

What Secret the Bisection Method Hides?

by

Namir Clement Shammas

Introduction

Over the past few years I have modified the simple root-seeking Bisection Method to enhance it. I obtained good results by adding more steps to that classic algorithm. Recently, I decided to give it another go at it using a different approach. The core equation used for updating the guess for the root of $f(x)=0$, given a root-bracketing interval $[A, B]$ is:

$$C = (A + B) / 2 \tag{1}$$

The value of C replaces A if $f(C)$ and $f(A)$ have the same sign. Otherwise, C replaces B . Equation 1 allows the algorithm to reduce the root-bracketing interval by half for each iteration. The Bisection Method is one of the few root-seeking algorithms where you can calculate ahead the number of iterations required to refine the guess for the root given the initial root-bracketing interval and the solution tolerance value.

My first attempt was very naïve, and frankly, lacked planning—simply replace the integer 2 with the golden ratio, 1.618033989, and see what happens. The results were terrible! The new algorithm that used the golden ratio diverged! I continued my naïve thinking by replacing the golden ratio with simpler numbers like 1.5 and 1.75, as if that would rectify the problem! The results were just as disappointing.

I almost gave up on the attempt to improve the Bisection Method until I realized that equation 1 is really:

$$C = (1 \cdot A + 1 \cdot B) / (1 + 1) \tag{1b}$$

A more general version of equation 1b is:

$$C = (w_1 \cdot A + w_2 \cdot B) / (w_1 + w_2) \tag{2}$$

Setting both weights w_1 and w_2 to 1 yields equations 1 and 1b. This makes sense because as the bisection iterations progress, A and B approach and surround the true value of the root ρ . This calculus limit turns equation 1 into:

$$\begin{aligned}
 \lim_{A,B \rightarrow \rho} C &= (\rho + \rho) / 2 \\
 &= 2 \rho / 2 \\
 &= \rho
 \end{aligned}
 \tag{3}$$

Which allows the value in C to also diverge towards the true value of the root. In addition, it became clear that replacing 2 with any other value causes C to diverge from the root! You can draw the same conclusion if you replace the variables A and B with ρ using equation 2:

$$\begin{aligned}
 \lim_{A,B \rightarrow \rho} C &= (w_1 \rho + w_2 \rho) / (w_1 + w_2) \\
 &= (w_1 + w_2) \rho / (w_1 + w_2) \\
 &= \rho
 \end{aligned}
 \tag{4}$$

The question becomes how can I use the golden ratio with equation 2? We start with the premise that:

$$\text{Phi} = w_1 + w_2 = 1.618033989 \tag{5}$$

How do we split the golden ratio, Phi , among the two weights in equation 5? There are two general approaches:

- Manual splitting by selecting one weight, say w_1 , and calculating the other as $w_2 = \text{Phi} - w_1$.
- Calculating one of the weights using the absolute function values at A and B and the value of Phi . The other weight is calculated using equation 5.


Let me focus on the first approach for splitting the sum of weights. I chose a simple scheme to use when the values of the weights w_1 and w_2 are not both equal to 1:

```

If |f(A)| < |f(B)| then
    C = (A + (Phi - 1) * B) / Phi
Else
    C = (B + (Phi - 1) * A) / Phi
End If

```

Listing 1. The If statement that needs to be included in modifying the Bisection Method when weights w_1 and w_2 are not both equal to 1.



Throughout this paper, I use Phi to be more than just the golden ratio. I use it to represent the sum of the weights w_1 and w_2 that appear in equation 2.

This means that I assign 1 to the weight associated with the interval end having the smallest absolute function. Consequently, I assign the value $\Phi-1$ to the weight associated with the other interval end. In the case of using the golden ratio, one weight is 1 and the other is 0.618033989. Such a partition assigns the higher weight of 1 to the advantageous interval end. Likewise, the partition assigns a smaller weight (that is, a number smaller than 1) to the disadvantageous interval end. Of course, there is virtually an infinite number of ways to split the golden ratio into two unequal positive numbers!

Testing with the Golden Ratio

I compared using classical Bisection methods with the golden ratio variant using two functions—one with three roots and the other with a single root. Table 1 shows the results and specifies the initial range [A, B] and the tolerance used.

