Project 1 Report

Taylor Ellington

September 17, 2015

1 Part A

First I implemented the nest function as per the pseudocode in the textbook

```
#include "nest.hpp"

    double nest ( const Matrix& a, double x ){
        int size = a.Size() -1 ;
        double p = a(size);

        for( int i = size -1;  i >= 0 ; i--){
            p = a(i) + ( x*p ) ;
        }

        return p;
    }
```

This allowed me to solve n-degree polynomials.

Next, I solved out the coefficients for the Taylor Polynomial. This was done with the assistance of wolfram alpha. These coefficients are hardcoded into a double[]. Looking back I realize that there could be some roundoff error or other consequences for approaching these numbers this way.The following code is the entirety of proj1_a.cpp.

```
int main(int argc, char * argv[]){


    //create row vector z, from -3 to 3
    Matrix z =Linspace(-3, 3, 1, 600);

    //create vector of coefficients for taylor polynimial for cos x

    double coeff_temp[13] = {
            1,
            0,
            -0.5,
            0,
            0.041666666666666666666666666,
            0,
            -0.001388888888888888888888888,
            0,
```

```
        0.0000248015873015873015873015873015873015873015,
        0,
        -0.00000027557319223985890652,
        0,
        0.000000002087675698786809897921009032120143231254342365453476564 5};


//compute p4
Matrix coeff4(1, 4, coeff_temp);
Matrix  p4(z.Size());
for(int i = 0; i < 600; i++){
    p4(i) =  nest(coeff4, z(i));
}

//compute p8
Matrix coeff8(1, 8, coeff_temp);
Matrix  p8(z.Size());
for(int i = 0; i < 600; i++){
    p8(i) = nest(coeff8, z(i));
}

//compute p12
Matrix coeff12(1, 12, coeff_temp);
Matrix  p12(z.Size());
for(int i = 0; i < 600; i++){
    p12(i) =  nest(coeff12, z(i));
}

//compute f
Matrix  f(z.Size());
for(int i = 0; i < 600; i++){
    f(i) = cos( z(i) );
}

//compute err4
Matrix  err4(z.Size());
for(int i = 0; i < 600; i++){
    err4(i) = abs(f(i) - p4(i));
}
//compute err8
Matrix  err8(z.Size());
for(int i = 0; i < 600; i++){
    err8(i) = abs(f(i) - p8(i));
}
//compute err12
Matrix  err12(z.Size());
for(int i = 0; i < 600; i++){
    err12(i) = abs(f(i) - p12(i));
}

// put everything on disk
z.Write("z.txt");
p4.Write("p4.txt");
p8.Write("p8.txt");
```

```
    p12.Write("p12.txt");
    f.Write("f.txt");
    err4.Write("err4.txt");
    err8.Write("err8.txt");
    err12.Write("err12.txt");

    return 0;
}
```

Once this concludes executing, I have 8 text files contaning all the data needed to make the required plots. THe next step is to read them all in to the python environment. I noticed the Matrix.Write() function creates comma seperated lists, so i ulitized the python csv library to quickly read in all the data to appropriately named arrays. appropriately

In [2]:

