AM 2814G: Lab 1B

Aidan Sirbu, Katie Brown
asirbu@uwo.ca, kbrow327@uwo.ca

Western University — January 31, 2021

Abstract

The purpose of this lab was to investigate the errors resulting from approximating a function with a Taylor Polynomial. A Taylor Series was constructed for the given function, as well as an expression which bounded the truncation error when a finite number of terms was used. The Matlab software was used to plot the Taylor Approximation and the original function over several ranges, as well as calculate the absolute and relative error. The Taylor Polynomial was a very good approximation for the function over a relatively large range, but the error increased significantly when evaluated over small ranges. To investigate the source of error, the truncation error bounds were plotted against the error. The resulting plot demonstrated that the source of the error must have been round-off. To correct for this and minimize error, a function was constructed which uses the Taylor approximation within a small range, and the function's exact value outside this range.

Introduction and Background

The Taylor Series is an important mathematical tool used for numerically evaluating certain functions and approximating solutions to complex problems. The Taylor Polynomial of a function *f(x)* is a partial (finite) sum of the first *n* terms of the Taylor Series, given by

$$T_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

Taylor Polynomials provide an approximation of a function about a point x_0, which generally becomes more accurate as the number of terms increases. When a finite number of terms is used, there will be an associated truncation error, which describes how much the Taylor Polynomial approximation differs from the real value of the function at x_0.

Matlab represents floating point numbers in double precision form. This format uses 64 bits to define a number: a sign, a Mantissa of length 52, and an exponent of length 11. As all numbers must be represented in a finite number of bits, some precision can be lost when using this format. Looking at the Machine Epsilon, ε_{mach}, allows us to understand Matlab's inherent precision. Machine Epsilon is the smallest possible increment; it is the distance between 1 and the smallest number greater than 1. For the double precision floating point standard set by the Institute of Electrical and Electronics Engineers, $\varepsilon_{mach} = 2^{-52}$.

Problem 1

We need to derive the Taylor Series with *n+1* terms and the associated truncation error for the function

$$f(x) = \frac{\log(x) - x + 1}{(x-1)^2}. \tag{1}$$

We assume $f(x)$ is $n+1$ times continuously differentiable so that we may take advantage of *Taylor's Theorem with Remainder* later in the derivation. To begin, we write the expanded form of

$$\log(x) = \sum_{k=1}^{n+2} \frac{(-1)^{k-1}}{k}(x-1)^k. \tag{2}$$

We choose to take the sum from *k=1* to *k=n+2* to reduce the number of terms in our final polynomial through algebraic manipulation and as our target polynomial must have *n+1* terms:

$$\log(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} \pm ... \pm \frac{(x-1)^n}{n} \pm \frac{(x-1)^{n+1}}{n+1} \pm \frac{(x-1)^{n+2}}{n+2} \tag{3}$$

From here we merely algebraically manipulate the left-hand side of the equation to match the given function *f(x)*. Whatever manipulations are done to the left-hand side must also be done to the right.

$$\begin{aligned}
\log(x) - x &= -1 - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} \pm ... \pm \frac{(x-1)^n}{n} \pm \frac{(x-1)^{n+1}}{n+1} \pm \frac{(x-1)^{n+2}}{n+2} \\
\log(x) - x + 1 &= -\frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} \pm ... \pm \frac{(x-1)^n}{n} \pm \frac{(x-1)^{n+1}}{n+1} \pm \frac{(x-1)^{n+2}}{n+2} \\
\frac{\log(x) - x + 1}{(x-1)^2} &= -\frac{1}{2} + \frac{(x-1)}{3} - \frac{(x-1)^2}{4} \pm ... \pm \frac{(x-1)^{n-2}}{n} \pm \frac{(x-1)^{n-1}}{n+1} \pm \frac{(x-1)^n}{n+2}
\end{aligned} \tag{4}$$

Now we have the expanded form of *f(x)*, denoted as the degree n Taylor Polynomial of *f(x)* consisting of *n+1* terms. We then collapse equation (4) into summation notation to yield:

$$T_n(x) = \sum_{k=0}^{n} \frac{(-1)^{k+1}}{k+2}(x-1)^k. \tag{5}$$

The above equation represents the Taylor series with *n+1* terms. According to *Taylor's Theorem with Remainder* [1] the associated truncation error can be found by subtracting the degree n Taylor Polynomial from *f(x)* to yield the Taylor Remainder denoted as $R_n(x)$

$$R_n(x) = f(x) - T_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x - x_0)^{n+1} \tag{6}$$

for some c between x and x_0. This represents the associated truncation error.

