Differentiation

Mitchell Miller

Physics 240

This assignment explores the properties and applications numerical differentiation methods. There are three different methods explored in this lab: forward difference, central difference, and extrapolated difference. Each of these methods calculates the derivative of a function with simple approximations based on the definition of the derivative. The forward difference method finds the value of the function at the desired point and a small distance past it and subtracts the values, finding the slope. The central difference method finds the values of the function at a small distance before and after the desired point and finds the slope between them. Finally, the extrapolated difference method is essentially the same as the central difference method, however smaller steps before and after the desired value are used. Each of these methods of differentiation will be explored using simple and complex integrals in this assignment. Additionally, these methods will be applied to two different root – finding methods.

1. Introduction

Differentiation is heavily utilized in scientific calculations. Derivatives are used to solve countless problems and equations, but this can become very tedious or sometimes simply impossible to do by hand. Fortunately, there are several different methods for calculating derivatives numerically by making slope approximations. Again, doing these methods by hand would also be quite tedious, however, simple computer programs can be written to perform these methods with great speed and accuracy. All three methods increase in accuracy as the step size used is decreased. There is, however, a practical limit to this because when the step size becomes very small, round – off error becomes a significant problem. Each of these methods requires very simple code to implement and, since there are no loops required, is very fast. The two methods for root – finding explored in this lab are the Bisection method and the Newton – Raphson method. The bisection method is slow but very reliable. As long as one knows the region in which there is a sign change in the function, this method will always converge. Finally, the Newton – Raphson method is fast, but does not always converge.

2. Theory

This lab uses the forward, central, and extrapolated difference methods to find the derivatives of two functions:

$$f_1(x) = \cos x \;\; , \;\; f_2(x) = e^x \qquad (1)$$

These functions have simple analytic first and second derivatives as follows

$$f'_1(x) = -\sin x \;\; , \;\; f'_2(x) = e^x \qquad (2)$$
$$f''_1(x) = -\cos x \;\; , \;\; f''_2(x) = e^x \qquad (3)$$

Using the definition of a derivative[1], the forward, central, and extrapolated difference methods for numerical differentiation are found to be the following[2]:

$$f'_{FD}(x) \cong \frac{f(x+h)-f(x)}{h} \quad (4)$$

$$f'_{CD}(x) \cong \frac{f(x+h/2)-f(x-h/2)}{h} \quad (5)$$

$$f'_{ED}(x) \cong \frac{8}{3h}\left(f\left(x+\frac{h}{4}\right)-f\left(x-\frac{h}{4}\right) - \left(f\left(x+\frac{h}{2}\right) - f\left(x-\frac{h}{2}\right)\right)\right) \quad (6)$$

In each of these methods, *h* is the step size used to find another point to determine the slope. As *h* is decreased, the accuracy of the approximation is increased until reaching a point where round – off error becomes an issue. The forward difference method is by far the most simple, taking the definition of the derivative of a function and applying it with small values of *h* instead of a limit as *h* approaches zero. The central difference method is slightly more complicated, but still clearly a result of the definition. This method splits the step size on either side of the desired point and calculates the slope between those two points. Finally, the extrapolated difference method is the most complicated. This method combines the central difference method using quarter steps in either direction with the central method with half steps. By combining both these approximations, the linear and quadratic error terms are eliminated.
Next, this assignment explores a problem involving Newton's second law, which requires a second derivative. To solve this, the central difference method is applied twice. This results in the following equation:

$$f''(x) \cong \frac{f(x+h)+f(x-h)-2f(x)}{h^2} \quad (7)$$

This equation can potentially result in subtractive cancellation due to the large value of 2*f(x)* being subtracted from an equally large term, *f(x+h)+f(x-h)*.
The final section of this assignment explores root – finding methods. The first is the Bisection method, which works by taking a range within which the desired root exists[3]. The algorithm splits the range in half and tests whether or not the new value positive or negative[4]. It continues to split the region, converging to the point where the function changes signs, which is the location of the desired root. Newton's method is significantly faster, but also more complicated and can potentially diverge to infinity or become trapped in an infinite loop. This algorithm works by projecting a straight line tangent to the starting point[5]. It then takes the zero of this tangent line to be the next point of interest. This quickly converges to the desired zero of the function as long a good starting point is chosen. If a guess leads the algorithm to choose a point, such as a local max or min, with a nearly horizontal slope the method will quickly diverge to infinity[6]. This result can be prevented by including backtracking in the algorithm which test for divergence or an infinite loop and modifies the guess if either is detected.

3. Experimental Method

The first part of this lab seeks to find the derivative of the two functions in equation (1) using all three methods described. To do this, three separate functions were created. Each of the approximations described in equations (4) (5) and (6) were calculated for both functions. Next, the second derivative of the cosine function was found over four cycles using equation (7).

The last section of this lab seeks to solve for the energies of a particle in a square well. The solutions for this problem are described by the following equations:

$$f(x) = \sqrt{10 - E_B}\tan\sqrt{10 - E_B} - \sqrt{E_B} \qquad (even) \quad (8.a)$$

$$f(x) = \sqrt{10 - E_B}\cot\sqrt{10 - E_B} - \sqrt{E_B} \qquad (odd) \quad (9)$$

To solve these, the Bisection and Newton's method were applied.

