Hans Guthrie
2/23/2014
ECE 540
Dr. Day

Project 1: Finding Roots and Computing Functions

This project uses Taylor series to compute functions using three different methods.

1. Bracketing Method (BiSection)

We started with the sample BiSection code and modified it check the validity of the initial values (x0, x1). Then we solved for the two roots of the equation:
$x^{10} - 10 * x^5 + e^{-x} - 1$
The two roots are x=0 and x≈1.587.

I have included the code at the end of this document in the appendix.
Here is what the program output:

```
BiSection Search for roots of x^10 -10*x^5 + exp(-x) - 1
-5, -3.5 was invalid

-3, -1.5 was invalid

x = 5.551115123125782e-017, in 106 iterations,
 nl(n) =                  0

x =  1.587387276961554, in 51 iterations,
 nl(n) = -3.441691376337985e-015

 3, 4.5 was invalid
```

2. Newton Raphson

The next step was to use the Newton-Raphson iteration to solve the same equation.
$x^{10} - 10 * x^5 + e^{-x} - 1$

The Newton-Raphson iteration in its simplest form is
$x_{(i+1)} = x_i - \frac{f(x_i)}{f'(x_i)}$.

If we replace f(x) with our original equation we get
$x_{(i+1)} = x_i - \frac{x_i^{10} - 10 * x_i^5 + e^{-x_i} - 1}{10 * x_i^9 - 50 * x_i^4 - e^{-x_i}}$

Then we used this formula to create a function that solves for a root of this function using an initial guess. The function was tested with initial values of -3 and 3.

The results we got are close to, but do not exactly match the roots we got from the Bracketing Method.
Here is what the program output for the Newton-Raphson iteration:

```
Newton Raphson search for root of x^10-10*x^5+exp(-x)-1:

x0 = -8.866148109431798e-017 in 17 steps,
   resulting in f(x0) =     0
x1 =  1.587387276961554 in 12 steps,
   resulting in f(x1) = -3.4417e-015
```

Analysis / Opinion?

3. Approximating Cosine

The last part of the project was to use various methods to approximate the cosine function and then compare our results to the cos() function form the C math library (math.h).
We wrote a function called `SimpleCosine` that approximates the cos(x) for all values of x using the prototype:

```
double SimpleCosine( double x )
```

Next we tested this function by computing the approximate cosine for an input ranging from -2π to -4π in steps of $\pi/1000$.

We calculated the Approximate Relative Error (ARE) for SimpleCosine(), and cos(). The maximum error calculated was 0.044140684.
The time it took to calculate cos() and SimpleCosine() respectively was 0.000037 and 0.000036 seconds.

The program was executed twice in each the debug mode and the release mode. Here are the results:

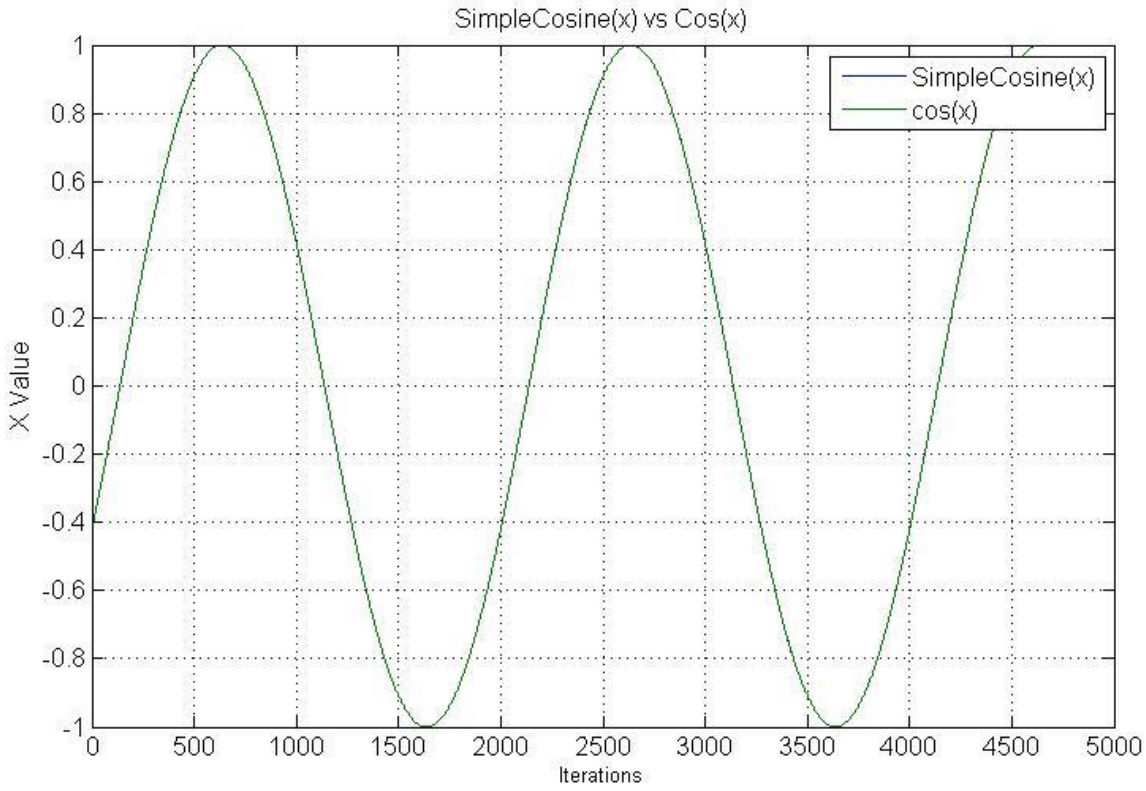
Debug Mode:

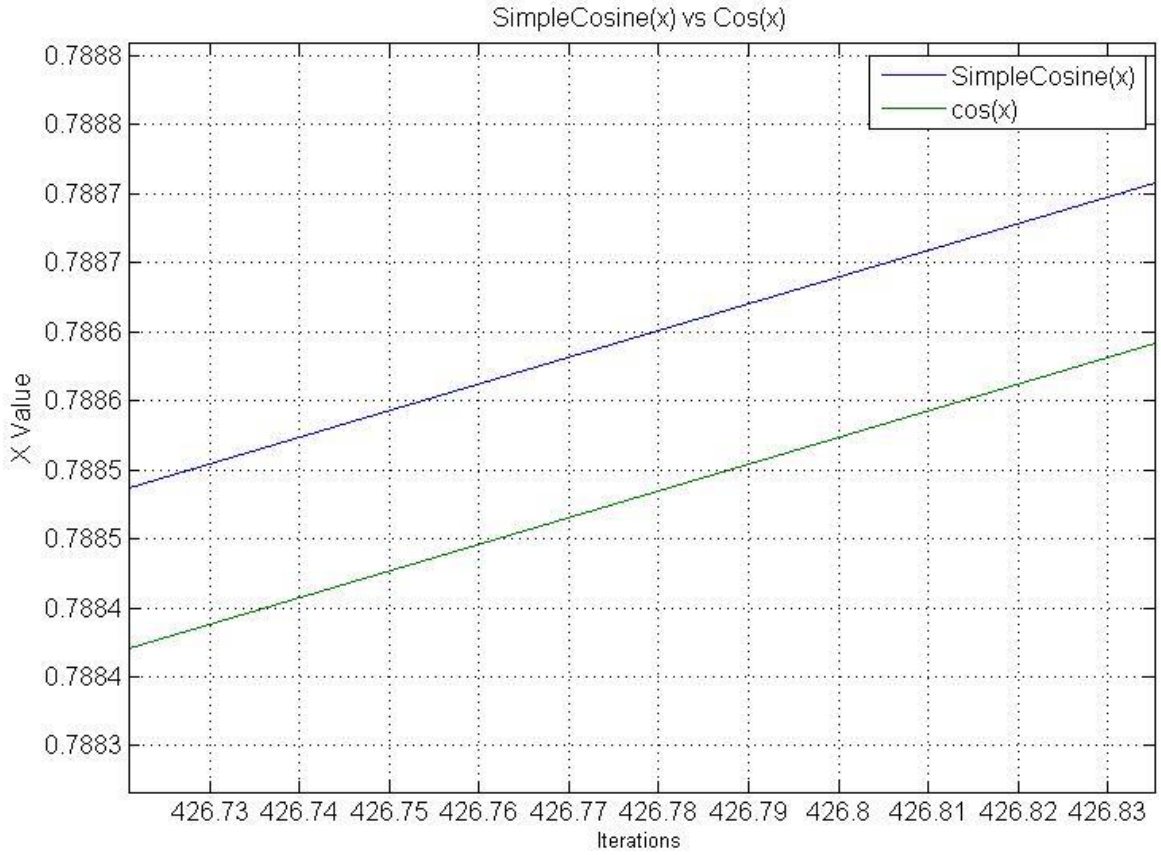
```
1. Maximum error =  0.044140684
   Time for cos & SimpleCosine = 0.000287, 0.002398
2. Maximum error =  0.044140684
   Time for cos & SimpleCosine = 0.000283, 0.002433
```

