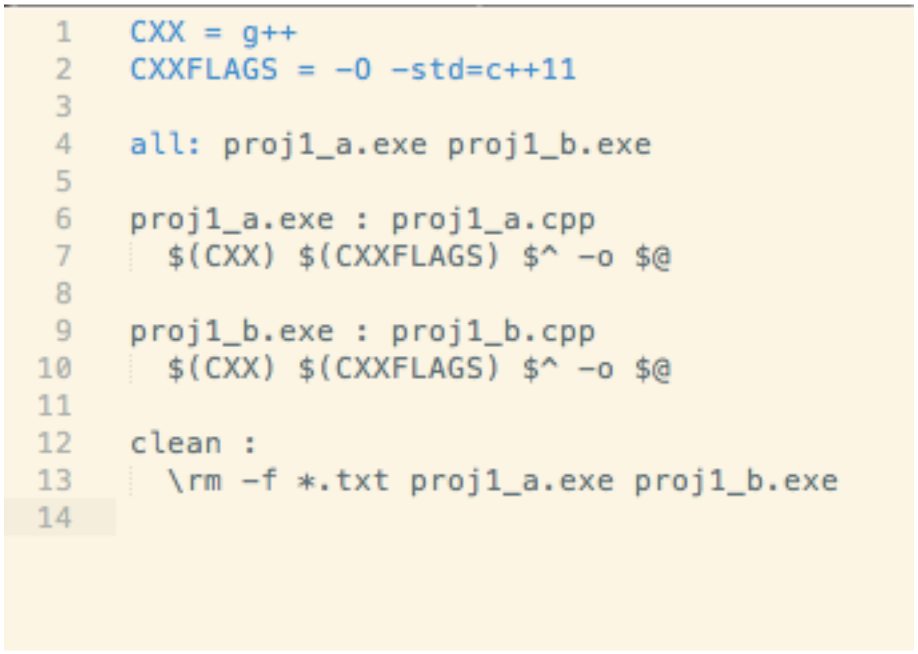
```cpp
  err4.Write("err4.txt");
  err8.Write("err8.txt");
  err12.Write("err12.txt");

  return 0;
}

double factorial(double x, double result = 1) {
  if (x == 1) return result; else return factorial(x - 1, x * result);
}