Now we must construct an expression that bounds the truncation error, assuming $n > 2$, for a given value of *x*. If $f(x)$ has $n+1$ derivatives in the interval $|x - x_0| \leq r$ and for $|f^{(n+1)}(x)| \leq M$, then equation (6) may be bounded such that [2]

$$|R_n(x)| \leq \frac{M}{(n+1)!}|x-1|^{n+1} \text{ for } |x-1| \leq r. \tag{7}$$

Problem 2

Firstly, we modify the M-file from Lab1A to suit the Taylor Series derived above, in equation (5). This defines the function lab1bP2(x,n), which uses a *for* loop to build the Taylor Series *T* for *f(x)*:

```
for k = 0:n
    T = T + (-1).^(k+1)./(k+2)*(x-1).^k;
```

When defining the function, it is necessary to modify the standard mathematical operators into vector operators with '.', as we will be using a Linspace array for *x*.

We are able to initialize the sum at 0 due to the form of equation (5); we collapse the Taylor Polynomial into a series that starts at $k = 0$, so that the first term is be $-\frac{1}{2}$.

```
function T = lab1bP2(x, n)
    T = 0;
```

Note that the complete code of file lab1bP2.m and a description of the function parameters can be found in the Code Appendix.

Once the Taylor Series *T* is defined, it is plotted against the actual function *f(x)* to determine and graphically illustrate its accuracy. Firstly, *x* is defined to be a Linspace array of 50 points over the range 0.999 to 1.001:

```
x=linspace(0.999,1.001,50);
```

Next, *T* was set as the Taylor Series of equation (5) using the function lab1bP2(x,n). We choose n to be 3, as we are looking for the Taylor Series up to the 4th (cubic) term and our Series begins at the zeroth term:

```
T=lab1bP2(x,3);
```

Finally, *T* are *f(x)* are both plotted against *x*:

```
plot(x,T,x,(log(x)-x+1)./(x-1).^2)
```

The resulting plot is Figure 1(a) in the Plot Appendix. In this plot, the curves for *f(x)* and *T* are essentially indistinguishable. This demonstrates that over the range 0.999 to 1.001, The Taylor Series up to the cubic term is a very good approximation for *f(x)*.

Problem 3

Firstly, we are asked to change the range of the plot from Question 2 to $(1 - 10^{-7}, 1 + 10^{-7})$. The result is shown in Figure 1(b) and the code in Listing 3 in their respective appendices. Then we are asked to modify the script such that it plots the relative and absolute errors of function lab1bP2(x,n). The results are shown in Figures 2(a) and 2(b).

The analysis now consists of investigating the source of error. We should compare the error against a reasonable bound for the truncation error of the Taylor series $T_3(x)$. We use a modified version of the truncation error bound derived in equation (7). Define S as the sum of partial sum S_n and the infinite remainder R_n such that:

$$S = S_n + R_n \tag{8}$$

In this case, however, we have employed the use of Taylor series up to the 4th term which occurs when $n = 3$. Therefore, equation (8) becomes:

$$S = S_3 + R_3 \tag{9}$$

Attention must be now brought to the fact that our Taylor series for $f(x)$ is an alternating series so long as $x > 1$. This allows us to take advantage of the *Error Estimation Theorem for Alternating Series*. This theorem states

$$|R_n| = |S - S_n| = |Error| \tag{10}$$

such that the remainder/error must not be greater than the first term in the infinite remainder. Since we are taking the Taylor polynomial of degree 3 (which includes the 4 terms), we can write:

