# Project 01

Seth Goodman
*APSC 607 - Fall 2017*

# Introduction

---

This project explored three methods of approximating the roots of six different functions. Functions were examined over one or more non overlapping ranges, where unique roots exist, using the Bisection, Newton's, and Secant methods with varying tolerances. The behavior and characteristics of these methods are reviewed by examining the effectiveness of approximations across iterations, as well as through a comparison of the error (convergence rates) associated with each approximation. The following sections will present the methods used in MATLAB to explore functions, as well as the outputs and results.

# Methods

---

All computations were performed using MATLAB using the code (Table 1) accompanying this report (in a zip file). The *Results* section of this report contains the outputs for each function and range along with related observations and discussion. All figures and tables found in this report are available in the *output* subdirectory of the accompanying zip file. Additionally, all code and figures found in the zip file can be accessed via GitHub[1].

**Table 1**

| M File | Description |
|---|---|
| *main.m* | Primary script which runs all code. Includes definitions of the function examined and other variable definitions such as tolerance and ranges. Manages plotting and table outputs. |
| *bisection.m* | Function implementing Bisection method |
| *newton.m* | Function implementing Newton's method |
| *secant.m* | Function implementing Secant method |
| *modified_newton.m* | Function implementing modified version of Newton's method for dealing with multiple roots (Note: not used in core code and not fully tested) |

[1] https://github.com/sgoodm/apsc607/tree/master/project_01

The code for the Bisection, Newton's, and Secant methods are largely standardized (except where input parameters vary slightly based on specifics of method) and accept the function to examine, tolerance, number of iterations and key parameter such as starting point. In addition, each function accepts options which indicate whether to print additional outputs when run (such as for examining results for each iteration or debugging). All functions utilize standardized messages to indicate errors, status updates or successful calls.

Derivations used for approximation with Newton's method were performed first by hand (see *manual_derivations.JPG* in zip file) using basic derivative rules[2] for each term in the equation. All derivatives were confirmed using Wolfram Alpha[3] as well as MATLAB's differentiation function[4] from the Symbolic Math Toolbox. These derivations were then manually added to the MATLAB code to be used as input arguments for the Newton's method function.

Each function produces three arrays used to generate plots (approximation values at each iteration, error value at each iteration, array of iteration number), and outputs the same four additional values: number of iterations run, root approximation at last iteration run, difference/error at final iteration, and a status code indicating success/error (e.g., failure to converge).

For each function-range combination (11 total, plus 4 additional cases used to examine specific properties of certain function-range combination) five figures are produced. The first is a plot of the true function over the specified range with the zero line and exact root value displayed. The second is a comparison of the error at each iterations for each method.

The last three plots are the root approximation at each iteration for each method. The results of approximations for each of the three tolerance values are plotted side by side in each of the approximation plots. In addition, the final values (approximation and difference) for each tolerance are included the legends of these plots. Finally, a table containing results from all the cases explored (function-range-method-tolerance combinations) is output as a CSV file.

The *Results* portion of the report is broken down into sections for each of the six functions examined in this project. Results for functions which were examined over multiple ranges are combined. The results for the first function will provide additional contextual information about how the figures were generated, and go over the behavior of the methods in some detail. Subsequent sections will be condensed to only include essential results[5] and noteworthy remarks specific to that case, when needed.

[2] https://www.mathsisfun.com/calculus/derivatives-rules.html
[3] https://www.wolframalpha.com/
[4] https://www.mathworks.com/help/symbolic/differentiation.html
[5] Figures or tables that are not included in the Results section are included in the outputs subdirectory of the zip file containing this report.

# Results

---

# Function A

Figure A-1 is a plot of the true function (Equation A) over the specified range (1 to 2) generated using *fplot* function in MATLAB, along with a zero line to identify where roots exist. The actual root which exists within this range was calculated using the *fzero* function in order to validate the results of the approximation methods.

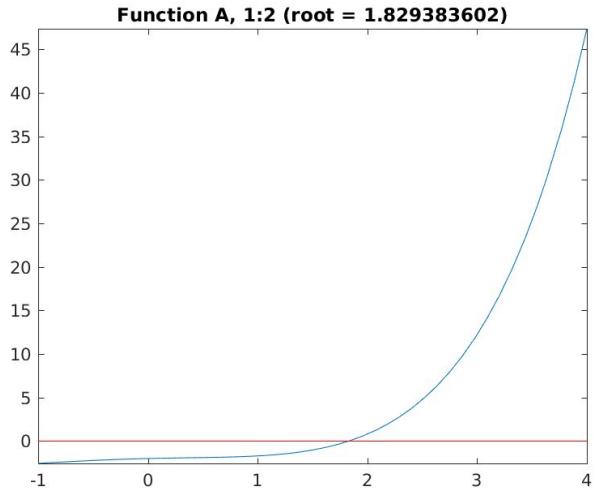



**Figure A-1**

$$f(x) = e^{x} + 2^{-x} + 2 * cos(x) - 6$$

**Equation (A)**

In each of the following figures (Figures A-2, A-3, A-4, A-6), the results for all three tolerance settings are plotted alongside each other. As indicated by the legends, the largest tolerance (10E-3) appears as a dark blue dashed line[6], the middle tolerance (10E-5) appears as light blue dashed line, and the smallest tolerance (10E-8) appears as pink circular points.

The number of iterations required to reach the specified tolerance is indicated in the legends, as are the final approximation values (value at last iteration) and the final difference/error value (difference at last iteration; hereafter referred to as *difference value*, rather than *error*). The equation for the difference value varies slightly between methods. For the Bisection method, the difference is defined as half the difference between the points *a* and *b* which are being bisected in the current iteration (i.e., the difference between *a* and *b* in the *next* iteration). For Newton's Method, the difference is the absolute difference between *p* and *p0*. And similarly, for the Secant method, it is the absolute difference between *p* and *p1*.[7]

[6] Technically this is a solid blue line, but the overlaid light blue dashed line makes the solid dark blue line appear dashed.
[7] See code (https://github.com/sgoodm/apsc607/tree/master/project_01) for variable references

The Bisection method's approach of "splitting the difference" each iteration often results in a plot of approximation that has the appearance of attenuation around the true root (Figure A-2). This characteristic of Bisection plots is characteristic of the linear nature of this method's convergence. The results from Newton's Method (Figure A-3 and A-4) and the Secant method (Figure A-5) are in stark contrast due to the Bisection, as they converge quadratically.