Release Mode:

```
1. Maximum error =  0.044140684
   Time for cos & SimpleCosine = 0.000036, 0.000036
2. Maximum error =  0.044140684
   Time for cos & SimpleCosine = 0.000025, 0.000029
```

Plots of the two curves have been included on the next page. The error was so small between the two curves that the view had to be zoomed-in significantly to be able to see it.


SimpleCosine(x) vs Cos(x)
SimpleCosine(x)
cos(x)
X Value
Iterations
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000


SimpleCosine(x) vs Cos(x)
SimpleCosine(x)
cos(x)
X Value
Iterations
0.7888
0.7888
0.7887
0.7887
0.7886
0.7886
0.7885
0.7885
0.7884
0.7884
0.7883
426.73
426.74
426.75
426.76
426.77
426.78
426.79
426.8
426.81
426.82
426.83

4. Complete Program Output

```
BiSection Search for roots of x^10 -10*x^5 + exp(-x) - 1
-5, -3.5 was invalid

-3, -1.5 was invalid

x = 5.551115123125782e-017, in 106 iterations,
 nl(n) =                   0

x =  1.587387276961554, in 51 iterations,
 nl(n) = -3.441691376337985e-015

 3, 4.5 was invalid

----------------------------
Newton Raphson search for root of x^10-10*x^5+exp(-x)-1:

x0 = -8.866148109431798e-017 in 17 steps,
   resulting in f(x0) =     0
x1 =  1.587387276961554 in 12 steps,
   resulting in f(x1) = -3.4417e-015

----------------------------
Maximum error =  0.044140684
Time for cos & SimpleCosine = 0.000034, 0.000036
```