$$|\frac{\log x - x + 1}{(x-1)^2} - T_3(x)| = |R_3(x)|. \tag{11}$$

Taking the first term in the infinite remainder $R_3(x)$, by the *Error Estimation Theorem for Alternating Series* [3,4], we can bound our error such that:

$$|Error| \leq |-\frac{(x-1)^4}{6}|. \tag{12}$$

Now that we have bounded the truncation error for $x > 1$, however, a challenge arises when searching for a bound for $x < 1$. The entire premise of the bound in equation (12) relies on the Error Estimation for *Alternating* series, however for $x < 1$ each positive term in the series becomes negative as each have the form:

$$\frac{(x-1)^a}{b} \tag{13}$$

where a and b are odd integers. Firstly we take another look at equation (7). This bounded error was derived using the *Taylor's Theorem with Remainder* which does not rely on the notion that the series is alternating. Next, we are tasked with finding a bound *M* such that $f^{(n+1)}(x) \leq M$, or in this case where $n = 3$, $f^{(4)}(x) \leq M$. However, taking the 4th derivative of *f(x)* poses complications as it is a very complex derivative. Instead, we shall take $f(x) = \frac{1}{(x-1)^2}$ since the denominator dominates equation (1) as $x \to 0^+$ because polynomials always dominate logarithms. Taking the 4th derivative of *f(x)* yields:

$$f^{(4)}(x) = \frac{120}{(x-1)^6}. \tag{14}$$

Now, we must find an upper bound for this function in the range *x<1*. This poses a challenge as the function contains as asymptote at *x=1*. However, we can take advantage of matlab's inherent precision known as Machine Epsilon (ε_{mach}). The largest value equation (14) may possess is when the denominator is as small as possible. This occurs when $x \approx 1$ so we choose $x = 1 - \varepsilon_{mach}$. Equation (14) then becomes:

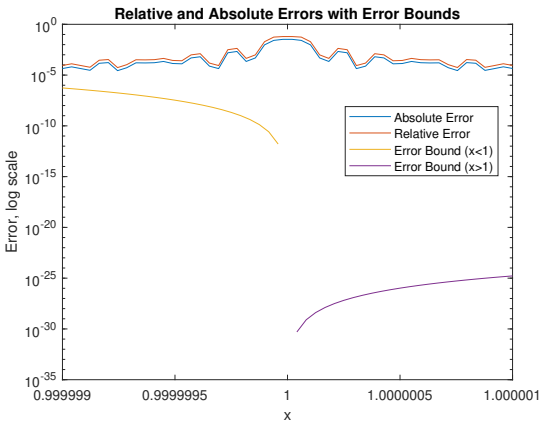
$$f^{(4)}(1-\varepsilon_{mach}) = \frac{120}{(-\varepsilon_{mach})^6} \tag{15}$$

Due to matlab's limitation of understanding values less than ε_{mach}, we get rid of the exponent 6 as this would cause matlab to work with floating point values less than its limitation, yielding:

$$M = \frac{120}{\varepsilon_{mach}} \implies |R_3(x)| \leq \frac{120}{(4!)(\varepsilon_{mach})}|x-1|^4 \tag{16}$$

We now have obtained a bound for error when $x < 1$.

Considering the results shown in the figure to the right, we conclude that the largest source of error is roundoff, rather than truncation error. This is, because the truncation error bounds are significantly lower than the computed absolute and relative errors. The error bound for $x > 1$ is about 25 orders of magnitude too small, while the error bound for $x < 1$ is only around 5 orders of magnitude too small. While one could make a stronger case for roundoff error being the main source for range $x > 1$, it also plays a factor in the range $x < 1$. One reason on why the roundoff errors are so large is possibly due to matlab's limitations in finite precision. As the Taylor series is computed using a *for* loop, consecutive iterations lose precision as decimal points are rounded off using the Rounding to Nearest Rule. This implies that the directly computed function is more accurate. This is to be expected as an exact value shall always be more accurate than a value subject to errors due to matlab's inherent precision limitations.