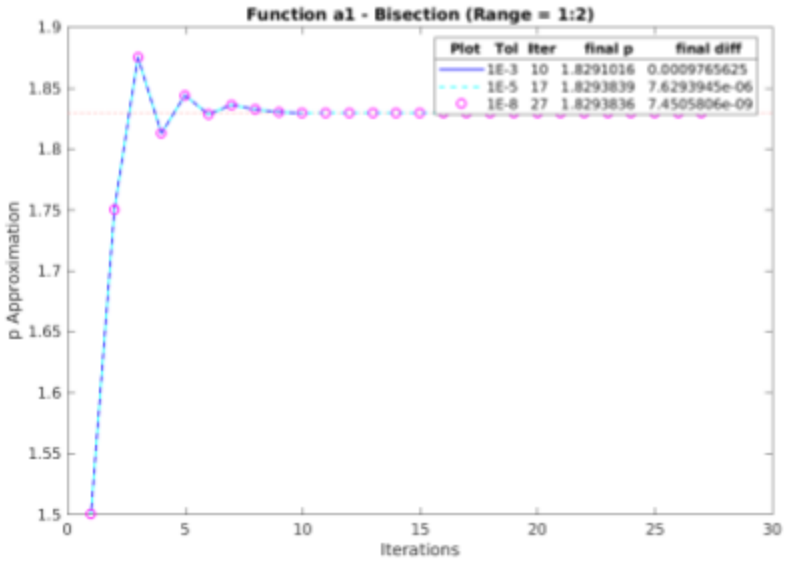



**Figure A-2 - Bisection method**

For Newton's Method, the initial starting point used is the range minimum. A better starting point can generally be used to reduce the number of iterations needed for convergence. The range minimum (*p0* = 1, figure A1-3) was compared to a value of approximately 90% of the result from the Bisection method (*p0* = 1.65, figure A1-4) in Table A1-1.

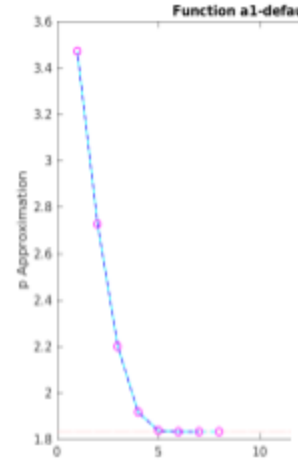
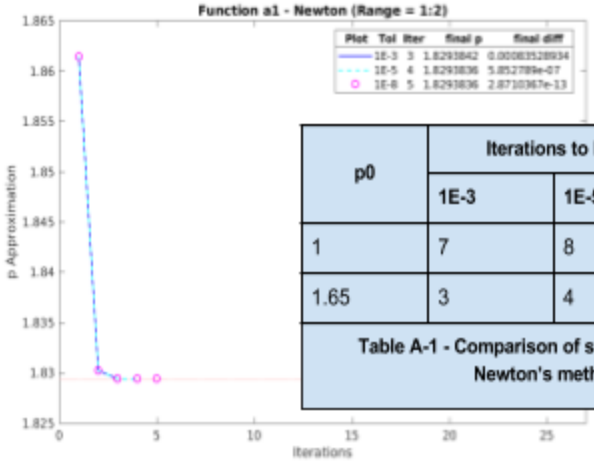



**Figure A-3 : Default p0**




| p0 | Iterations to Reach Tolerance | | |
|---|---|---|---|
| | 1E-3 | 1E-5 | 1E-8 |
| 1 | 7 | 8 | 8 |
| 1.65 | 3 | 4 | 5 |

**Table A-1 - Comparison of starting points for Newton's method**

**Figure A-4 : Improved p0**

Improving the starting point for Newton's method reduced the number of iterations required to achieve the specified tolerance by about 50%. In situations where significant iterations (and run times) are needed to reach an approximation within the desired tolerance, improving the starting point estimate would likely be beneficial. For most of the cases examined in this project, sufficient iterations can be run very quickly and using the range minimum as a starting point for Newton's method is sufficient.

This function is actually the only example in this project where Newton's Method using the default starting point is slower to converge than the Secant method (Figure A-6). After improving the starting point, the error plots look similar to the other functions (Figure A-7).

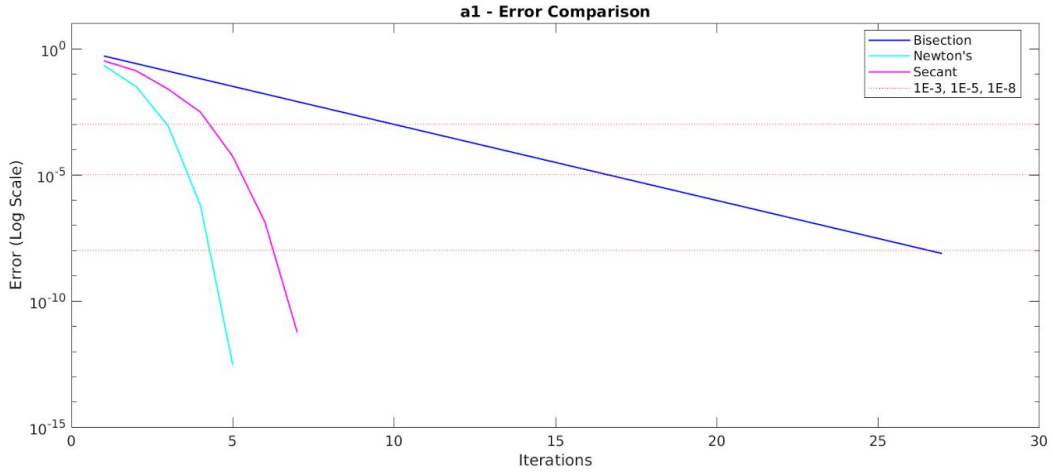



**Figure A-5 : Error comparison using default starting point for Newton's Method**

Two of the examples explored later in this report (functions C and E) will deal with an additional scenario where adjusting the starting point is necessary due to convergence on a root outside the specified range when using the range minimum as the default starting point for Newton's Method.