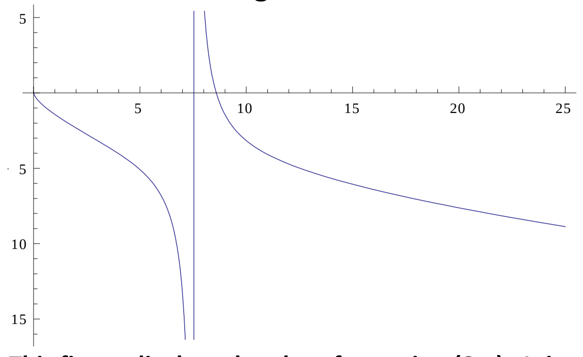
4. Data and Analysis

The first task in this assignment is to find the numerical derivative of equation (1) at 0.1, 1, 100. All three methods successfully find the derivative of the two functions. The results from each function at each point are shown in Figure 1-6. Every one of these figures displays the same trend. The accuracy of all three methods increases as the step size is decreased, until a point at which the effects of round – off error begin to cause the error to increase again. From these results, one can conclude that the ideal step size for the forward, central, and extrapolated difference methods are approximately 1E-8, 0.0001, and 0.01, respectively. For the cosine function, the extrapolate difference method results in the lowest error, but for the exponential function, the error for both the extrapolated and central difference methods have approximately the same minimum. The extrapolated difference method, however, does reach this error at a higher step size.

Next, the error in the second derivative was found. This error is shown in Figure 7. This figure displays the error of the second derivative of cos(x) between 0 and 4π in increments of π/10. This figure displays results similar to the previous figures for most points. There are some points, however, that result in significant error at all step sizes. This could be a result of the built in cosine function when programming or some other random error. It is possible that this is also the result of subtractive cancelling. The final, and most likely, possible cause of this error would be the fact that cos(n/2*π) is zero, so when calculating error, the result is divided by an extremely small number. This could easily prevent the error from reaching a reasonable error regardless of the step size used.

The final section of this lab seeks to find the even solutions to the energy states of a particle in a square well. The plot of equation (8) is shown in Figure A.

Figure A



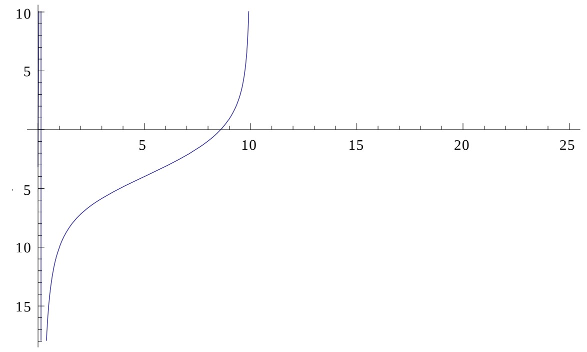
This figure displays the plot of equation (8.a). It is clear that there are discontinuities in it, making root – finding more difficult.

The singularities in this function can present a problem for the root – finding methods. To combat this, the book suggests an alternate form to equation (8) as follows:

$$f(x) = \sqrt{E_B}\cot\sqrt{10 - E_B} - \sqrt{10 - E_B} \quad (even) \ \ (8.b)$$

This function is plotted in Figure B.

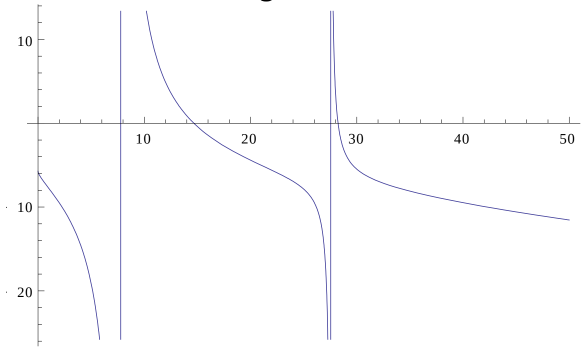
Figure B



This figure displays the plot of equation (8.b). It has identical solutions to equation (8.a), but the singularities are moved to different locations.

This function has the same roots as equation (8.a), with different singularities. In each of these equations, the *10* represents the depth of the potential well. Figure C displays the results when this value is increased.

Figure C



This figure displays the plot of equation (8.a) with the *10* replaced with *30*.

From analysis of Figure C, one can conclude that by increasing the well size, the number of energy states available increases.
The error for the zeroes found by the two methods is displayed in Table 1. Both methods found both roots with a great deal of accuracy (relative to Mathematica). Newton's method had slightly high error for the smaller root, but not significantly.