<i>Equation</i>	<i>A</i>	<i>B</i>	<i>Tolerance</i>	<i>Iterations w/Bisection</i>	<i>Iterations w/Golden Ratio</i>
$\exp(x)-3*x^2$	3	7	$1e-8$	29	23
	3	4	$1e-8$	28	22
	-3	0	$1e-8$	29	23
	-1	0	$1e-8$	28	21
$\ln(100)-x$	0	5	$1e-8$	28	23
	2	5	$1e-8$	29	22
	4	5	$1e-8$	27	22

Table 1. Results comparing the Bisection Methods with the variant that uses the golden ratio.

Table 1 shows very encouraging results with the golden ratio variant requiring fewer iterations than the classical Bisection Method in every test.

Can We Expand on the Concept?

We can certainly expand on the above concept. For example, we can set $w_1 + w_2$ to 4, and then split the weights into 1 and 3, or better yet, into 1.5 and 2.5. I can also set these sums to 10 and split them into 6 and 4, or 6.5 and 3.5, and so on. I can even set the sum of weights to 1000, and split them into all sorts of combinations, such as 600 and 400, 650 and 350, 700 and 300, and so on. How we split the sum of the two weights is going to influence the speed of conversion. This influence is due to the relative values of weights that determines how strongly we favor the better interval end for each iteration.

I thought about replacing the golden ratio with other numbers other than 2. The first set included numbers smaller than 2 like $\sqrt{2}$, 1.25, 1.2, 1.15, and 1.1. In the second category I chose numbers greater than 2 like $\ln(10)$, π , and $\ln(100)$. Both sets yield algorithm variants that converge to the root faster than the classical Bisection Method! When working with the first set of $\sqrt{2}$, 1.25, 1.2, 1.15, and 1.1, I split each of these values into 1 and the remaining fractional value.

The function $f(x)=\exp(x)-3*x^2$ has roots at -0.4590 , 0.9100 , and 3.7331 . It has a maximum at $x= 0.2045$ and a minimum at $x= 2.8331$.

Table 2 shows the number of iterations for various sums of weights for the initial root-bracketing interval of $[3, 7]$. Figure 1 shows the plots for the results in Table 2.

<i>W1 + W2</i>	<i>Iterations</i>
1.1	24
1.15	23
1.2	23
1.25	21
1.414213562	21
1.618033989	23
2	29
2.302585093	26
3.141592654	22
4.605170186	21

Table 2. Results for $f(x)=\exp(x)-3*x^2$ for $[3, 7]$ and tolerance of $1e-8$.