```

nest.cpp



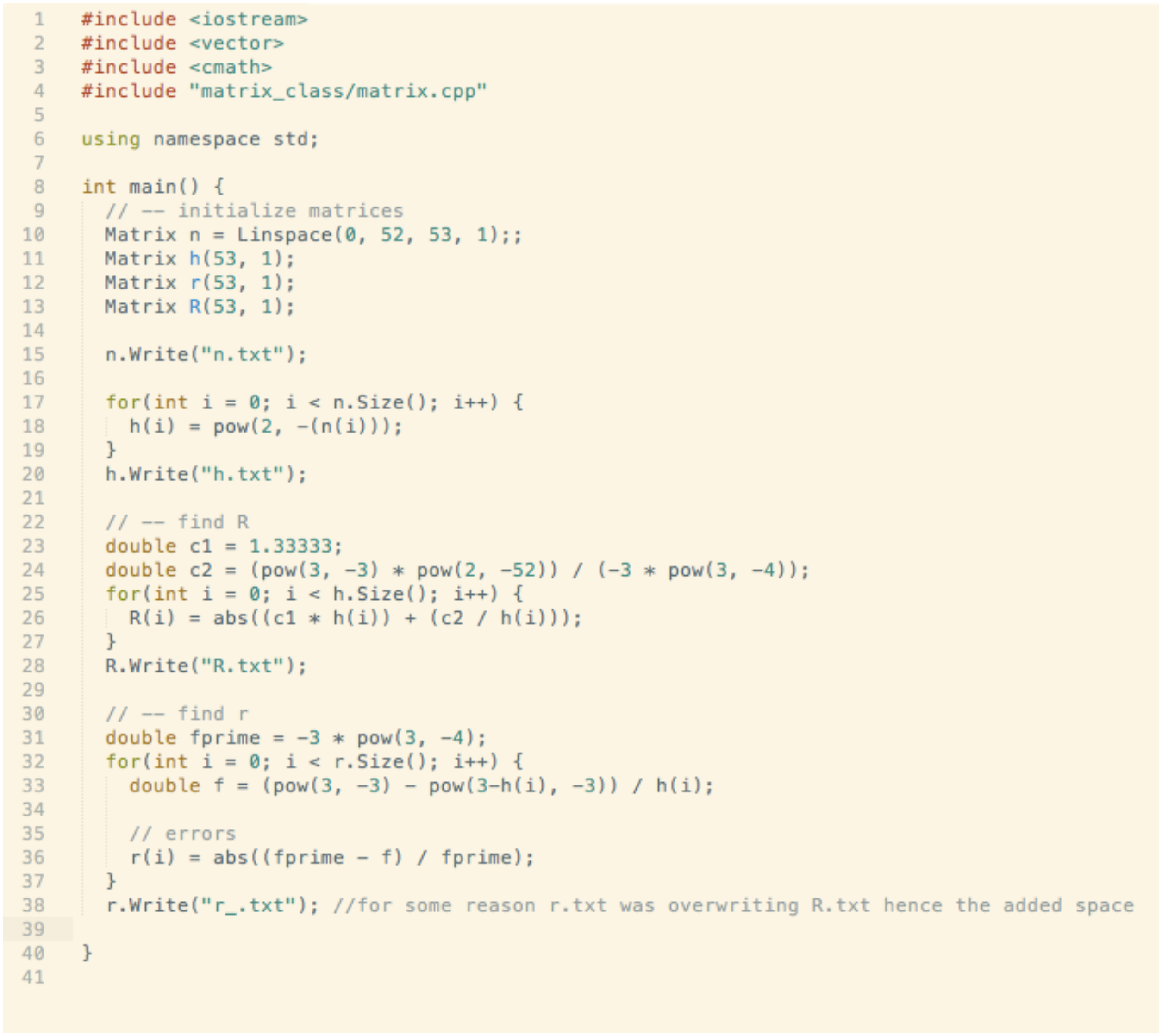
```cpp
#include <iostream>

using namespace std;

double nest(Matrix &a, double x) {
  // a is the coefficient matrix, x is the value passed in,
  int size = a.Size();
  double p = a(size - 1);
  for (int i = (size - 2) ; i >= 0; i--) {
    p = a(i) + (x * p);
  }
  return p;
}

```

Makefile



```makefile
CXX = g++
CXXFLAGS = -O -std=c++11

all: proj1_a.exe proj1_b.exe

proj1_a.exe : proj1_a.cpp
	$(CXX) $(CXXFLAGS) $^ -o $@

proj1_b.exe : proj1_b.cpp
	$(CXX) $(CXXFLAGS) $^ -o $@

clean :
	\rm -f *.txt proj1_a.exe proj1_b.exe

```

Proj1_b.cpp



```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include "matrix_class/matrix.cpp"

using namespace std;

int main() {
  // -- initialize matrices
  Matrix n = Linspace(0, 52, 53, 1);;
  Matrix h(53, 1);
  Matrix r(53, 1);
  Matrix R(53, 1);

  n.Write("n.txt");

  for(int i = 0; i < n.Size(); i++) {
    h(i) = pow(2, -(n(i)));
  }
  h.Write("h.txt");

  // -- find R
  double c1 = 1.33333;
  double c2 = (pow(3, -3) * pow(2, -52)) / (-3 * pow(3, -4));
  for(int i = 0; i < h.Size(); i++) {
    R(i) = abs((c1 * h(i)) + (c2 / h(i)));
  }
  R.Write("R.txt");

  // -- find r
  double fprime = -3 * pow(3, -4);
  for(int i = 0; i < r.Size(); i++) {
    double f = (pow(3, -3) - pow(3-h(i), -3)) / h(i);

    // errors
    r(i) = abs((fprime - f) / fprime);
  }
  r.Write("r_.txt"); //for some reason r.txt was overwriting R.txt hence the added space

}

```