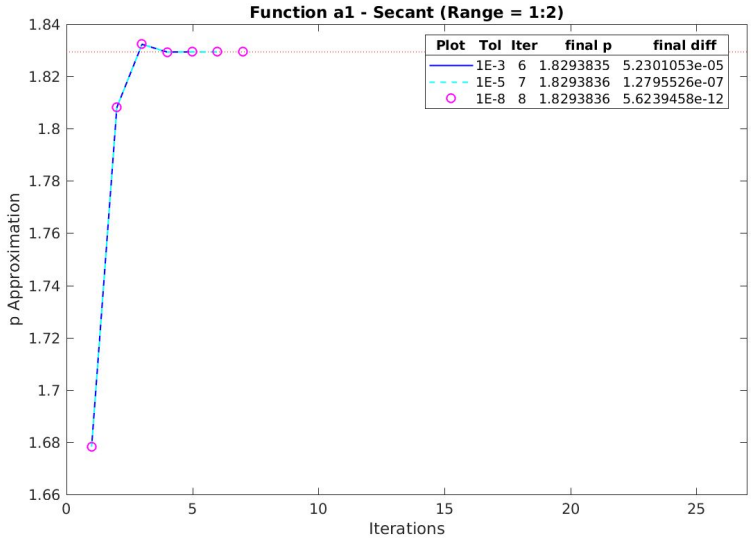



**Figure A-6 - Secant method**

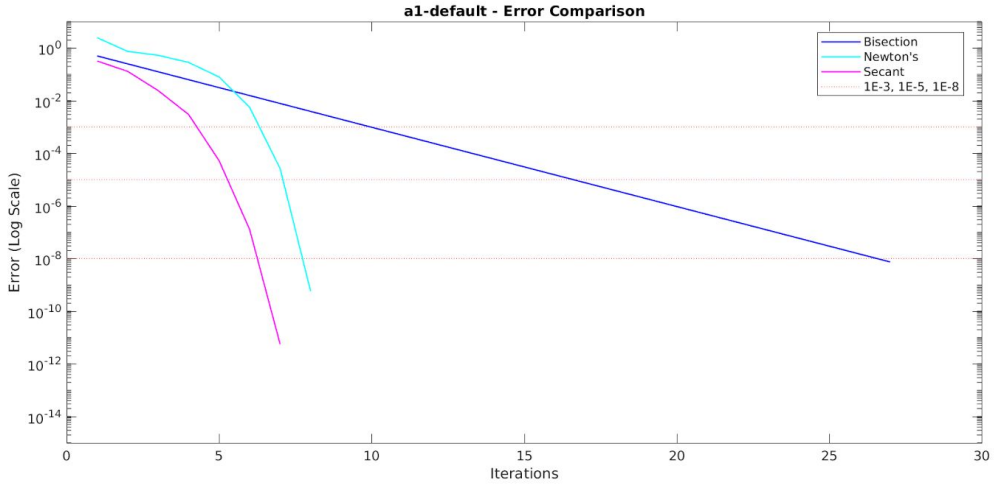



**Figure A-7 : Error comparison using improved starting point for Newton's Method**

While the Bisection method performed only slightly worse than the other methods at the largest tolerance, as tolerance decreased the Newton's method (using the adjusted starting point of 1.65, mentioned above) and the Secant method were able to converge notably faster (Table A-2).

**Table A-2 - Comparison of methods**

| Name | Range | Method | Iterations to Reach Tolerance | | |
|---|---|---|---|---|---|
| | | | 1E-3 | 1E-5 | 1E-8 |
| A1 | 1:2 | bisection | 10 | 17 | 27 |
| | | newton | 3 | 4 | 5 |
| | | secant | 6 | 7 | 8 |

The notable difference between the Bisection method and Newton's or Secant method can be attributed to the linear convergence of the Bisection method compared to the quadratic convergence of Newton's method and the Secant method. Although Newton's Method in particular can provide highly accurate approximations with fewer iterations, it does so by assuming:

$$(p - p_0)^2 << |p - p_0|$$

For the terms as seen in the below equation[8] for the first Taylor polynomial:

$$f(p) = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2} f''(\xi(p)),$$

Which reduces to the simplified equation for Newton's Method:

$$p \approx p_0 - \frac{f(p_0)}{f'(p_0)} \equiv p_1.$$

[8] Burden, R., Faires, J., Numerical Analysis. 2010

This results in an accurate approximation that requires few iteration, given an accurate starting point. In addition to necessitating an accurate starting point estimation, Newton's method also requires calculating the derivative of the main function (and in certain cases dealing with multiple roots, the second derivative) which can become computationally intensive for complicated functions (not an issue that will be explored in this project).

The final difference value is smallest for the smallest tolerance when using Newton's method, while requiring the fewest iterations. Going beyond the tolerance range for this project, to values of 1E-10, 1E-12, and 1E-15 Newton's Method (as well as the Secant method) will effectively converge on a zero difference value (though this may simply be such a small number that the computation displays it as zero[9].

**Table A-3 - Comparison of method with smaller tolerances**

| Name | Range | Method | Iterations to Reach Tolerance | | |
|---|---|---|---|---|---|
| | | | 1E-10 | 1E-12 | 1E-15 |
| B1 | 1.3:2 | bisection | 34 | 40 | 50 |
| | | newton | 5 | 5 | 6 |
| | | secant | 8 | 9 | 9 |

As seen in the Table A-3, Newton's method required only a single additional iteration with a reasonably accurate starting point (1.65) to reach a tolerance of 1E-15 (as did the Secant method) compared to a tolerance of 1E-8, while the Bisection method require 23 additional iterations. The accuracy of both Newton's Method and the Secant method can be clearly seen in the plot of the error when using smaller tolerance values (Figure A-8), as can the linear nature of the Bisection method.