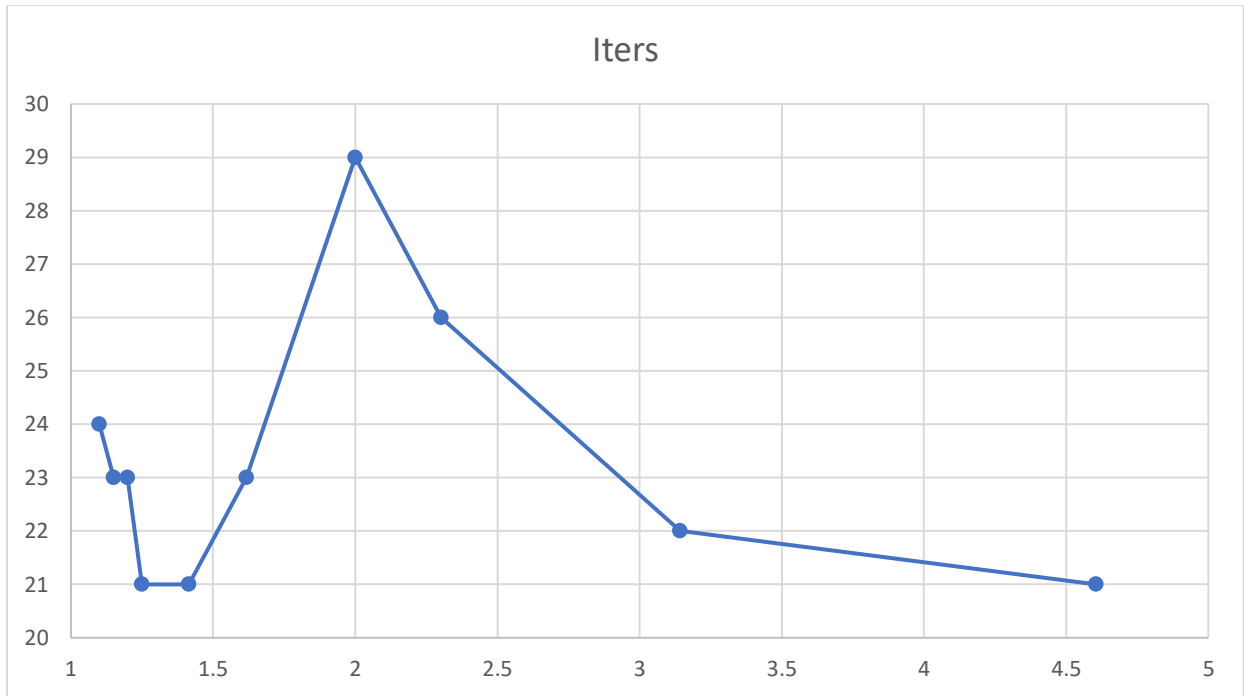


Figure 1. Results for $f(x)=\exp(x)-3*x^2$ for $[3, 7]$ and tolerance of $1e-8$.

Table 2 and Figure 1 indicate that the classical Bisection method can easily experience faster convergence when the denominator 2 in equation 1 is replaced by weights whose sum is less than 2 or greater than 2, so long these weights obey equation 2 and the implementation of the algorithm uses the If statement appearing in Listing 1.

Let’s look at how a shorter root-bracketing interval, say $[3, 4]$, affects the number of iterations for the various weights.

Table 3 shows the number of iterations for various sums of weights. Figure 2 shows the plots for the results in Table 3.

<i>W1 + W2</i>	<i>Iterations</i>
1.1	29
1.15	26
1.2	26
1.25	22
1.414213562	23
1.618033989	22
2	28

<i>W1 + W2</i>	<i>Iterations</i>
2.302585093	23
3.141592654	22
4.605170186	21

Table 3. Results for $f(x)=exp(x)-3*x^2$ for $[3, 4]$ and tolerance of $1e-8$.

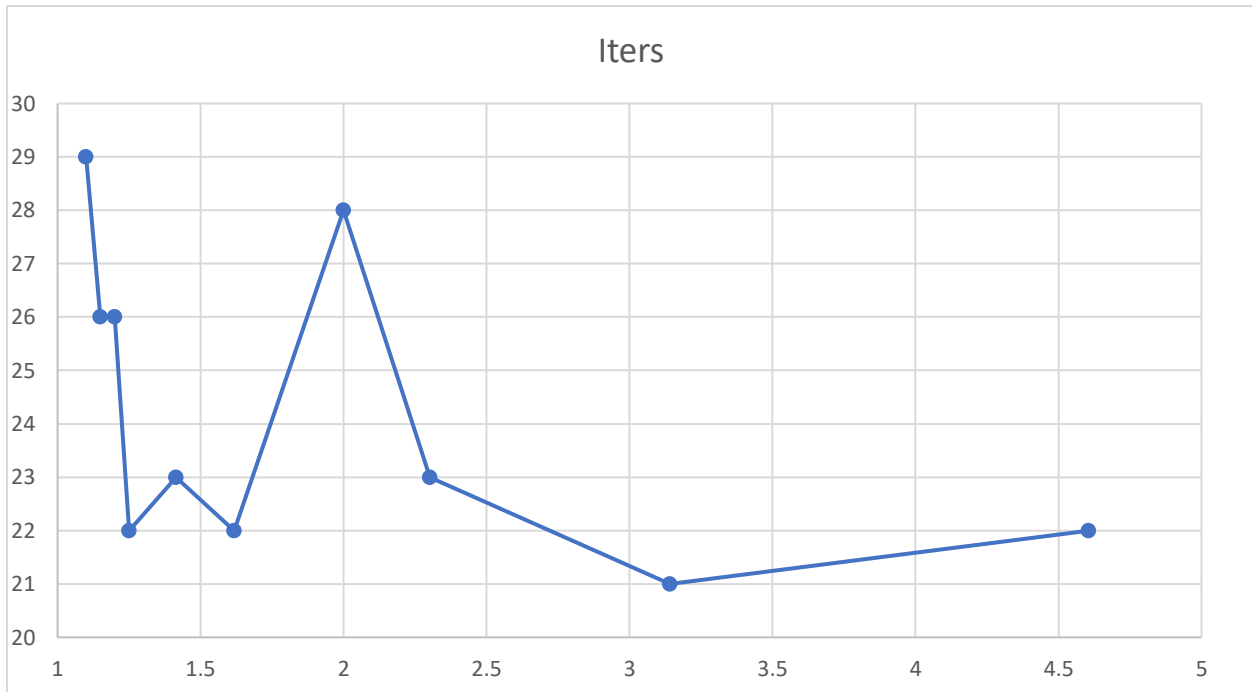


Figure 2. Results for $f(x)=exp(x)-3*x^2$ for $[3, 4]$ and tolerance of $1e-8$.