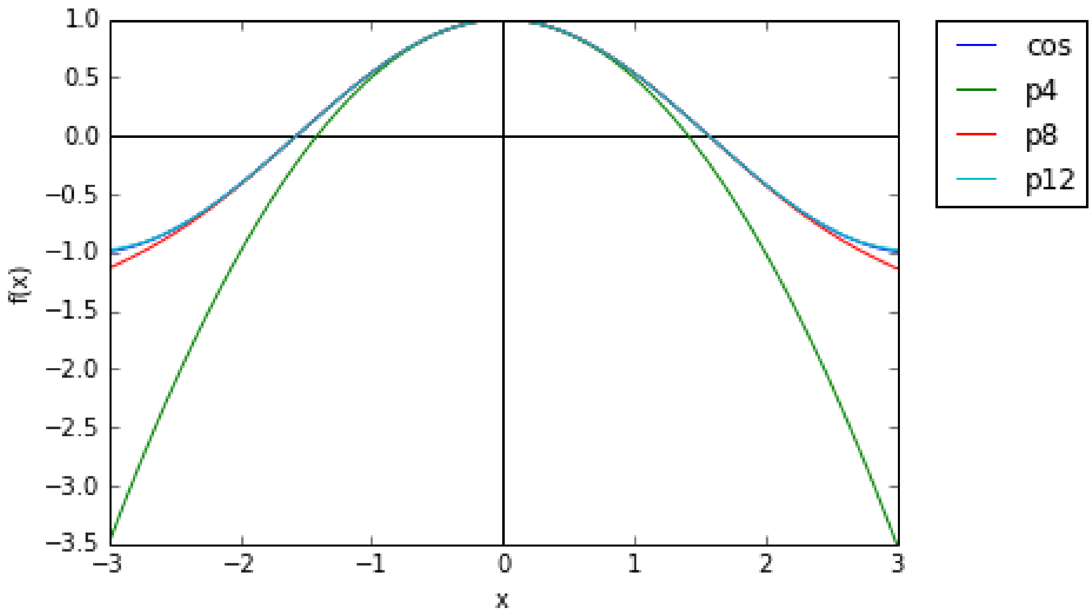
```python
import csv

with open('f.txt', 'rb') as file:
    f = file.read().replace('\n', '').split('  ')
    f.remove('')
    file.close()

with open('z.txt', 'rb') as file:
    z = file.read().replace('\n', '').split('  ')
    z.remove('')
    file.close()

with open('p4.txt', 'rb') as file:
    p4 = file.read().replace('\n', '').split('  ')
    p4.remove('')
    file.close()

with open('p8.txt', 'rb') as file:
    p8 = file.read().replace('\n', '').split('  ')
    p8.remove('')
    file.close()

with open('p12.txt', 'rb') as file:
    p12 = file.read().replace('\n', '').split('  ')
    p12.remove('')
    file.close()

with open('err4.txt', 'rb') as file:
    err4 = file.read().replace('\n', '').split('  ')
    err4.remove('')
    file.close()

with open('err8.txt', 'rb') as file:
    err8 = file.read().replace('\n', '').split('  ')
    err8.remove('')
    file.close()

with open('err12.txt', 'rb') as file:
    err12 = file.read().replace('\n', '').split('  ')
    err12.remove('')
    file.close()
```

From here, I can create the required plots.

In [3]:

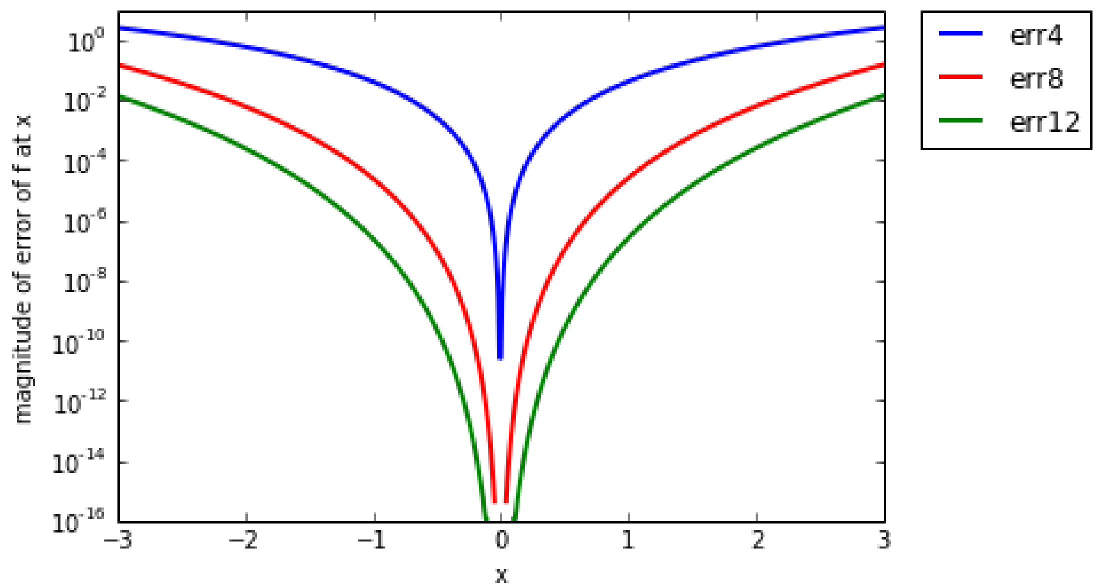
```python
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np

plt.axhline(0, color='black')
plt.axvline(0, color='black')
```

```
plt.ylabel( 'f(x)' )
plt.xlabel( 'x'   )

plt.plot(z, f, label='cos')
plt.plot(z, p4, label='p4')
plt.plot(z, p8, label='p8')
plt.plot(z, p12, label='p12')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)


plt.show()
```



In [4]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np




fig = plt.figure()
ax = fig.add_subplot(1,1,1)

plt.ylabel( 'magnitude of error of f at x' )
plt.xlabel( 'x'   )

line, = ax.semilogy(z, err4, color='blue', label="err4", lw=2)
line2, = ax.semilogy(z ,err8, color='red', lw=2, label="err8")
line3, = ax.semilogy(z, err12, color='green', lw=2, label="err12" )