5. Conclusion

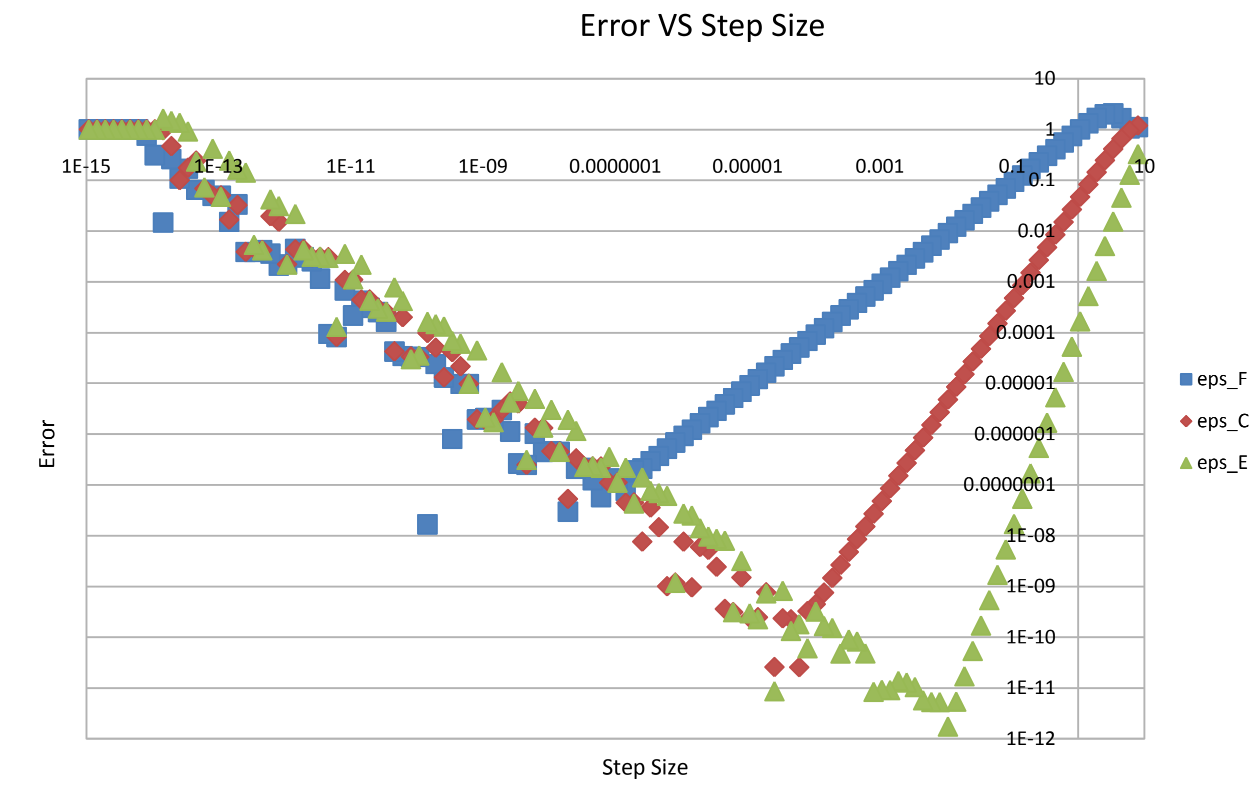
This lab successfully demonstrated the properties of numerical differentiation methods. First, the derivatives of the functions e^x and cos(x) were found using all three methods, the forward, central, and extrapolated difference. Next, the second derivative of cos(x) was found using the central difference method. Finally, the solutions to a quantum particle in a square well were found using the Bisection and Newton's Method.

6. Bibliography

[1]"Numerical Differentiation -- from Wolfram MathWorld." *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*. Web. 19 Mar. 2010. <http://mathworld.wolfram.com/NumericalDifferentiation.html>.
[2]"Numerical Differentiation." *Cal State Fullerton Web*. Web. 19 Mar. 2010. <http://math.fullerton.edu/mathews/n2003/NumericalDiffMod.html>.
[3]"Bisection Method -." *Wikipedia, the Free Encyclopedia*. Web. 19 Mar. 2010. <http://en.wikipedia.org/wiki/Bisection_method>.
[4]"The Bisection Method." *S.O.S. Math*. Web. 19 Mar. 2010. <http://www.sosmath.com/calculus/limcon/limcon07/limcon07.html>.
[5]"The Newton-Raphson Method." *S.O.S. Math*. Web. 18 Mar. 2010. <http://www.sosmath.com/calculus/diff/der07/der07.html>.
[6]"Newton's Method -- from Wolfram MathWorld." *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*. Web. 19 Mar. 2010. <http://mathworld.wolfram.com/NewtonsMethod.html>.