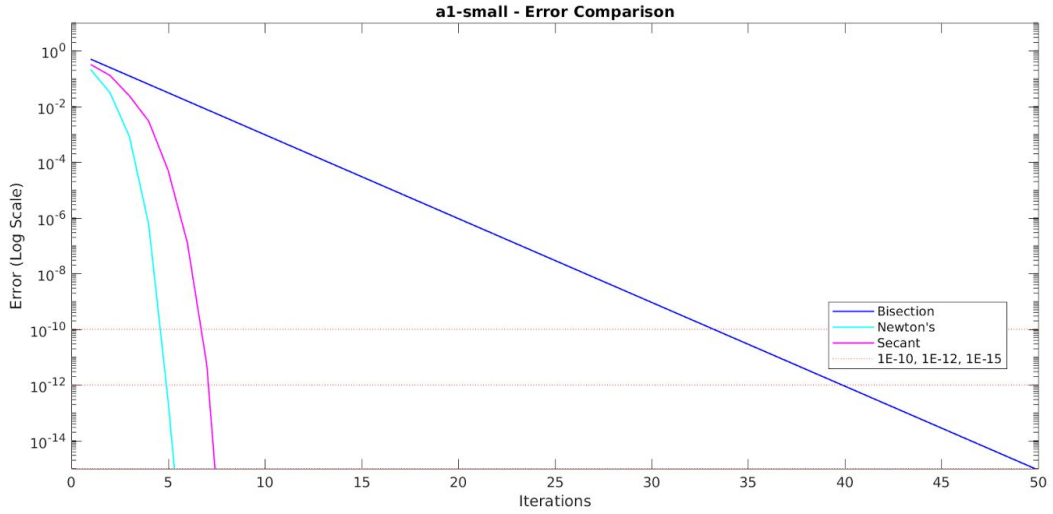



**Figure A-8**

[9] See following link on MATLAB precision limitions (general limitations of floating point representations apply) https://www.mathworks.com/help/fixedpoint/ug/limitations-on-precision.html

# Function B

The second function was examined over only a single range, containing a root at 1.397748476. The resulting plot of this function (Equation B) can be seen in Figure B-1.

$$f(x) = log(x-1) + cos(x-1) \qquad \textbf{Equation (B)}$$

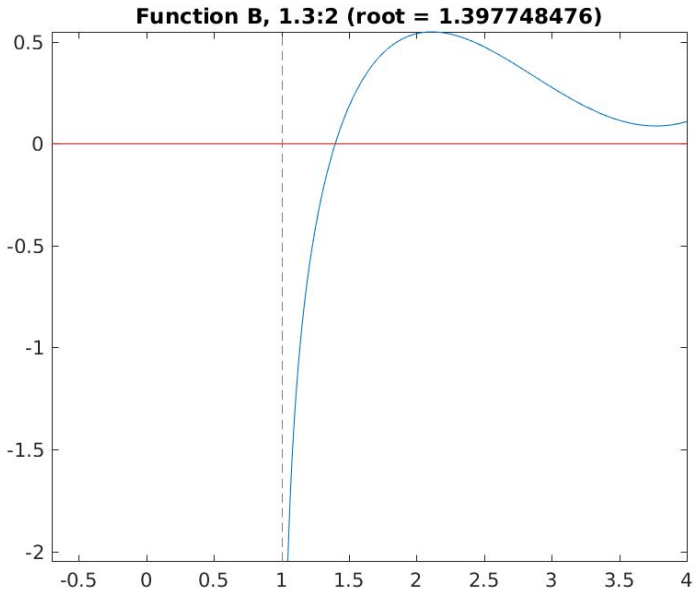



**Figure B-1**

It is worth noting here that for this case, Newton's method still converged faster than the Secant method despite using the default starting point due to the default starting point being quite close to the root. For the previous example using function A, the default value (1) was further from the root (1.8…).

The Bisection Method took significantly longer to converge on the root than either Newton's Method or the Secant Method, similar to Function A. As seen in Table B-1, each successive tolerance threshold takes only a single additional iteration to reach when using Newton's Method, while the Bisection Method can require up to 10 additional iterations.

**Table B-1**

| Name | Range | Method | Iterations to Reach Tolerance | | |
|---|---|---|---|---|---|
| | | | **1E-3** | **1E-5** | **1E-8** |
| B1 | 1.3:2 | bisection | 10 | 17 | 27 |
| | | newton | 3 | 4 | 5 |
| | | secant | 7 | 9 | 10 |

Interestingly, even though the Secant method clearly outperforms the Bisection Method at any tolerance which would produce a reasonable degree of accuracy, during the first few iterations the Bisection Method does technically produce more accurate results (Figure B-2).