(Problem 3c) Relative and Absolute Error with Truncation Error bounds

Problem 4

We now want to create a matlab function that can evaluate equation (1) to a given accuracy for any input value x. This function will be named *myfofx(x)* where x is the input. This function will need to be able to handle input in the form of an integer/floating point value as well as in the form of an array of integers or floating point value. To begin this function,

```
function [fx] = myfofx(x)
    T = 0;
```

we declare the function name as the header. Furthermore, the value *T* will be our summed Taylor Series approximation, initialized as zero. This code must evaluate the input using a Taylor Series approximation if the input falls within the range $1 - 10^{-6} < x < 1 + 10^{-6}$. Should the input fall out of that range, we must use the exact value (*i.e.* use the function defined as *f(x)* in equation (1) to evaluate).

The first challenge we faced was handling array input. To evaluate using two different methods (approximation or exact) we must employ an *if-else* block statement. The *if* statement must contain a parameter which checks if the input lies within our target range. However, simply using a boolean check such as *if* $x < 1 + 10^{-6}$ would cause matlab to throw an error. Through further analysis, it was found that when the input was an array (in our case, a 1x50 array) it attempted to compare an array against a floating point value. To append this mistake, we take the first and last entry of the array and assign it to variables to use in the if statement:

```
    first = x(1);
    last = x(length(x));
```

Variable *first* was assigned to the first index inside of the array while *last* was assigned to the last index by employing the built-in *length()* method. Since matlab treats lone values as 1x1 arrays, should the user input such a value, both its first and last index is the value accounting for non-array entries as well.

To continue, an *if-else* block was used:

```
    if (last<1+10^-6) && (first>1-10^-6)
        ...
    else
        fx = (log(x)-x+1)./(x-1).^2;
    end
```

The function checks whether the array lies within the given range. If this condition should be satisfied, the code executed under the *if* statement can be found in the code appendix and an explanation can be found in Problem 2, as the code is identical. Should the condition not be satisfied, the *else* statement is executed, evaluating the input using the exact function $f(x)$.

The functions correctness was validated by plotting it over the same ranges as problems 2b and 3a (*shown in Figure 1*). The results were as expected. The plots, as shown in Figure 3, matched the plots of Figure 1 perfectly. Figure 3(a) plots the results over a range $(0.999, 1.001)$. Since this range is outside the target bounds for the Taylor approximation, $myfofx(x)$ used the exact value of $f(x)$. This produced a plot identical to that of Figure 1a. Furthermore, Figure 3(b) shows plots the results over a range of $(1 - 10^{-7}, 1 + 10^{-7})$. Since this lies within our target range, $myfofx(x)$ used the Taylor approximation. The result was plotted against that of the exact function $f(x)$ over the same range to stress that it did, in fact, use the Taylor approximation as it should. Overall, the function $myfofx(x)$ worked as expected when compared to Figure 1.

Summary of Results

Over the course of this investigation, the Taylor series of equation (1) was analyzed in a variety of ways. First, the Taylor series expansion was derived from the Taylor polynomial of $log(x)$. The series was then collapsed into sigma notation to allow for a better transition into Matlab code. Using *Taylor's Theorem with Remainder*, we were able to bound the truncation error of the degree *n* Taylor polynomial. This proved to be of use in further analysis.

We were then tasked to engineer a Matlab function which would allow for the calculation of the Taylor polynomial approximation centered at $x = 1$ of any *x* and for any number of terms. The resulting plots agreed with the expected outcome. Over a range of $(0.999, 1.001)$, the plots of $f(x)$ and $T_3(x)$ were essentially indistinguishable. However, adjusting the range to $(1 - 10^-7, 1 + 10^-7)$ yielded significant difference between the approximation and the exact value. This was expected as over a such a small range, the Taylor approximation falls victim to error, whether it be truncation or roundoff. This prompted further study into the nature of the error.