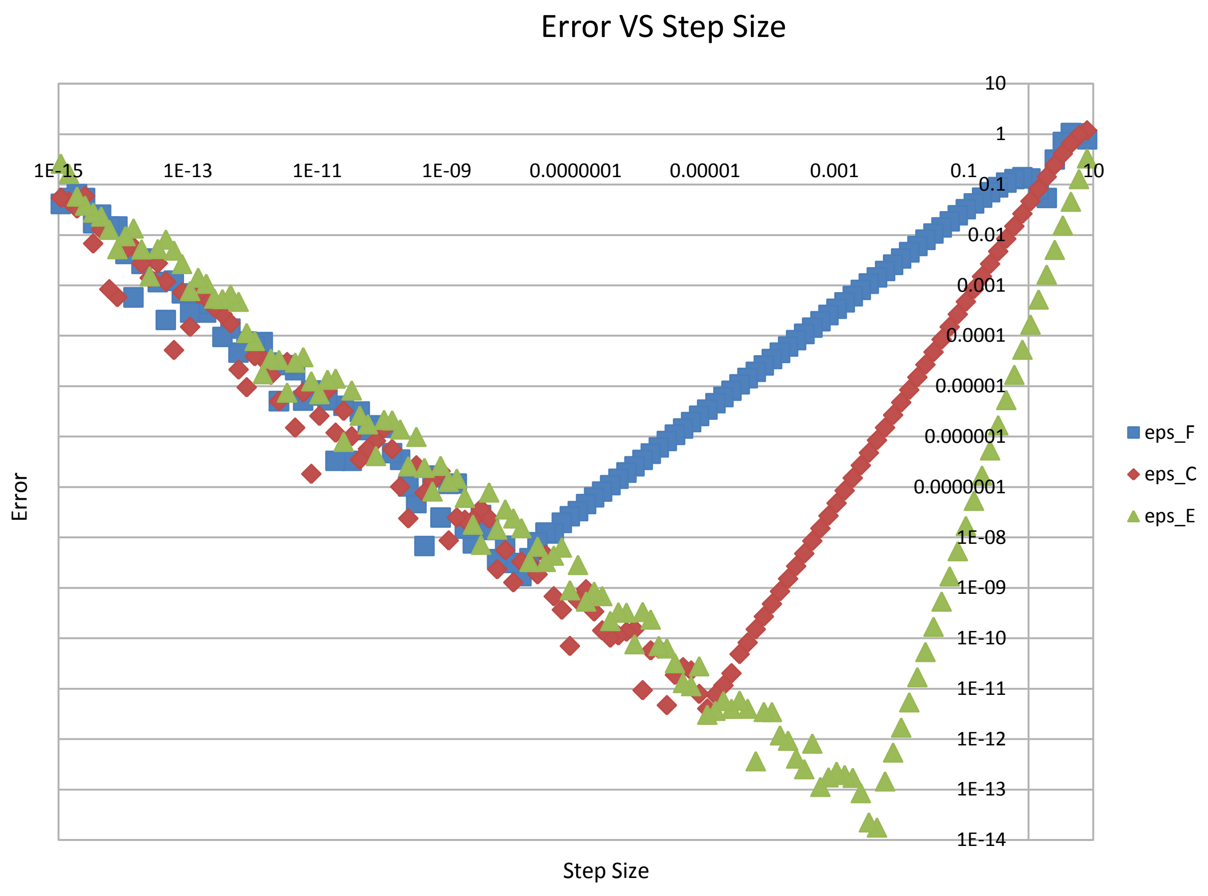
APPENDIX A: TABLES AND FIGURES

Figure 1



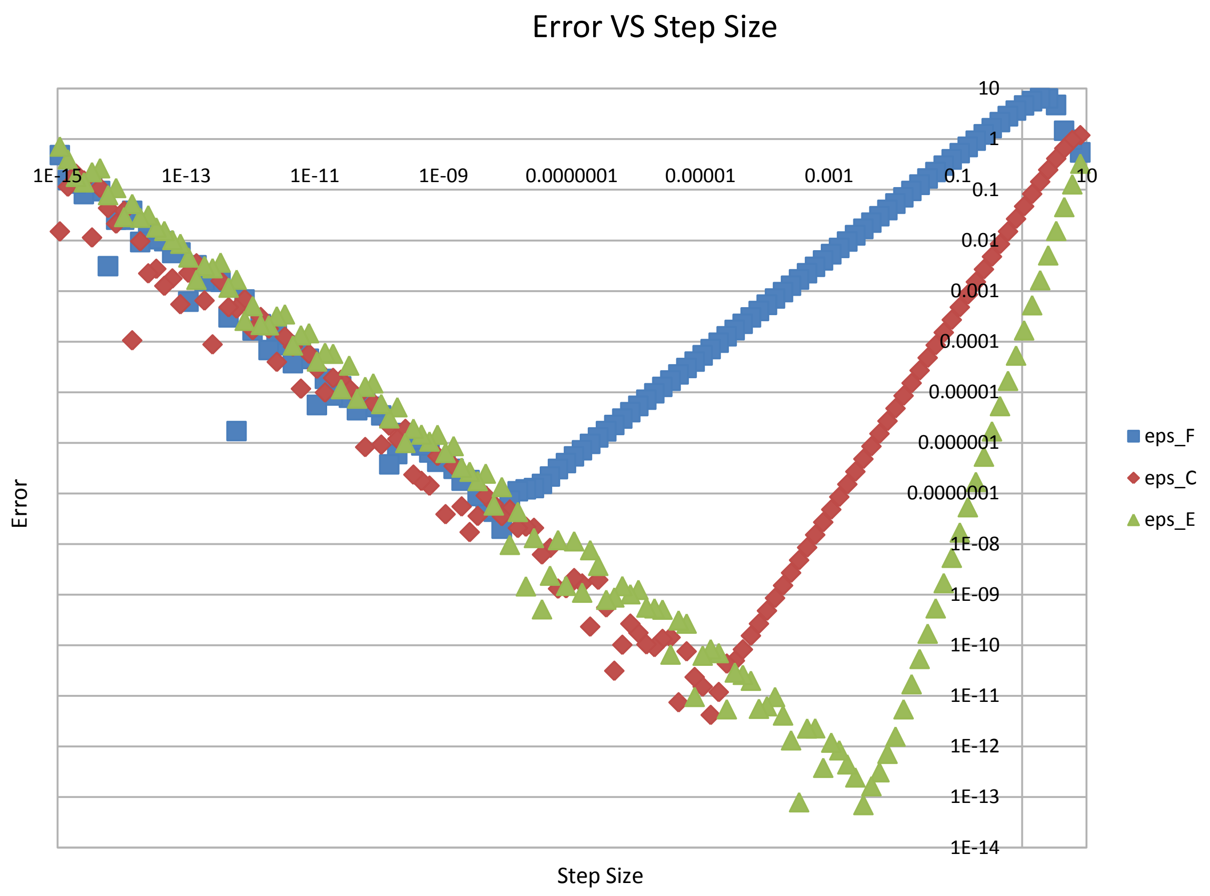
This figure displays the error for the derivative of cos(100). All three methods are plotted, showing the error versus the step size used. It is clear that the extrapolated difference method resulted in the highest accuracy

