proj1_a

September 14, 2016

```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: import matplotlib.pyplot as plt
```

1 Approximation of a Function by Taylor Polynomials

To begin part 1, I implemented Horner's nested multiplication using C++. This reduced the number of arithmetic operations performed for each of our Taylor Polynomial approximations. I reduced

$$P = a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1} + a_n x^n$$

to

$$P = a_0 + (a_1 + x(a_2 + x^2(... + x^{n-1}(a_{n-1} + a_n x^n)...)))$$

This reduced the number of operations performed to n additions and n multiplications for every time an approximation is computed. Next, I created three varying degrees of a Taylor series representation of $f(x) = e^x$. The first Taylor polynomial was

$$T_4(x) = 1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3$$

The second Taylor polynomial was

$$T_8(x) = 1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7$$

The third and final Taylor polynomial was

$$T_{12}(x) = 1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7 + \frac{1}{40320}x^8 + \frac{1}{367880}x^9 + \frac{1}{3678800}x^{10} + \frac{1}{39916800}x$$

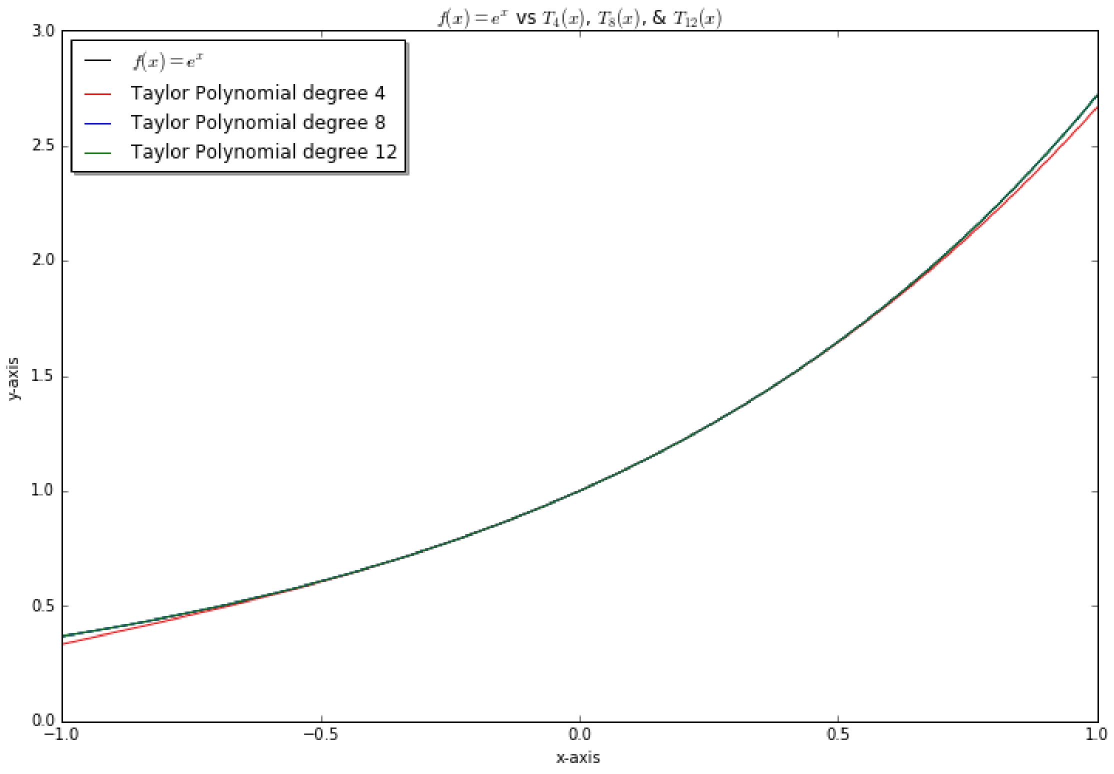
For each of these Taylor polynomials I approximated every x value in the vector z = -1, 10.99, -0.98, . . . , -0.01, 0, 0.01, . . . , 0.98, 0.99, 1.0 and created the vectors **p4, p8, p12** respectfully. They contained the approximation of $f(x) = e^x$ of each Taylor series for each x value in z. Next, I create **err4, err8, err12** that contain

$$|err_n - p_n|$$

or the absolute error of the approximation from the real value. The final portion of Part A required I write **p4, p8, p12, err4, err8, err12, & f** (f being the $f(x) = e^x$ for every x in z) to .txt files.