To analyze the source of the error, we first plotted the absolute and relative error. Next, we needed to find a bound for the truncation error. For the range of $x > 1$ we were able to take advantage of the theorem describing *Error Estimation of Alternating Series* which greatly simplified the process in deriving an upper bound. The challenge, however, became significant as we also needed a bound over the range $x < 1$. To achieve this we revisited the arbitrary bound derived in Problem 1. Since we were analyzing the Taylor polynomial of degree 3, we were able to find an expression for *M* by taking advantage of matlab's inherent precision error known as *machine epsilon*. Plotting each bound against the error plots, it was found that the truncation error was not the main problem. The bounds fell greatly below the plotted errors. This prompted the notion that the error was actually due to roundoff. In hindsight, this was to be expected as the ranges used must have caused matlab to round and lose precision over consecutive iterations of the Taylor polynomial calculation. The question was: how could one manipulate the Taylor polynomial function to prevent, or at least minimize, such an error?

It is understood that using the Taylor approximation for any value too far from its centered point leads to large errors. The approximation itself breaks down. To correct this, in Problem 4, function *myfofx(x)* was engineered. This function uses the Taylor approximation so long as the input is within a range of 1 ± 10^{-6}, and the exact value for any input outside it. The accuracy of this was verified in Figure 4 in the plot appendix. One could observe that the function did indeed execute as intended. This served to minimize error when using it to compute results of $f(x)$ at any *x*.

Overall, this investigation allowed for the exploration of error involving Taylor approximation. To motivate further study, one may analyze the roundoff error further. One could derive bounds for the roundoff error to more rigorously prove that the main source of error was in fact roundoff and not truncation.

References

[1] Sauer, T. (2019). Numerical analysis (3rd ed., p. 23). Hoboken, NJ: Pearson.

[2] CK-12. (2020, April 20). Maclaurin Series Truncation Error. Retrieved January 31, 2021, from https://www.ck12.org/calculus/maclaurin-series-truncation-error/lesson/taylor-and-maclaurin-polynomials:-series-truncation-error/#:~:text=A%20Maclaurin%20series%20is%20a,with%20x%3Dx_0%3D0.&text=A%20series%20truncation%20error%20is,used%20to%20estimate%20a%20function.

[3] Error Estimation for Approximating Alternating Series. (n.d.). Retrieved January 31, 2021, from http://mathonline.wikidot.com/error-estimation-for-approximating-alternating-series

[4] Xie, S. (2019, January 18). Error Estimation of Alternating Series. Retrieved January 31, 2021, from https://medium.com/self-study-calculus/error-estimation-of-alternating-series-a42be5d978b

Code Appendix

```
function T = lab1bP2(x, n)

    % Initialize sum as 0
    T = 0;

    % Loop over terms in series from k=0 to k=n
    for k = 0:n
        % Taylor polynomial in series notation form converted to operate in a
        % for loop
        T = T + (-1).^(k+1)./(k+2)*(x-1).^k;
    end
end
```

Listing 1: Function lab1bP2(x, n) takes parameters: x, the point or array of points at which the Taylor approximation is to be made; n, the number of terms of the Taylor series to be computed

```
x=linspace(0.999,1.001,50);
T=lab1bP2(x,3);
plot(x,T,x,(log(x)-x+1)./(x-1).^2)
```

Listing 2: Used to plot Figure 1(a)

```
x1=linspace(1-10^-7,1+10^-7,50);
T1=lab1bP2(x1,3);
plot(x1,T1,x1,(log(x1)-x1+1)./(x1-1).^2);
```

Listing 3: Used to plot Figure 1(b)

```
x2=linspace(1-10^-6,1+10^-6,50);
T2=lab1bP2(x2,3);
Fx=(log(x2)-x2+1)./(x2-1).^2;
AbsError=abs(T2-Fx);
RelError=abs(T2-Fx)./abs(Fx);
semilogy(x2,RelError);
semilogy(x2,AbsError);
```

Listing 4: Used to plot Figures 2(a) and 2(b)

```
x2=linspace(1-10^-6,1+10^-6,50);
xlt1=linspace(1-10^-6,1,25);
xgt1=linspace(1,1+10^-6,25);
T2=lab1bP2(x2,3);
Fx=(log(x2)-x2+1)./(x2-1).^2;
AbsError=abs(T2-Fx);
RelError=abs(T2-Fx)./abs(Fx);
trunc = (120/(eps)*abs(xlt1-1).^4);
semilogy(x2,AbsError,x2,RelError,xlt1,trunc,xgt1,((xgt1-1).^4)./6);
```