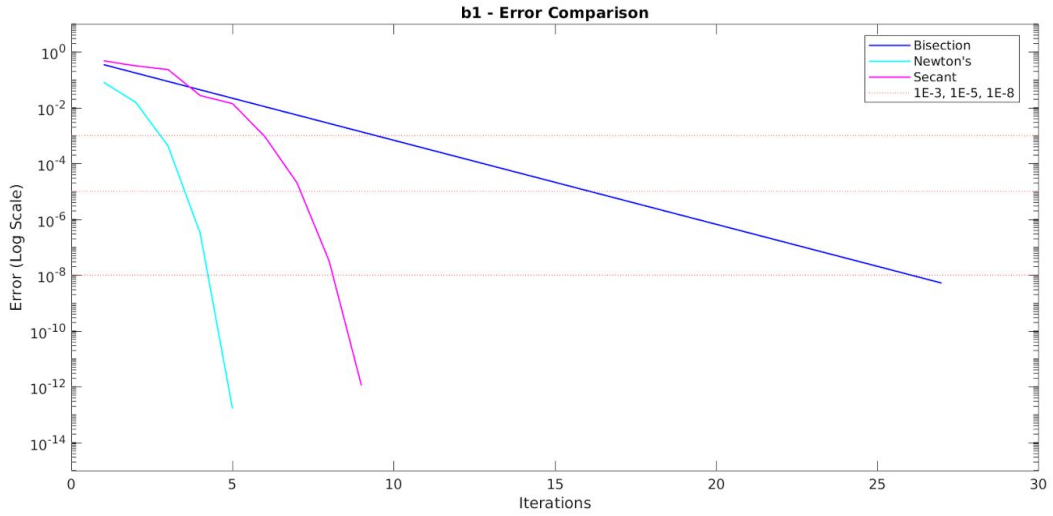



**Figure B-2**

# Function C

$$f(x) = 2 * x * cos(2 * x) - (x - 2)^2$$ **Equation (C)**

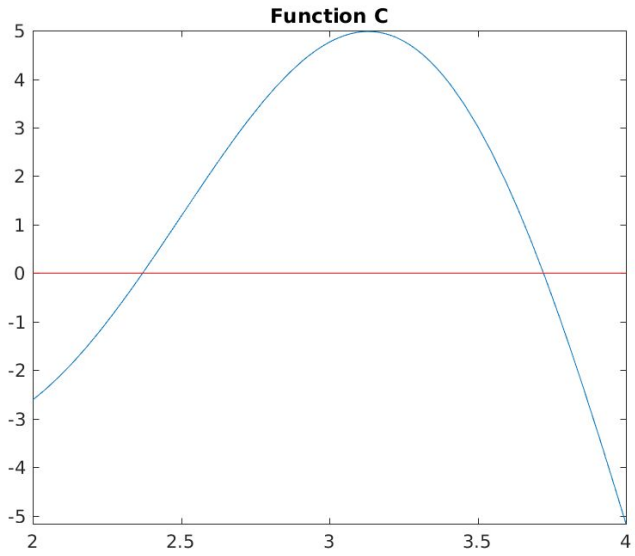



**Figure C-1**

**Table C-1**

| Range | True Root |
|---|---|
| 2:3 | 2.370686918 |
| 3:4 | 3.722112773 |

As with function B, the default value and root for the first range of function C are reasonably close together, so the default value ends up being a reasonable starting point for Newton's method. Newton's method is slightly faster to converge than the Secant method, yet the final accuracy is actually notable better (~5E-15 for Newton's vs ~10E-9 for Secant) which is visible in the error comparison plot for (Figure C-2) This is indicative of Newton's method's ability to make substantial improvements with every additional iteration.

**Table C-2**

| Name | Range | Method | Iterations to Reach Tolerance | | |
|---|---|---|---|---|---|
| | | | 1E-3 | 1E-5 | 1E-8 |
| C1 | 2:3 | bisection | 10 | 17 | 27 |
| | | newton | 3 | 4 | 5 |
| | | secant | 5 | 6 | 6 |
| C2 | 3:4 | bisection | 10 | 17 | 27 |
| | | newton | 4 | 4 | 5 |
| | | secant | 6 | 7 | 8 |

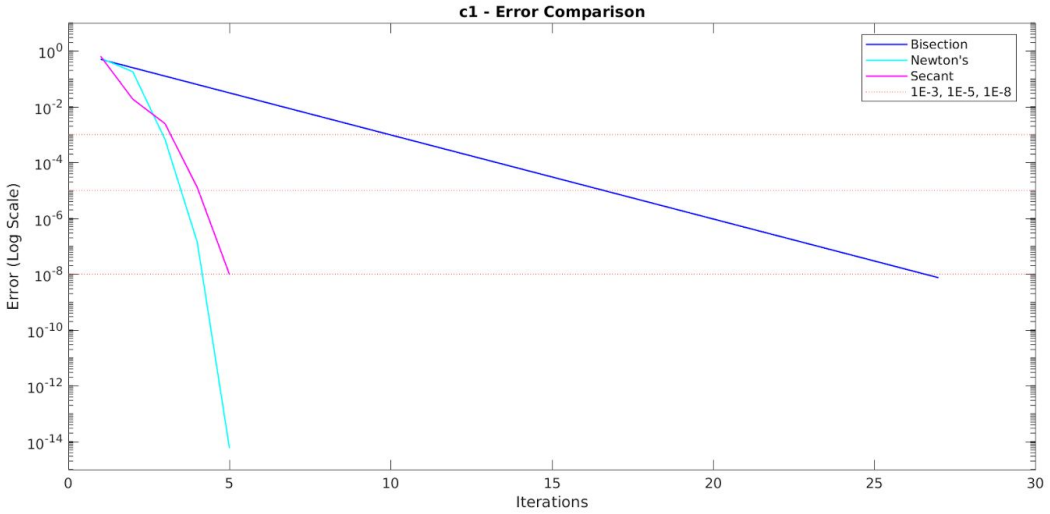



**Figure C-3**

For the second range of function C (3:4), there is a local maximum within the range for the function. Due to the initial starting point occurring on the opposite site of this local maxima, this is an instance where the initial starting point for Newton's method results in convergence on the wrong root (Figure C-3).

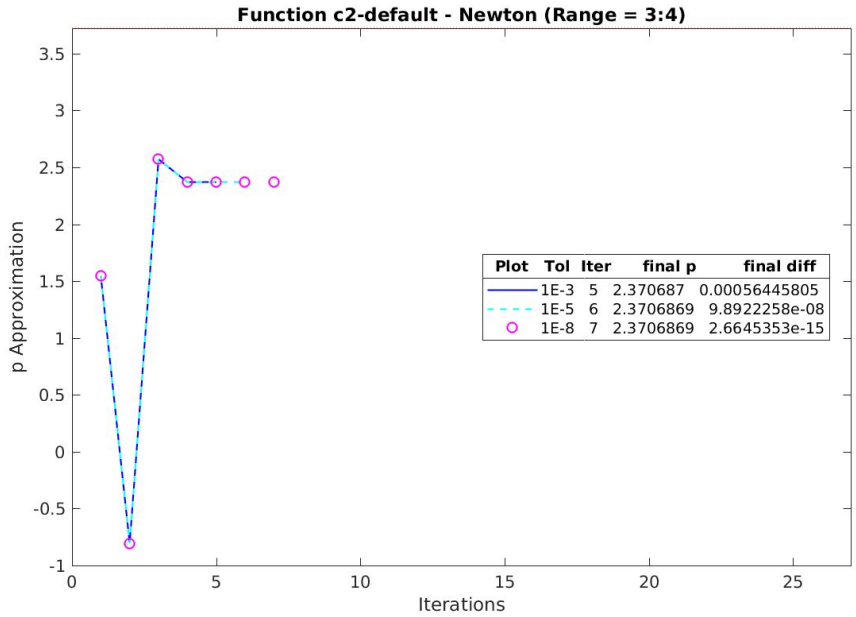



**Figure C-2**

Due to a local maximum within the specified range, and roots on either side of the maximum, a starting point on one side of the local maximum is likely to converge on the root on the same side of the local maximum. When the starting point is set to the range minimum, zero, it converges on the root of 2.37 which as they are both to left of the local maximum. By adjusting the starting point to be on right of the

local maximum, the same as our desired root, Newton's method will converge on the desired root of 3.72 (Figure C-4).