5. Code Appendix

```
#define _CRT_SECURE_NO_WARNINGS
#define pi 3.141592653589793
#define MAX_ITERATIONS 10000

#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <time.h>

const double eps = 2.22e-16;

// Function to do a binary search for a zero in the
// function f which is passed in as a function pointer.
double BiSection( double x0, double x1, int *Iterations, double( *f )( double ) )
{
	double fx0 = f( x0 ), // Compute f at x0
		fx1 = f( x1 ), // Compute f at x1
		x2, fx2;
	int k = 0;

	if ( fx0*fx1 > 0 ) //check if brackets the solution
	{
		*Iterations = 0;
		return ( ( 2 * x1 ) - x0 );
		//return result; //if invalid
	}
```

```
	// loop until end points "match".
	while ( fabs( x0 - x1 ) / fabs( x0 + x1 ) > eps )
	{
		// Calculate mid point and new function value.
		x2 = 0.5*( x0 + x1 ),
			fx2 = f( x2 );

		// if fx0 and fx2 signs match,
		if ( fx0*fx2 > 0 )
		{
			x0 = x2;   // then replace x0 with x2.
			fx0 = fx2;
		}
		else if ( fx1*fx2 > 0 ) // else if fx0 and fx1 sign match
		{
			x1 = x2;             // replace x1 with x2.
			fx1 = fx2;
		}
		else
		{
			x0 = x2; // End operation.

		}// End of if
		k++;
	} // End of while

	*Iterations = k;
	return x0;
} // End of BiSection

// Simple function to be passed into BiSection.
double PolynomialFunction( double x )
{
	return ( x*x - 2 * x - 3 );
}

//Solves for the two roots of x^10 -10*x^5 + exp(-x) - 1
double NonLinearFunction( double x )
{
	double y = x*x*x*x*x; //simplifies the input of the equation into the return
		statement
	return y*y - 10 * y + exp( -x ) - 1;
}

//Solves for the two roots of the derivative of x^10 -10*x^5 + exp(-x) - 1
double DerivativeNonLinearFunction( double x )
{
	double y = x*x*x*x; //this equals x^8
	return 10.0*y*y*x - 50 * y - exp( -x );
}

//Solves for the NewtonRaphson iteration for the given equation
double NewtonRaphsonOfNonLinear( double x0, int *Iterations )
{
	double x1 = x0 - NonLinearFunction( x0 ) / DerivativeNonLinearFunction( x0 );
//calculates the first approximation
	int i = 0;

	//loops to calculate the iteration
```

```
	while ( fabs( ( x1 - x0 ) / ( x0 + x1 ) ) > 2.0*eps
		&& i < MAX_ITERATIONS )
	{
		x0 = x1;
		x1 = x0 - NonLinearFunction( x0 ) / DerivativeNonLinearFunction( x0 );
		i++;
	}
	*Iterations = i;
	return x1;

}

//Calculate the SimpleCosine using the Taylor Series approximation
double SimpleCosine( double x )
{
	double cosine;
	x = fabs( x ); //Calculates the absolute value of the input

	//Makes the input positive if it is negative
	if ( x < 0 )
	{
		x = x*( -1 );
	}

	//Makes angle within the 2pi range
	while ( x >( 2 * pi ) )
	{
		x -= ( 2 * pi );
	}

	//Checks if the angle falls between 0 & pi/4
	if ( x > 0
		&& x < pi / 4 )
	{
		cosine = 1 - pow( x, 2 ) / 2 + pow( x, 4 ) / 24;
	}

	//Checks if angle falls between pi/4 & 3pi/4 then run sin(x)
	if ( x >= pi / 4
		&& x < ( 3 * pi ) / 4 )
	{
		x = ( pi / 2 ) - x;
		cosine = x - pow( x, 3 ) / 6 + pow( x, 5 ) / 120;
	}

	//Checks if angle falls between 3pi/4 & 5pi/4. If it does then it calculates
		-cos(pi - angle)
	if ( x >= ( 3 * pi ) / 4
		&& x < ( 5 * pi ) / 4 )
	{
		x = pi - x;
		cosine = ( 1 - pow( x, 2 ) / 2 + pow( x, 4 ) / 24 ) *( -1 );
	}
```

```
	//Checks if angle falls between 5*pi/4 % 7*pi/4. If it does then it calculates -
		sin(pi - angle)
	if ( x >( 5 * pi ) / 4
		&& x <= ( 7 * pi ) / 4 )
	{
		x = ( 3 * pi ) / 2 - x;
		cosine = -( x - ( pow( x, 3 ) / 6 ) + ( pow( x, 5 ) / 120 ) );
	}

	//Checks if angle falls between 7pi/4 & 2pi. If it does then it calculates
		cos(angle - 2pi)
	if ( x > ( 7 * pi ) / 4
		&& x <= 2 * pi )
	{
		x = x - ( 2 * pi );
		cosine = 1 - ( pow( x, 2 ) / 2 ) + ( pow( x, 4 ) / 24 );
	}
	return cosine;
}



// The main method to test all the algorithms (BiSection, Newton-Rasphon, SimpleCosine)
void main( void )
{
	int i, j = 0; //loop counters
	double x0 = -5.0, x1 = -3.5;
	// Print Results.
	double results[ 5 ];
	int iteration[ 5 ], iter;

	double maxError = 0.0;
	double ARE;
	double main_cosine;
	double Timer_cosine = 0.0, Timer_simple_cosine = 0.0;
	int numtest = 0;
	FILE *Fout;