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)


plt.show()
```



With the data now plotted, it quickly becomes apparent that the more terms in a Taylor series, the better approximation you get in the end. both p4 and p8 have error magnitudes greater than 1 by the outer limits of what we calculated. p12 remains a decent approxmiation however it too has a large error at the extremes versus at 0. However, p12 is the best approximation for cos(-3) out of the possible choices.

looking at the plots, we can see that the upper bound of the error for each of these 3 seies is found at the extremes, so we can evaluate the series at that value and get the bounds.

$$E_4 = 2.510007503399555$$
$$E_8 = 0.1475075033995548$$
$$E_12 = 0.013588478743330227$$

This is consistent with what we expect from the graphs, as we increase the number of terms from 4 to 8 to 12 the error decreases. The decrease is almost around an order of magnitude on each step. These intervals are consistent with what the semilog plot shows.

2 Part B

For this step all of the mathematics is easly done in code, except for getting the derivations of f(a). These I did by hand.

$$f(a) = x^{-3}$$
$$f'(a) = -3x^{-4}$$
$$f''(a) = 12x^{-5}$$

for neatness' sake, I seperated these functions out in to their own methods, accepting a double and returing one as well. These methods precede the main method of proj2_b.cpp.

```
#include "matrix.hpp"
#include <stdio.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <math.h>
using namespace std;

double f(double a){
    return ( pow(a, -3) );
}

double fPrime(double a){
    return ( -3 * ( pow(a, -4 ) ) );
}

double fDoublePrime(double a){
    return (12 * ( pow(a, -5) ) );
}

int main(int argc, char* argv[] ) {

    //SETUP
    //Make row vector with increments 1...52, Delta = 1.0
    Matrix n = Linspace(1, 52, 1 , 52 );

    //declare other matrices that are important
    Matrix h( n.Size() );
    Matrix r( n.Size() );
    Matrix R( n.Size() );

    // calculate out h
    for(int i = 0; i < n.Size(); i++){
        h(i) = pow(2, (-1*n(i) ) );
    }

    // holder double for FFD
    double forwardFiniteDifference;

    //C constants
    double c1 = abs((fDoublePrime(3))/( 2* fPrime(3)));
    double c2 = abs(  (f(3) * pow(2, -52) ) / (fPrime(3) )   );

    //Major calculation loop
    for(int i = 0; i < n.Size(); i++){
        forwardFiniteDifference = (f(3 + h(i)) - f(3) )/ h(i);


        r(i) = abs(  (fPrime(3) - forwardFiniteDifference) / ( fPrime(3) ) );

        R(i) = (c1 * h(i) ) + (c2 / h(i) );

    }
```

```
    // save everything
    n.Write("n.txt");
    h.Write("h.txt");
    r.Write("r.txt");
    R.Write("R.txt");


}
```

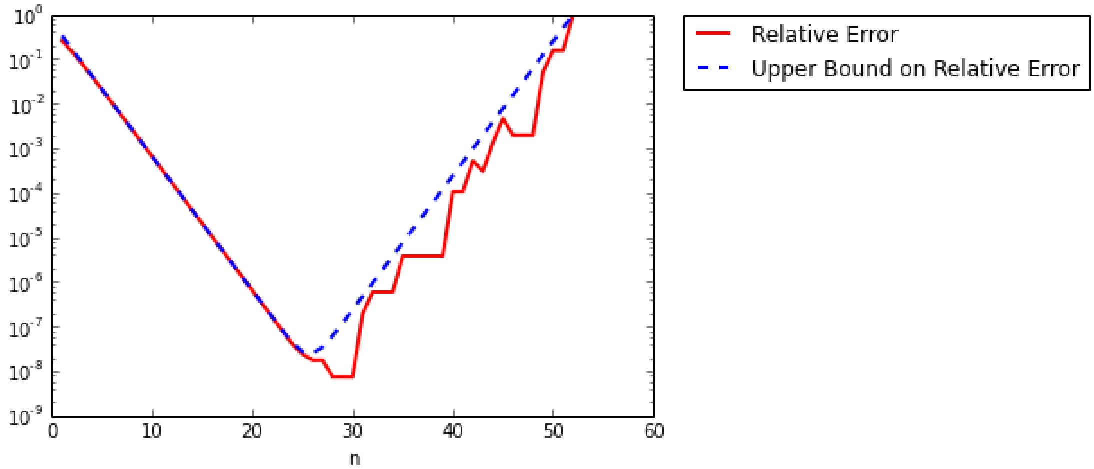
Now we can plot the data and get some insight in to what is going on. Again I used the csv library to quickly read in the text files. THen we plot r and R against n and h

In [5]:

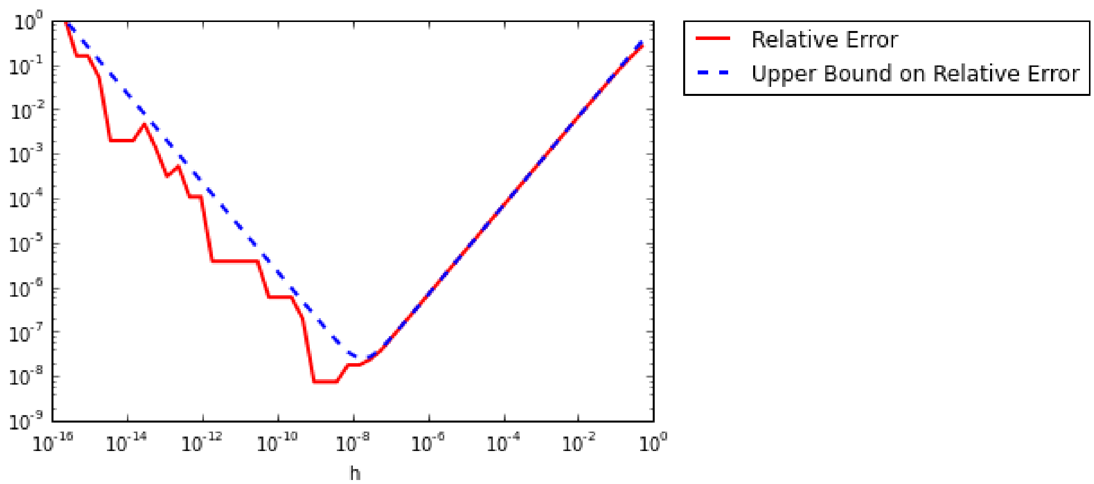
```
import csv