Figure 2



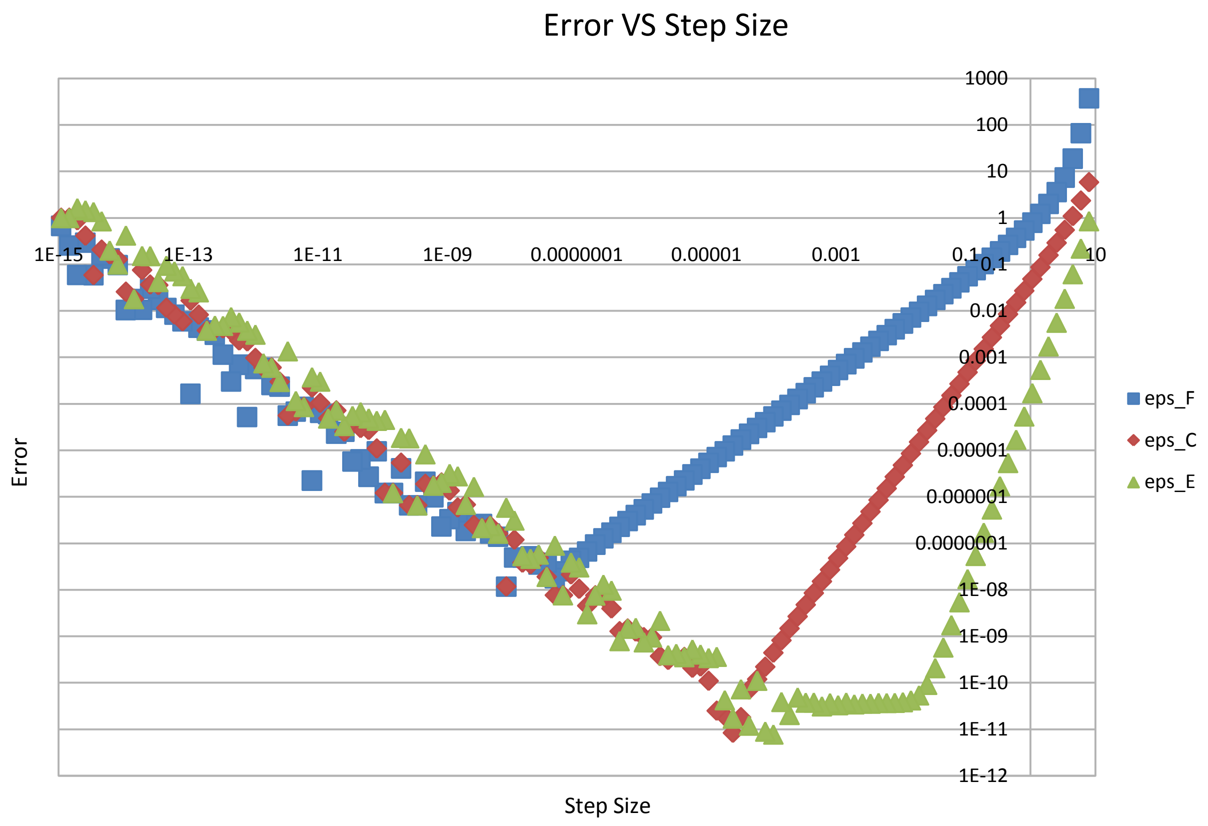
This figure displays the error for the derivative of cos(1). All three methods are plotted, showing the error versus the step size used. It is clear that the extrapolated difference method resulted in the highest accuracy

Figure 3



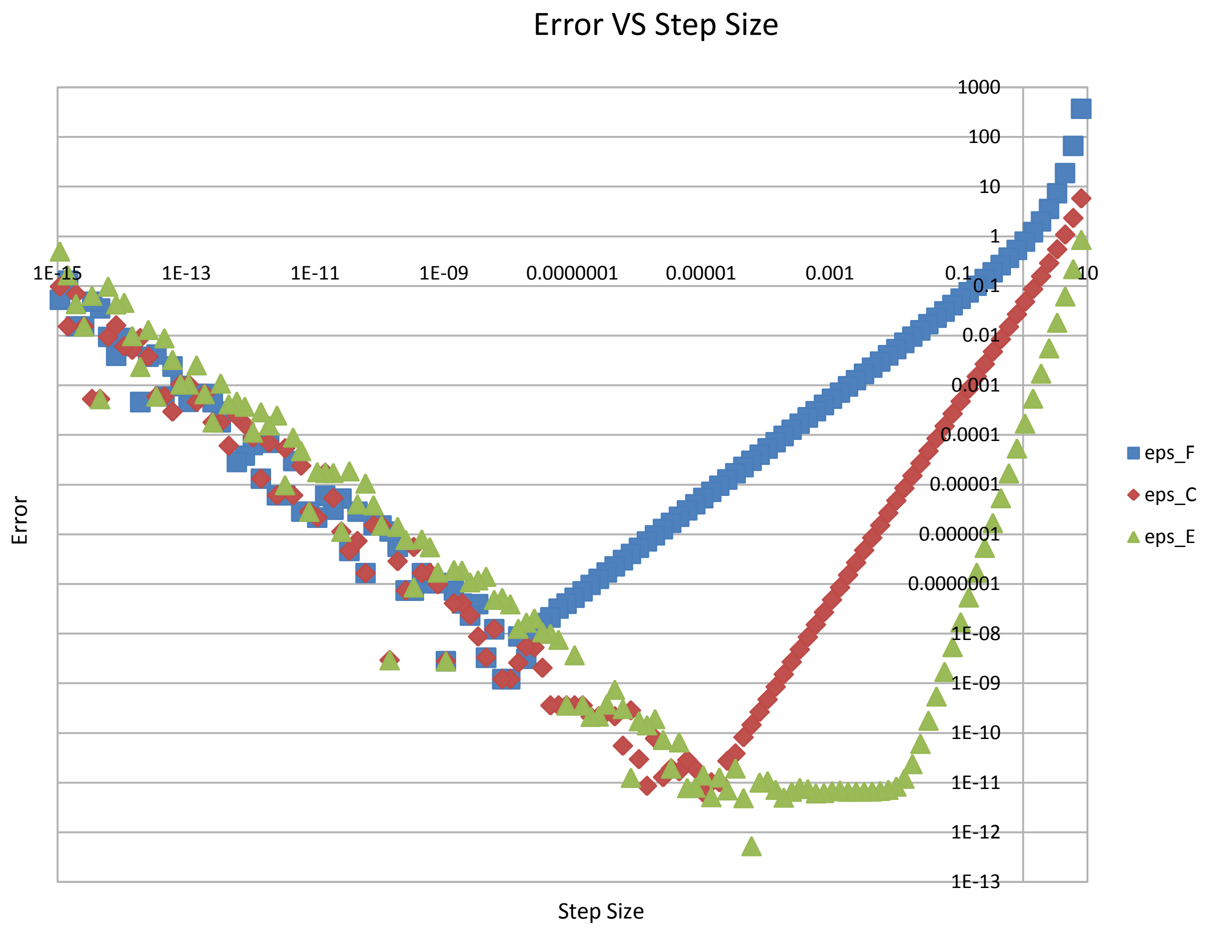
This figure displays the error for the derivative of cos(0.1). All three methods are plotted, showing the error versus the step size used. It is clear that the extrapolated difference method resulted in the highest accuracy