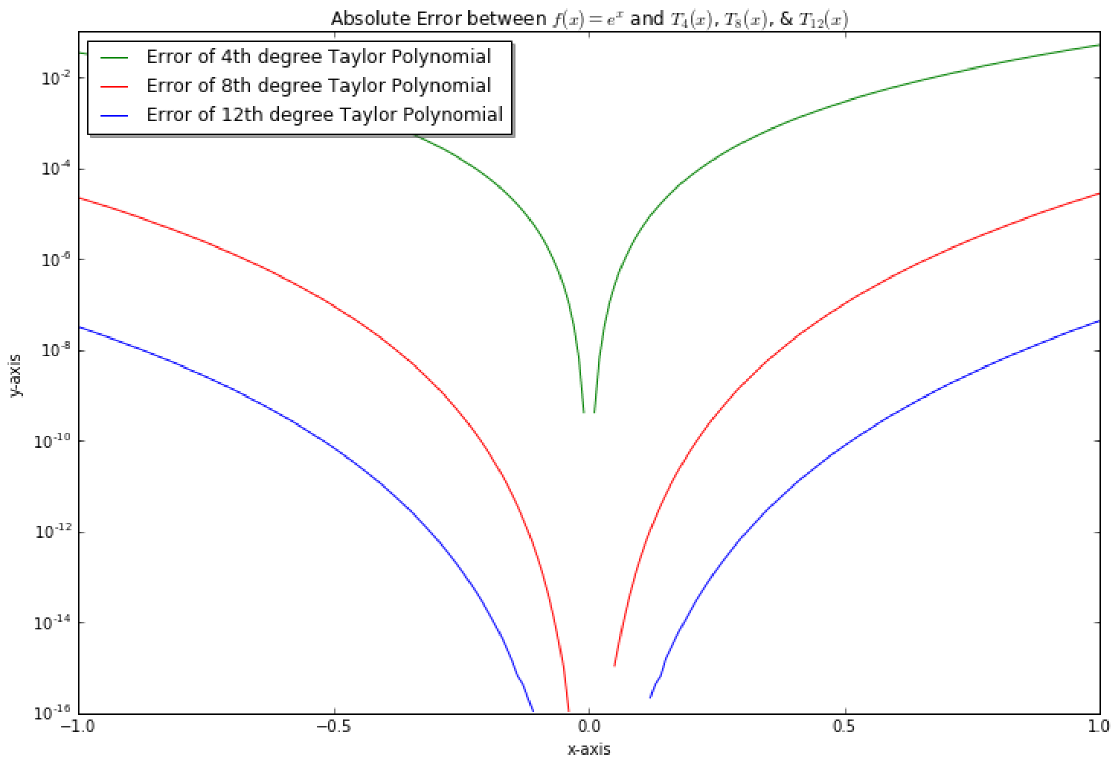
```
In [3]: z = open("z.txt",'r').read()
        z.replace(" ","")
        z = z.split("\n")
        z.remove("")

        p4 = open("p4.txt",'r').read()
        p4.replace(" ","")
        p4 = p4.split("\n")
        p4.remove("")

        p8 = open("p8.txt",'r').read()
        p8.replace(" ","")
        p8 = p8.split("\n")
        p8.remove("")

        p12 = open("p12.txt",'r').read()
        p12.replace("  ","")
        p12 = p12.split("\n")
        p12.remove("")

        fun = open("f.txt",'r').read()
        fun.replace("  ","")
        fun = fun.split("\n")
        fun.remove("")

        err4 = open("err4.txt",'r').read()
        err4.replace("  ","")
        err4 = err4.split("\n")
        err4.remove("")

        err8 = open("err8.txt",'r').read()
        err8.replace("  ","")
        err8 = err8.split("\n")
        err8.remove("")

        err12= open("err12.txt",'r').read()
        err12.replace("  ","")
        err12 = err12.split("\n")
        err12.remove("")

        for i in range(0,len(fun)-1):
            z[i] = float(z[i].strip())
            p4[i] = float(p4[i].strip())
            p8[i] = float(p8[i].strip())
            p12[i] = float(p12[i].strip())
            fun[i] = float(fun[i].strip())
            err4[i] = float(err4[i].strip())
            err8[i] = float(err8[i].strip())
            err12[i] = float(err12[i].strip())
```

```
In [4]: fig = plt.figure(figsize=(12,8))
        ax = fig.add_subplot(111)

        ax.plot(z,fun,"k-", label="$f(x)=e^x$")
        ax.plot(z,p4,"r-", label="Taylor Polynomial degree 4")
        ax.plot(z,p8,"b-", label="Taylor Polynomial degree 8")
        ax.plot(z,p12,"g-", label="Taylor Polynomial degree 12")
        ax.set_title("$f(x)=e^x$ vs $T_4(x)$, $T_8(x)$, & $T_{12}(x)$")
        ax.set_xlabel("x-axis")
        ax.set_ylabel("y-axis")

        legendPlot1 = ax.legend(loc="upper left", shadow=True)

        plt.show()
```