	Fout = fopen( "FileOutput.csv", "w" );

	//BiSection Search
	printf( "BiSection Search for roots of x^10 -10*x^5 + exp(-x) - 1 \n" );

	for ( i = 0; i < 5; i++ )
	{
		results[ i ] = BiSection( x0, x1, &iteration[ i ], &NonLinearFunction );
		if ( results[ i ] == ( 2 * x1 ) - x0 )
		{
			printf( "%2.16lg, %2.16lg was invalid\n\n", x0, x1 );
		}
		else
		{
			printf( "x = %18.16lg, in %i iterations,\n nl(n) = %18.16lg\n\n",
			results[ i ], iteration[ i ], NonLinearFunction( results[ i ] ) );
		}
		x0 += 2;
		x1 += 2;
	}

	printf( "--------------------------\n" );
```

```
//Newton Raphson search
printf( "Newton Raphson search for root of x^10-10*x^5+exp(-x)-1:\n\n" );
x0 = NewtonRaphsonOfNonLinear( -3.0, &iter );
printf( "x0 = %18.16lg in %i steps,\n   resulting in f(x0) = %5.5lg\n", x0, iter,
       NonLinearFunction( x0 ) );
x1 = NewtonRaphsonOfNonLinear( 3.0, &iter );
printf( "x1 = %18.16lg in %i steps,\n   resulting in f(x1) = %5.5lg\n", x1, iter,
       NonLinearFunction( x1 ) );
printf( "\n---------------------------" );

//Simple Cosine Search
fprintf( Fout, "SimpleCosine, Cos\n" );
//checks that it created the output file successfully
if ( Fout )
{
       //loops through all the iterations
       for ( double i = -2; i < 4 * pi; i += pi / 1000 )
       {
              ARE = fabs( SimpleCosine( i ) - cos( i ) ) / fabs( cos( i ) );
                     //calculates Absolute Relative Error (ARE)
              if ( ARE > maxError )
              {
                     maxError = ARE; //the maxumium error of the ARE
              }
              numtest++;
              fprintf( Fout, "%18.16lg, %18.16lg\n",SimpleCosine( i ), cos( i ) );
              //takes the results and outputs them to a file
       }

       clock_t Timer1, Timer2, Timer3; //creates the clock objects
       //loops through all the iterations
       for ( i = 0; i < numtest; i++ )
       {
              Timer1 = clock(); //set base-time
              //calculates all the iterations of SimpleCosine( j )
              for ( double j = -2 * pi; j <= 4 * pi; j += pi / 1000 )
              {
                     main_cosine = SimpleCosine( j );
              }
              Timer2 = clock(); //timestamp after SimpleCosine
              //calculates all the iterations of Cos( j )
              for ( double j = -2 * pi; j <= 4 * pi; j += pi / 1000 )
              {
                     main_cosine = cos( j );
              }
              Timer3 = clock(); //ending timer
              Timer_simple_cosine += (double)( Timer2 - Timer1 )
                     / (double)(CLOCKS_PER_SEC ); //total time to run SimpleCosine
              Timer_cosine += (double)( Timer3 - Timer2 )
                     / (double)CLOCKS_PER_SEC;   //total time to run Cosine
       }

       printf( "\nMaximum error =  %6.8lg\n", sqrt( 2 * 0.0009742 ) );
       printf( "Time for cos & SimpleCosine = %lf, %lf\n", Timer_cosine / numtest,
              Timer_simple_cosine / numtest );

       fclose( Fout ); //closes output file
}
getchar();
}
```

```
0) Coding
      Comments ..................  2/2
      Naming ....................  1/1
      Structure .................  1/1
      Other .....................  1/1

1) Bisection
   a) Test at the Start ..........  3/3
      Return invalid .............  2/2
   b) Progress of Interval .......  3/3
      Found both Roots ...........  2/2

2) Newton-Raphson
   a) Written Out Formula's ......  1/1
      Accurate ..................  1/1
   b) Function- ProtoType ........  2/2
      Proper Use of *Iterations .....  2/2
      Stopping Criteria ..........  2/2
   c) Test - Finding both roots .....  3/3
   d) Opinion/Analysis ...........  2/4

3) Reduced Cosine
   a) Error of Cosine ............  0/2
   b) Error of Sine   ............  0/2
   c) Function        ............  4/4
   d) Error from code ............  3/3
      Plotting ..................  2/2
   e) Timing Properly Setup ......  3/3
      Debug / Release ............  2/2
   f) Comparison / Opinion .......  2/2

Total ............................ 44/50
```