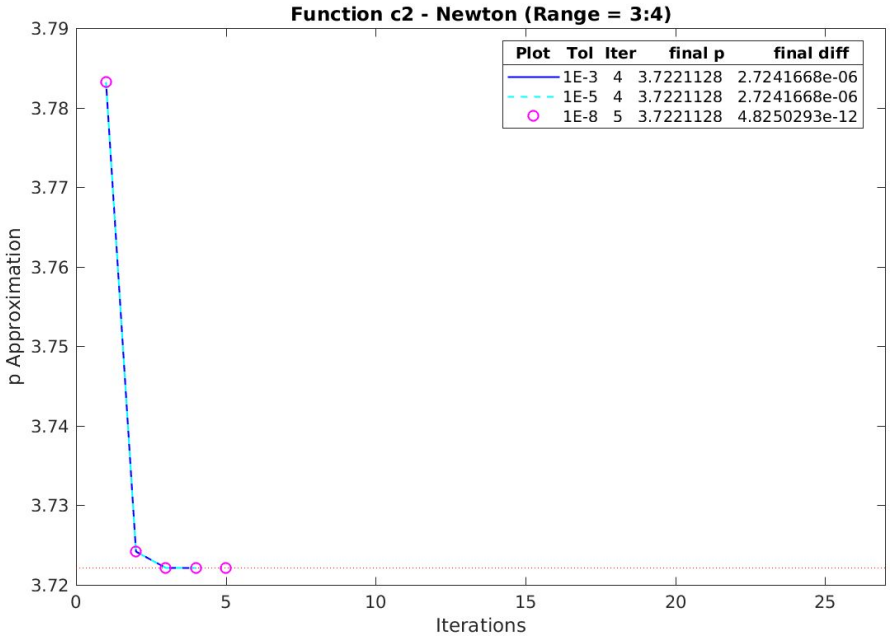



**Figure C-4**

After adjusting the starting point for Newton's method to converge on the correct root, the methods behave similar to the previous examples with Newton's Method slightly outperforming Secant, and both surpassing the efficiency of the Bisection method..

# Function D

The fourth function, seen in Equation D, was examined over two ranges which each contained a unique root (Table D-1). This function can be seen across the entirety of the ranges in Figure D-1

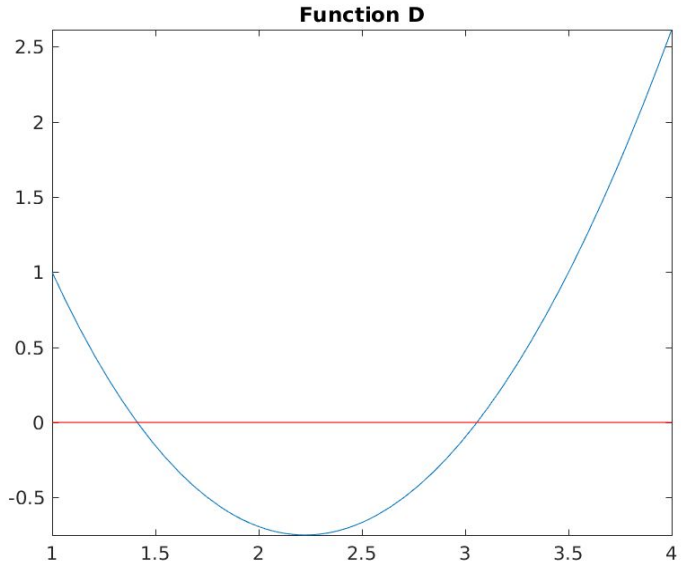



**Figure D-1**

$$f(x) = (x-2)^2 - log(x)$$

**Equation (D)**

**Table D-1**

| Range | True Root |
|---|---|
| 1:2 | 1.412391172 |
| e:4 | 3.05710355 |

The methods perform similarly over both ranges, with Bisection once again trailing both Newton's Method and the Secant method in terms of number of iterations required to reach each tolerance threshold. Newton's Method also had a clear advantage over the Secant method, require over 40% fewer iterations at the smallest tolerance (Table D-2).

**Table D-2**

| Name | Range | Method | Iterations to Reach Tolerance | | |
|---|---|---|---|---|---|
| | | | 1E-3 | 1E-5 | 1E-8 |
| D1 | 1:2 | bisection | 10 | 17 | 27 |
| | | newton | 4 | 4 | 5 |
| | | secant | 7 | 8 | 9 |
| D2 | e:4 | bisection | 11 | 17 | 27 |
| | | newton | 4 | 5 | 5 |
| | | secant | 6 | 7 | 8 |

Once again, a comparison of the error plots using log scale (Figure D-2) illustrates the linear nature of the Bisection method's convergence, while Newton's Method and the Secant method improve significantly due to quadratic characteristics of their convergence.