In the graph above we see two distinct lines. The first line we see is a combination of $f(x) = e^x$, Taylor Polynomial 8 and Taylor Polynomial 12. The second line that is visible is the red Taylor Polynomial 4. The reason this line is distinguishable from the other three is because its approximation of $f(x) = e^x$ is far less accurate than the 8th or 12th degree polynomial. This was expected because of the definition of a Taylor Series. The higher the degree of the Taylor polynomial, the higher degree of accuracy as values move away for the center point.

```
In [6]: fig = plt.figure(figsize=(12,8))
        ax = fig.add_subplot(111)

        ax.semilogy(z,err4,"g-", label="Error of 4th degree Taylor Polynomial")
        ax.semilogy(z,err8,"r-", label="Error of 8th degree Taylor Polynomial")
        ax.semilogy(z,err12,"b-", label="Error of 12th degree Taylor Polynomial")
        ax.set_title("Absolute Error between $f(x)=e^x$ and $T_4(x)$, $T_8(x)$, & $T_{12}(x)$")
        ax.set_xlabel("x-axis")
        ax.set_ylabel("y-axis")

        legendPlot2 = ax.legend(loc="upper left", shadow=True)

        plt.show()
```



The absolute error of the 4th degree polynomial becomes even more apparent when the error values are graphed. As you can see, the 8 and 12 degree polynomials have 0 or near 0 absolute error and are much better approximations of $f(x) = e^x$ as x moves away from the center point. This is also attributed to the arithmetic property of Taylor Polynomials. The higher degree of polynomial used will improve the degree of accuracy for values further from the center point.