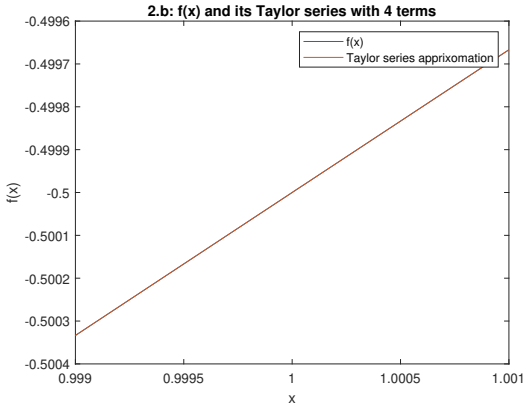
Listing 5: Used to plot Figures 2(a) and 2(b)

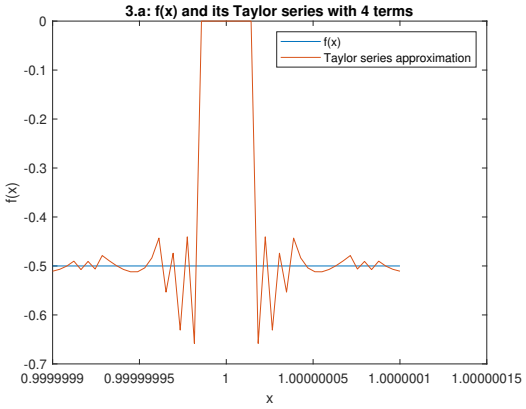
```
function [fx] = myfofx(x)
    % Initialize sum as 0
    T = 0;

    % Should user input array for x, take first and last entry of array
    first = x(1);
    last = x(length(x));
```

```
    % Check if array within range of 1-10^-6 < array < 1+10^-6
    if (last<1+10^-6) && (first>1-10^-6)
        % If within range, perform Taylor Series approximation
        % Loop over terms in series from k=0 to k=n
        for k = 0:8
        % Taylor polynomial in series notation form converted to operate in a
        % for loop
            T = T + (-1).^(k+1)./(k+2)*(x-1).^k;
        end
        fx = T;
    % If array not within range, use exact value to maintain accuracy
    else
        fx = (log(x)-x+1)./(x-1).^2;
    end
end
```

Listing 6: Used to plot Figure 4

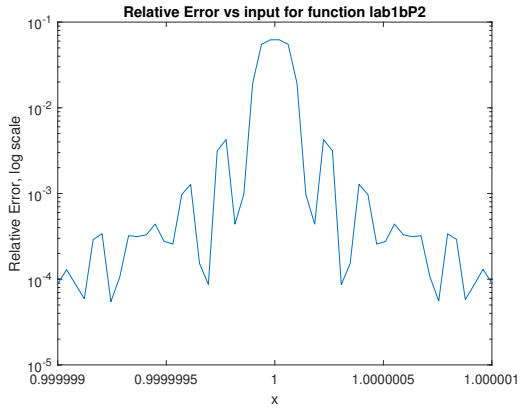
Plot Appendix

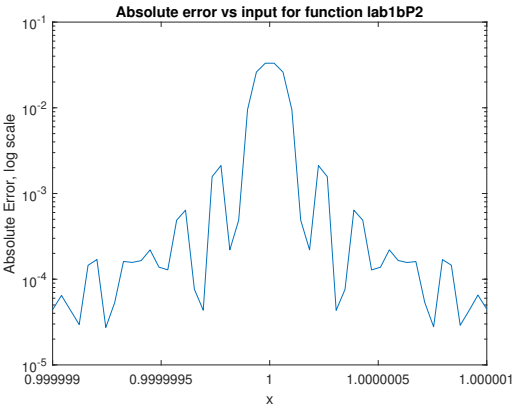




(a) Plotted using 50 points over range (0.999,1.001) (b) Plotted using 50 points over range $(1-10^{-7}, 1+10^{-7})$