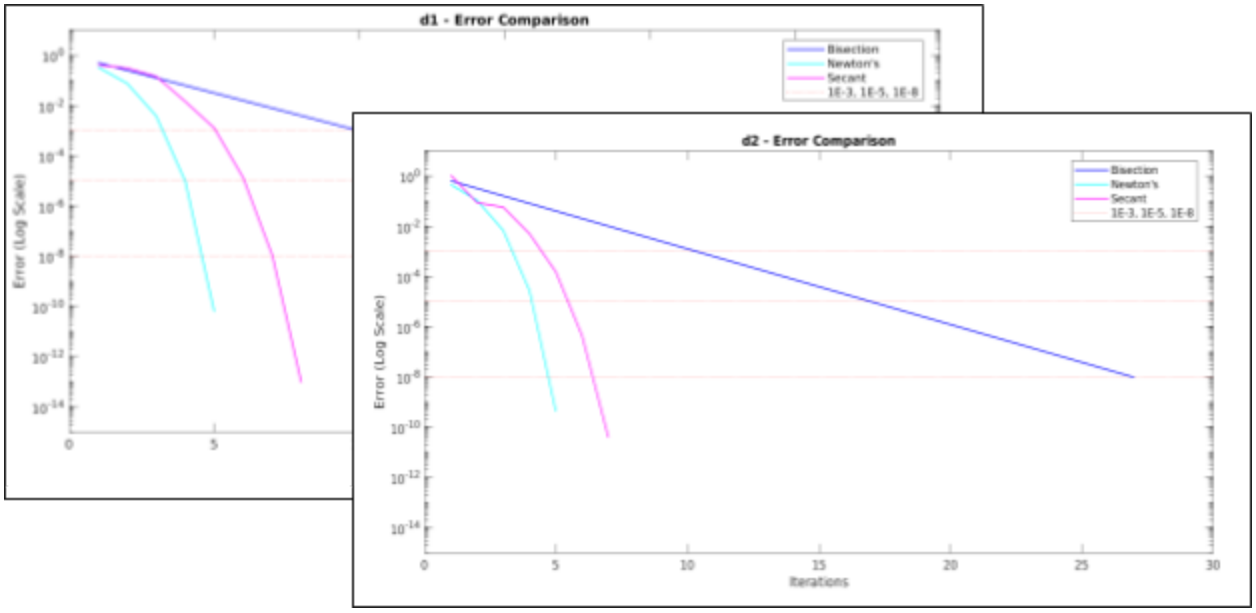



**Figure D-2**

# Function E

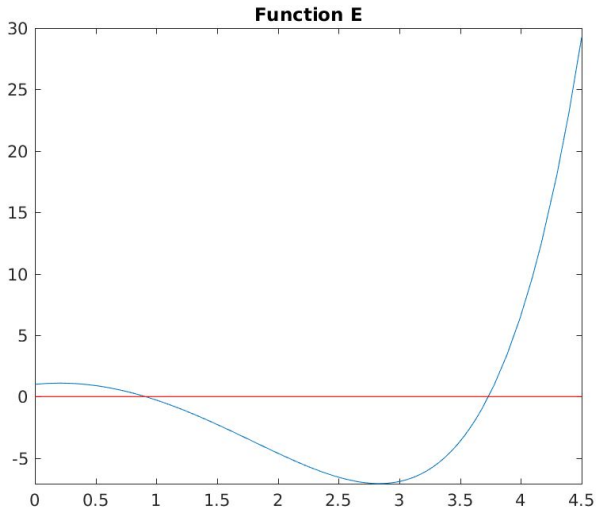



**Figure E1-1**

$$f(x) = e^x - 3 * x^2$$ **Equation (E)**

**Table E-1**

| Range | True Root |
|---|---|
| 0:1 | 0.9100075725 |
| 3:5 | 3.733079029 |

As seen in figure E-2 for Newton's method, this is the second instance where the initial starting point for Newton's method results in convergence on the wrong root. Due to a local maximum within the specified range, and roots on either side of the maximum, a starting point on one side of the local maximum is likely to converge on the root on the same side of the local maximum. When the starting point is set to the range minimum, zero, it converges on the root of -0.45 which as they are both to left of the local maximum. By adjusting the starting point to be on right of the local maximum, the same as our desired root, Newton's method will converge on the desired root of 0.91, as seen in figure E1-4.

**Table E-2**

| Name | Range | Method | Iterations to Reach Tolerance | | |
|---|---|---|---|---|---|
| | | | 1E-3 | 1E-5 | 1E-8 |
| E1 | 0:1 | bisection | 10 | 17 | 27 |
| | | newton | 4 | 5 | 6 |
| | | secant | 5 | 6 | 7 |
| E2 | 3:5 | bisection | 11 | 18 | 28 |
| | | newton | 8 | 9 | 10 |
| | | secant | 9 | 10 | 11 |

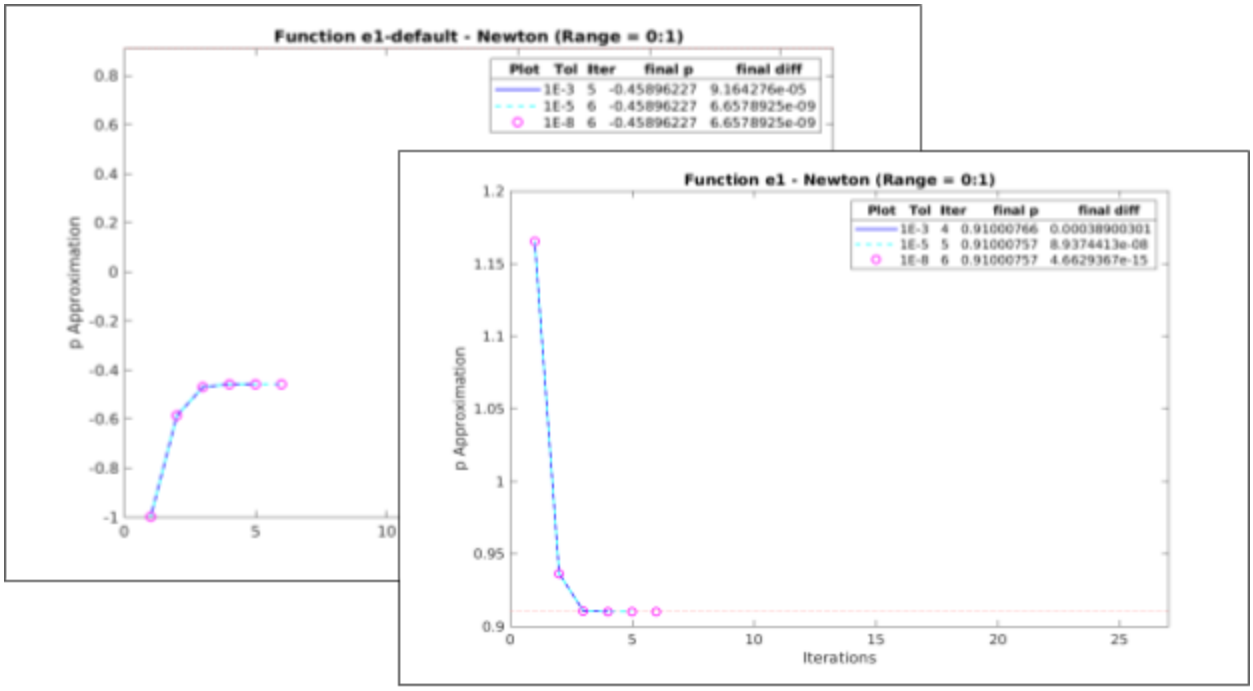



**Figure E-2**

It is worth noting that for E2, Newton's Method and Secant Method seem to do worse than Bisection early on, but as the iterations increase, they quickly correct and approach the true root, while bisection continues to only make linear progress (Figure E-3).