The upper bound for the first Taylor series polynomial is E_4 which equates to $|e^4 - T_4(x)|$ for all $-1 < x < 1$. E_4 is equal to

$$|e - (1 + 1 + \frac{1}{2} + \frac{1}{6})|$$

or $E_4 = .0516151618$

For the second polynomial's upper bound E_8 evaluates to $|e - T_8(x)|$ for all $-1 < x < 1$ and is equivalent to

$$|e - 1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7|$$

or $E_8 = 2.7860205E^{-5}$

The final polynomial's upper bound is E_{12} evaluates to $|e - T_{12}(x)|$ for all $-1 < x < 1$. This evaluates to

$$|e - 1 + 1x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7| + \frac{1}{40320}x^8 + \frac{1}{367880}x^9 + \frac{1}{3678800}x^{10} + \frac{1}{39916800}x^{11}$$

or $E_{12} = 2.260473789E^{-9}$

This shows that the upper bound for the error in approximating $f(x) = e^x$ is best approximated by the Taylor series $T_{12}(x)$

1.0.1 proj1_a.cpp

```cpp
/* Project - Project 1: Part A
 * Professor - Dr. Xu
 * Date - 9/15/16
 * Author - Jake Rowland
 * Purpose - Find approximations of a funtion and the error in those approximations
*/

#include "matrix.hpp"
#include "Nest.cpp"
#include <cmath>

//Find approximations of e^x for three taylor polynomials and the error of these approximations
int main()
{
    //Create constant e
    const double e = 2.71828182845904523536;

    //Use linspace to create vector [-1, -.99, -.98, ..., -.01, 0, .01, ..., .98, .99, 1]
    Matrix z = Linspace(-1,1,201);

    //Create a 4 element vector for taylor polynomial conastants
    //and create vector for the approximation
    Matrix t4(4);
    Matrix p4(201);

    //Create a 8 element vector for taylor polynomial conastants
    //and create vector for the approximation
    Matrix t8(8);
    Matrix p8(201);

    //Create a 12 element vector for taylor polynomial conastants
    //and create vector for the approximation
```

```
Matrix t12(12);
Matrix p12(201);

//Create vector for actual e^x values
Matrix f(201);

//Create three vectors for the error of |f(x)-Pn(x)|
Matrix err4(201);
Matrix err8(201);
Matrix err12(201);

//Define the taylor polynomial for T4
t4(0,0) = 1;
t4(1) = 1;
t4(2) = (double)1/2;
t4(3) = (double)1/6;

//Define the taylor polynomial for T8
t8.Insert(t4,0,3,0,0);
t8(4) = (double)1/24;
t8(5) = (double)1/120;
t8(6) = (double)1/720;
t8(7) = (double)1/5040;

//Define the taylor polynomial for T12
t12.Insert(t8,0,7,0,0);
t12(8) = (double)1/40320;
t12(9) = (double)1/367880;
t12(10) = (double)1/3678800;
t12(11) = (double)1/39916800;

//For all elements in vector z
for(int i = 0; i < z.Rows(); i++)
{
    //Compute the approximation
    p4(i) = Nest::nest(t4,z(i));
    p8(i) = Nest::nest(t8,z(i));
    p12(i) = Nest::nest(t12,z(i));

    //Compute the actual answer
    f(i) = pow(e,z(i));

    //Compute the error in approximation
    err4(i) = std::abs((double)f(i)-(double)p4(i));
    err8(i) = std::abs((double)f(i)-(double)p8(i));
    err12(i) = std::abs((double)f(i)-(double)p12(i));
}
```

```
    //Write each of the vectors to a text file
    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");
}
```

1.0.2 Nest.cpp

```
/* Project - Project 1: Part A
 * Professor - Dr. Xu
 * Date - 9/15/16
 * Author - Jake Rowland
 * Purpose - Implement Horners algorithm
*/

#include <iostream>
#include "matrix.hpp"

class Nest
{
public:
    //Static method that implements Horners algorithm
    static double nest(Matrix& a, double x)
    {
        //Gets the size of the vector and stores it locally
        int size = a.Rows();
        //Gets the highest power coefficient first
        double poly = a(size -1);
        //Loops from the highest power to the lowest
        for(int i = size -2; i >= 0; i--)
        {
            //Add the new coefficient to the poly scaled by x
            poly = a(i) + (poly * x);
        }
        //return the result
        return poly;
    }
};
```

```
In [ ]:
```