Figure 4



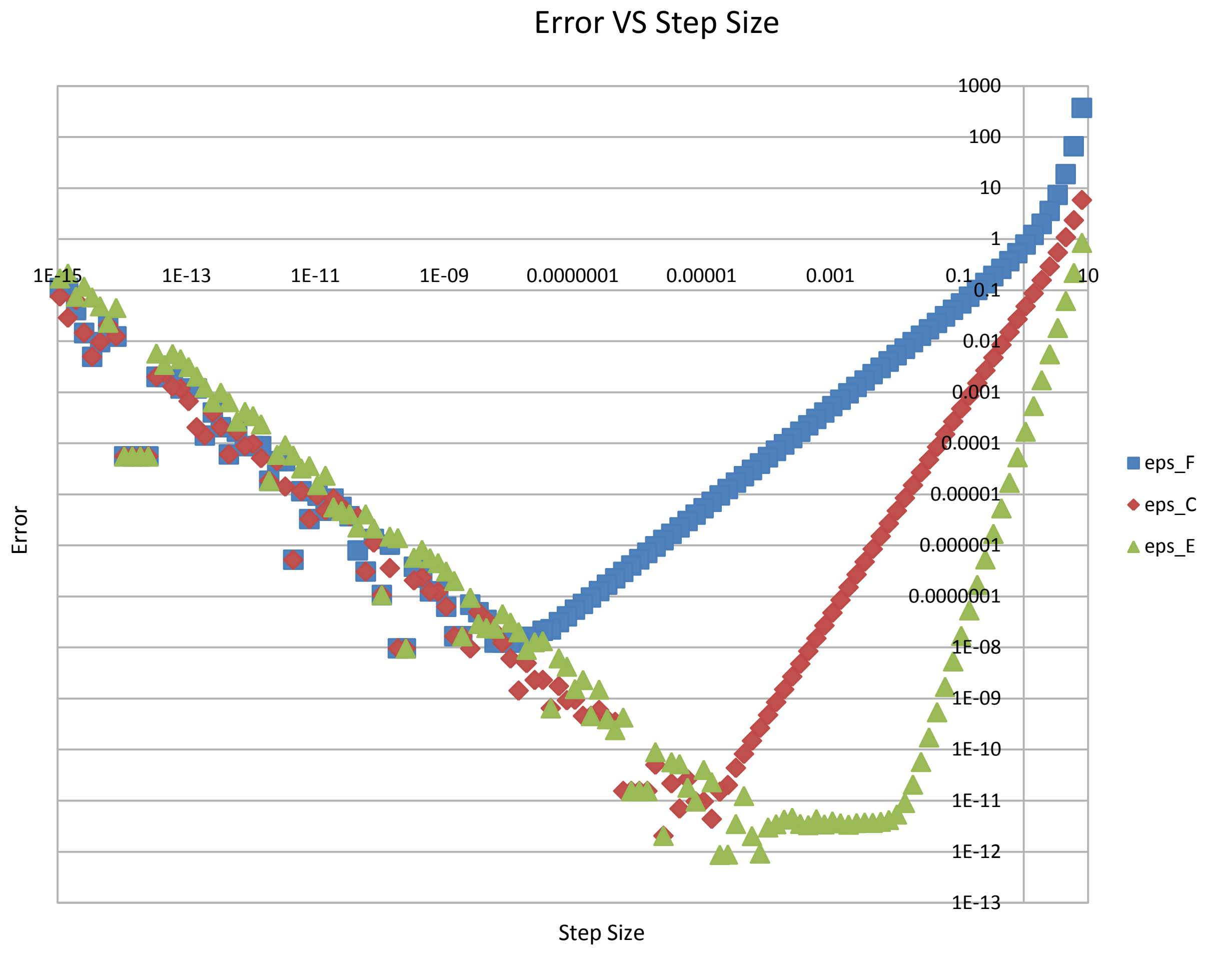
This figure displays the error for the derivative of e^(100). All three methods are plotted, showing the error versus the step size used. It is unclear whether the central or extrapolated difference method is more accurate; however the extrapolated difference method reaches the minimum fastest