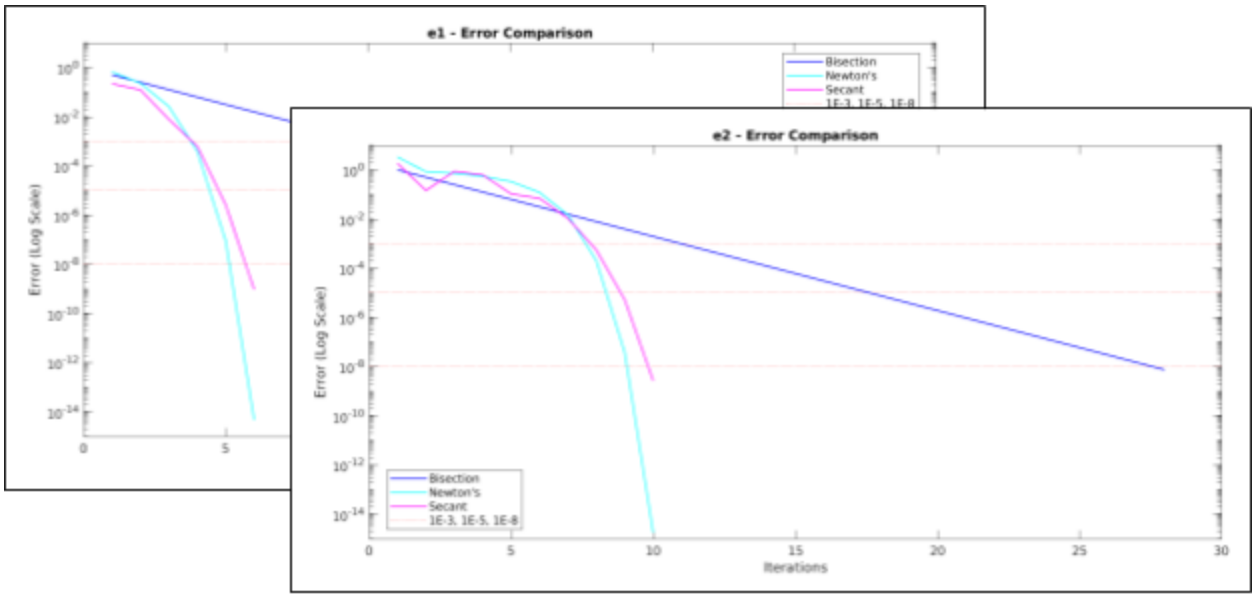
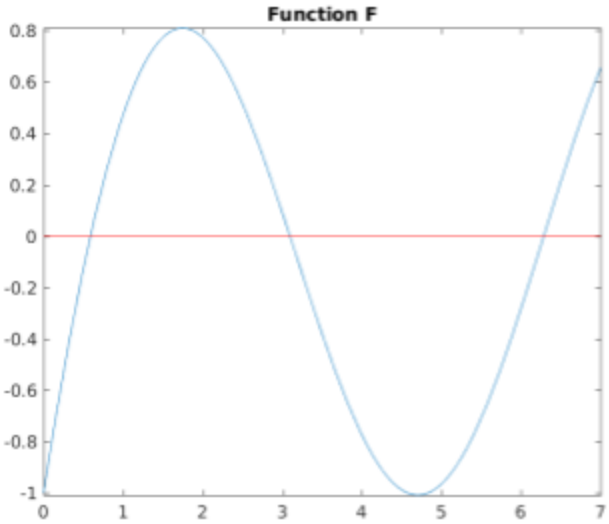



**Figure E-3**

# Function F

The final function, F, was examined over three ranges (Figure F-1) which each contain a unique root. The results of the approximations using Newton's Method, the Secant method, and the Bisection method across these ranges of function F indicated similar behavior for each method compared to the previous functions examined (Table F-1).




$$f(x) = \sin(x) - e\text{-}x$$

| Range | True Root |
|---|---|
| 0:1 | 0.588532744 |
| 3:4 | 3.096363932 |
| 6:7 | 6.285049273 |

**Figure F-1**

Newton's Method, using only the default starting point, was able to approach the true root most quickly (Table F-1), requiring only one additional iteration for each finer level of tolerance (in some cases it even surpassed multiple tolerance thresholds in a single iteration).

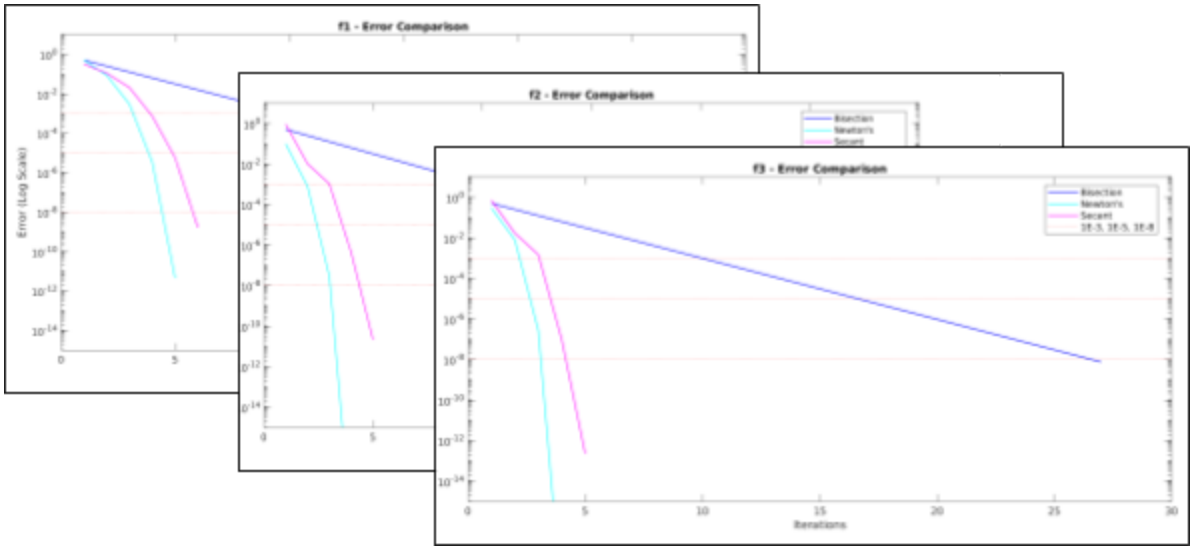



**Figure F-2**

**Table F-1**

| Name | Range | Method | Iterations to Reach Tolerance | | |
|---|---|---|---|---|---|
| | | | **1E-3** | **1E-5** | **1E-8** |
| F1 | 0:1 | bisection | 10 | 17 | 27 |
| | | newton | 4 | 4 | 5 |
| | | secant | 5 | 6 | 7 |
| F2 | 3:4 | bisection | 10 | 17 | 27 |
| | | newton | 2 | 3 | 4 |
| | | secant | 5 | 5 | 6 |
| F3 | 6:7 | bisection | 10 | 17 | 27 |
| | | newton | 3 | 3 | 4 |
| | | secant | 5 | 5 | 6 |