with open('n.txt', 'rb') as file:
    n = file.read().replace('\n', '').split('  ')
    n.remove('')
    file.close()

with open('h.txt', 'rb') as file:
    h = file.read().replace('\n', '').split('  ')
    h.remove('')
    file.close()

with open('r.txt', 'rb') as file:
    r = file.read().replace('\n', '').split('  ')
    r.remove('')
    file.close()

with open('R.txt', 'rb') as file:
    R = file.read().replace('\n', '').split('  ')
    R.remove('')
    file.close()
```

In [6]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np




fig = plt.figure()
ax = fig.add_subplot(1,1,1)


plt.xlabel( 'n'  )

line, = ax.semilogy(n, r, 'r-',label="Relative Error", lw=2)
line2, = ax.semilogy(n ,R, 'b--' , lw=2, label="Upper Bound on Relative Error")


plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)


plt.show()
```



In [16]:
```
import matplotlib.pyplot as plt
import numpy as np

plt.loglog(h,r,'r-', label="Relative Error", lw=2)
plt.loglog( h, R, 'b--', label="Upper Bound on Relative Error", lw=2)

plt.xlabel( 'h'  )
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



Simply from observing the graphs we notice an intresting behavior. Starting from the smallest n (or the largest n) we notice a smooth decline in both the relative error and the upper bound. This trend continues untill about half way through our spectrum of n (and h). Once we pass a crucial point, and continue towards the larger n values (and smaller h) we notice that the upper bound stays fairly smooth. However the relative error iteslf becomes a jagged line, darting aroudn and varying greatly. However the relative error never passes the upper bound, in accordance with the definition of an upper bound.The best approximation to the derivative will come from the point on the relative error line that is the lowest. according to the data set this point has the value of 7.450580652434979e-09. this value occurs at n = 30, or h = 9.313225746154785e-10.

There are two phenomena to describe and explain here. One is the decrease and then increase of both the relative error and upper bound. Secondly there is the smooth and then suddenly jagged relative error line. The first phenomena can be explained by looking at the function for r. the f'(a) term is present in both the numerator and the denominator. This term smooths the curve, brining the relative error and upper bound very much in line, untill a critical threshold is passed at a sufficently large n, or small h, wherein the f prime term is smaller in magnitude than the approximation term. Now as h decreases or n increases the relative error will approach the value of the approximation term. This explains why the two lines begin to differ, and why the relative error plot becomes jagged.

3 Makefile

```
################################################################################
# Project 1 Makefile
#
# Taylor Ellington
# Scientific HPC
# Fall 2015
################################################################################

# compiler & flags
CXX = g++
CXXFLAGS = -O -std=c++11
###################################

All : proj1_a.exe proj1_b.exe

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

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


matrix.o : matrix.cpp matrix.hpp
	$(CXX)$(CXXFLAGS) -c $< -o$@

nest.o : nest.cpp nest.hpp matrix.o
	$(CXX)$(CXXFLAGS) -c $< -o$@




clean :
	rm *.exe
	rm *.o
	rm *.txt
```