Figure 5



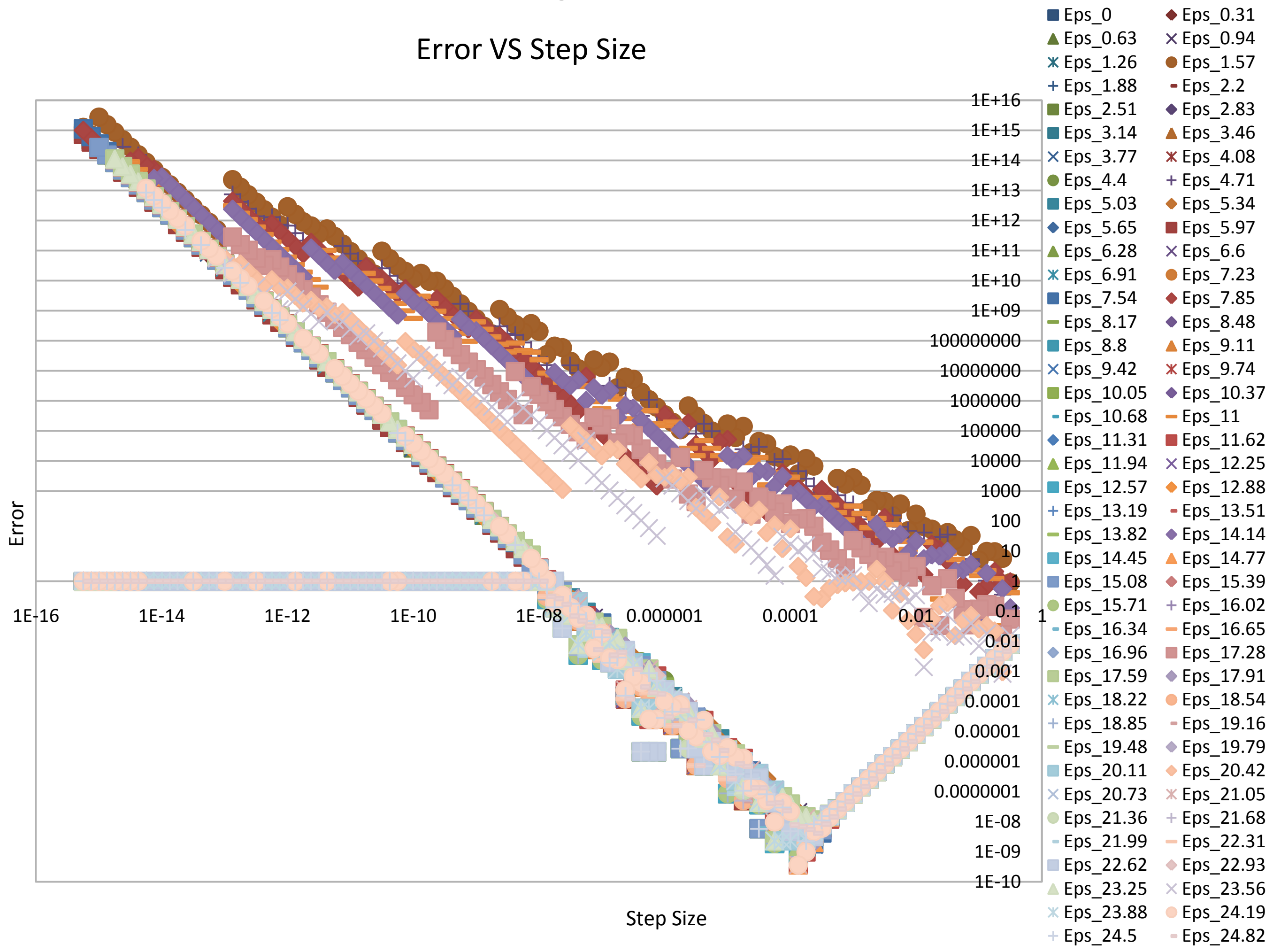
This figure displays the error for the derivative of e^(1). All three methods are plotted, showing the error versus the step size used. It is unclear whether the central or extrapolated difference method is more accurate; however the extrapolated difference method reaches the minimum fastest

Figure 6



This figure displays the error for the derivative of e^(0.1). All three methods are plotted, showing the error versus the step size used. It is unclear whether the central or extrapolated difference method is more accurate; however the extrapolated difference method reaches the minimum fastest

Figure 7



This figure displays the errors for four cycles of the second derivative of cos(x). Each value is plotted in a separate series. The cycles are divided into tenths, resulting in 40 points to differentiate

Table 1

Mathematica	Bisection	Newton	Bisection Error	Newton Error
0.00401926245332926000	0.00401926245332924000	0.00401926245332932000	5.17923E-15	1.48903E-14
8.59278527522983000000	8.59278527522983000000	8.59278527522983000000	0	0

This table shows the zeroes calculated by Mathematic, the Bisection, and Newton's Method for equation (8)

APPENDIX B: SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define e 2.71828182845

//Mitchell Miller
//Assignment 4
//PHYS 240 3/19/10

double function(double x){
	//Specify your desired function
	//return(pow(e,x));
	return(cos(x));
}

//Calculate the forward difference differential

//          f(x + h) - f(x)
// f'(x) = -----------------
//                 h

double forwardDiff(double xValue, double stepSize){
	double differentiatedValue;

	differentiatedValue = (function(xValue+stepSize) - function(xValue)) / stepSize;
	return(differentiatedValue);

}

//Calculate the central difference differential

//          f(x + h/2) - f(x - h/2)
// f'(x) = -----------------------
//                   h

double centralDiff(double xValue, double stepSize){
	double differentiatedValue;

	differentiatedValue = (function(xValue+stepSize/2)-function(xValue-stepSize/2)) /
stepSize;
	return(differentiatedValue);
```