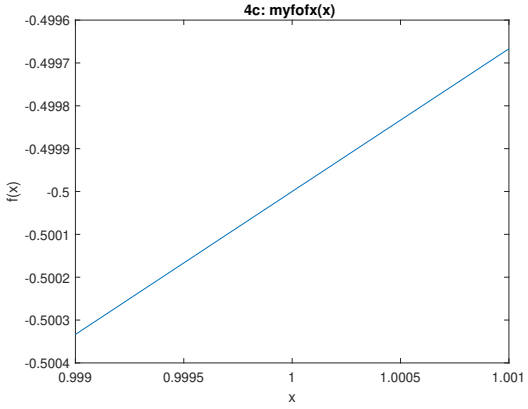
Figure 2: *(Problems 2b and 3a)* f(x) and its Taylor series with 4 terms over 2 different ranges.

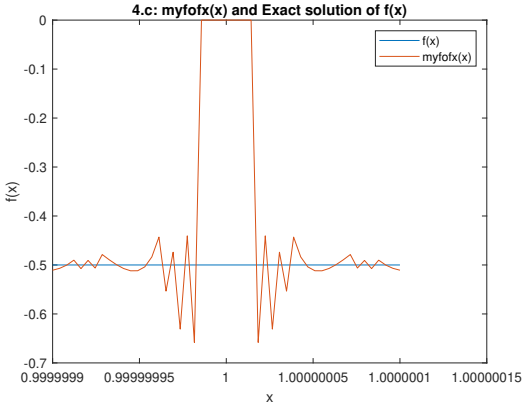




(a) Plotted using 50 points over range $(1-10^{-6}, 1+10^{-6})$ (b) Plotted using 50 points over range $(1-10^{-6}, 1+10^{-6})$

Figure 3: *(Problem 3b)* Relative Error (a), and Absolute Error (b) for function lab1bP2.m using a log scale.





(a) Plotted using 50 points over range (0.999,1.001) (b) Plotted using 50 points over range $(1-10^{-7}, 1+10^{-7})$

Figure 4: *(Problem 4c)* Result of using *myfofx(x)* function to reproduce results of problems 2 and 3. (b) has exact solution plotted as well such that one may make an easier comparison to Figure 1.

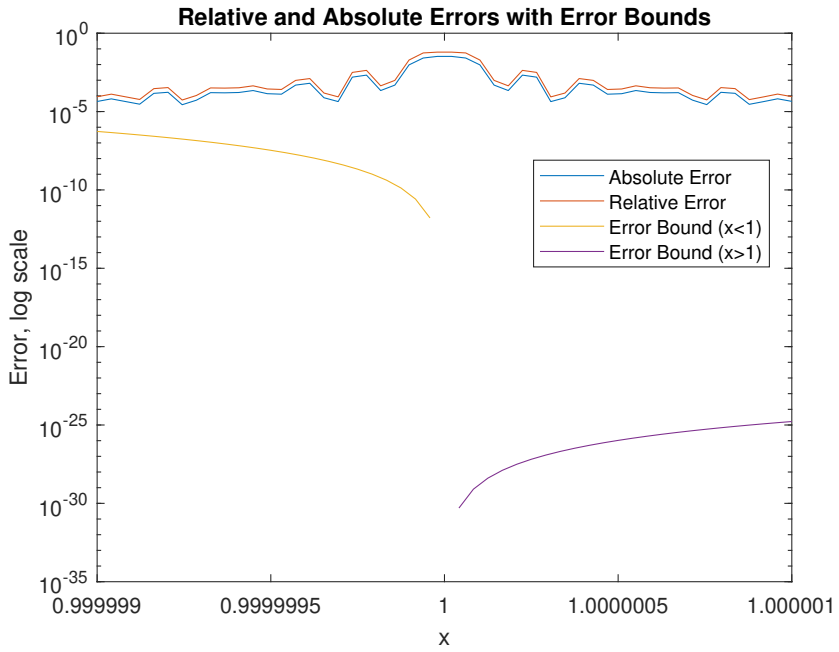


Figure 5: *(Problem 3c)* Relative and Absolute Error with Truncation Error bounds