```
}

//Calculate the extrapolated difference differential

//          f(x + h/4) - f(x - h/4)
// f'(x) = ------------------------
//                      h/2

double extrapolatedDiff(double xValue, double stepSize){
	double differentiatedValue;

	differentiatedValue = (8*(function(xValue+stepSize/4)-function(xValue-stepSize/4)) -
(function(xValue+stepSize/2)-function(xValue-stepSize/2))) / (3*stepSize);
	return(differentiatedValue);

}

int main(){
	//Initialize variables
	double loops;
	double forSolution,cenSolution,extSolution;
	double forError,cenError,extError;
	double analyticSolution = -sin(100);
	FILE *outfile;
	outfile=fopen("7.5.1.cos.error_10e1.txt","w");
	fprintf(outfile,"N\teps_F\teps_C\teps_E\n");

	//Calculate values
	for(loops=8;loops>=1e-15;loops=loops*0.75){
		forSolution = forwardDiff(100,loops);
		cenSolution = centralDiff(100,loops);
		extSolution = extrapolatedDiff(100,loops);

		//Calculate error
		forError = fabs((analyticSolution-forSolution) / analyticSolution);
		cenError = fabs((analyticSolution-cenSolution) / analyticSolution);
		extError = fabs((analyticSolution-extSolution) / analyticSolution);

		//Print results
		printf("h = %e\n", loops);
		printf("Forward Error = %e\n", forError);
		printf("Central Error = %e\n", cenError);
		printf("Extrapolated Error = %e\n", extError);
		fprintf(outfile,"%.20lf\t%.20lf\t%.20lf\t%.20lf\n",loops,forError,cenError,extError);
		}

		fclose(outfile);

}


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265358979323846264338327950288419716939937510582097494

//Mitchell Miller
//Assignment 4
//PHYS 240 3/19/10

double function(double x){
		//Specify your desired function
		return(cos(x));
}

//Calculate the central difference second order differential

//		f(x + h) + f(x - h) + 2f(x)
// f''(x) = ----------------------------
//			h * h

double centralDiff(double xValue, double stepSize){
		double differentiatedValue;

		differentiatedValue = (function(xValue+stepSize)+function(xValue-stepSize)-
2*function(xValue)) / (stepSize*stepSize);
		return(differentiatedValue);

}

int main(){
		//Initialize variables
		double loops;
		double cenSolution;
		double cenError;
		double xValue;
```

```
	FILE *outfile;
	outfile=fopen("7.6.1.error.txt","w");
	fprintf(outfile,"X Value\th\teps\n");

	//Calculate values
	for(xValue=0;xValue<=8*PI;xValue=xValue+PI/10){

		for(loops=PI/10;loops>=5e-16;loops=loops*0.75){
			cenSolution = centralDiff(xValue,loops);

			//Calculate error
			cenError = fabs((-cos(xValue)-cenSolution) / -cos(xValue));

			//Print results
			printf("h = %e\t", loops);
			printf("x = %lf\t", xValue);
			printf("Central Error = %e\n", cenError);
			fprintf(outfile,"%.20lf\t%.20lf\t%.20lf\n",xValue,loops,cenError);
		}

	}

	fclose(outfile);

}


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265358979323846264338327950288419716939937510582097494

//Mitchell Miller
//Assignment 4
//PHYS 240 3/19/10

double function(double x){
	//Specify your desired function
	return(sqrt(10-x)*tan(sqrt(10-x))-sqrt(x));
}

//Bisection method
double bisectionMethod(double xMin,double xMax){
	//Initialize variables
	double xMid=0,eps=1e-14;
	int counter;

	//Calculate zero
	for(counter=0;counter<100;counter++){

		//Find midpoint of segment
		xMid = (xMax + xMin) / 2.;
		printf("%d\t%.20lf\t%.20lf\n",counter,xMid,function(xMid));

		//Check sign of result
		if ( function(xMax)*function(xMid) > 0. )
			xMax = xMid;
		else xMin = xMid;

		//Test for convergence
		if ( fabs(function(xMid)) < eps){
		printf("Root = %.20lf\n",xMid);
		break;
		}

	}

}

//Newton-Raphson Algorithm using central difference approximation with backtracking
double newtonMethod(double xGuess){
	//Initialize variables
	double derivative,stepSize=1e-2,eps=1e-
14,functionAtX=function(xGuess),addedElement=stepSize;
	int counter,flag=0;

	//Calculate zero
	for(counter=0;counter<=100;counter++){

		flag=0;
		printf("%d\t%.20lf\t%.20lf\n",counter,xGuess,functionAtX);
		//Find central difference approximation
		derivative = (function(xGuess+stepSize/2)-function(xGuess-stepSize))/stepSize;
		//Change step size
		stepSize = -functionAtX/derivative;
		addedElement = stepSize;
		//Test for divergence and change guess
		while(flag==0){
        if(pow(fabs(function(xGuess+addedElement)),2)>pow(fabs(function(xGuess)),2)){
                        addedElement = addedElement/2;
                        printf("FAIL,%e\n",addedElement);}
                else{
                        xGuess+=addedElement;
                        flag=1;
                }
            }

            functionAtX=function(xGuess);
            flag=0;
            addedElement = stepSize;

            //Test for convergence
            if ( fabs(functionAtX)<eps){
                printf("Root = %.20lf\n",xGuess);
                break;
            }

        }

}

double main(){
        //bisectionMethod(8,9);
        newtonMethod(8.59278527522983814890);
}
```