Table 3 and Figure 2 indicate that the classical Bisection method can easily experience faster convergence when the denominator 2 in equation 1 is replaced by weights whose sum is less than 2 (except for 1.1) or greater than 2, so long these weights obey equation 2 and the implementation of the algorithm uses the If statement appearing in Listing 1.

Let’s look at a different root-bracketing interval, say $[-3, 0]$, and how it affects the number of iterations for the various weights.

Table 4 shows the number of iterations for various sums of weights. Figure 3 shows the plots for the results in Table 4.

<i>W1 + W2</i>	<i>Iterations</i>
1.1	21

<i>W1 + W2</i>	<i>Iterations</i>
1.15	24
1.2	25
1.25	20
1.414213562	21
1.618033989	23
2	29
2.302585093	25
3.141592654	22
4.605170186	24

Table 4. Results for $f(x)=\exp(x)-3*x^2$ for $[-3, 0]$ and tolerance of $1e-8$.

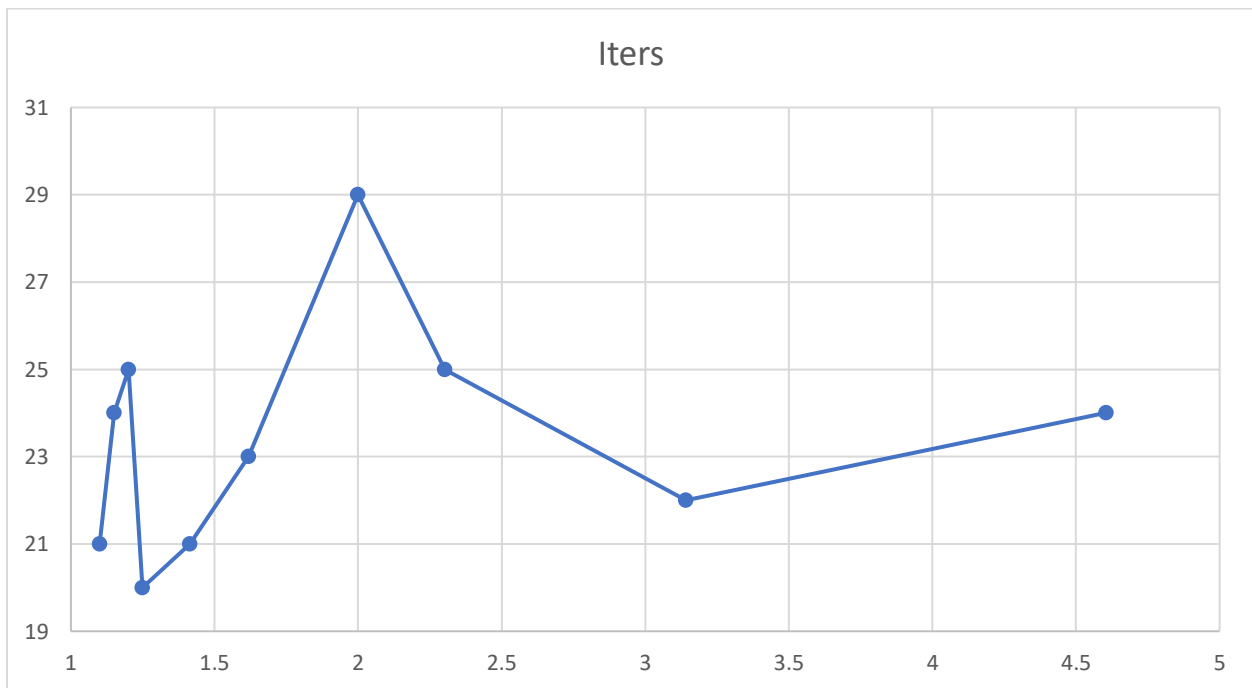


Figure 3. Results for $f(x)=\exp(x)-3*x^2$ for $[-3, 0]$ and tolerance of $1e-8$.

Table 4 and Figure 3 indicate that the classical Bisection method can easily experience faster convergence when the denominator 2 in equation 1 is replaced by weights whose sum is less than 2 or greater than 2, so long these weights obey equation 2 and the implementation of the algorithm uses the If statement appearing in Listing 1.

Let’s narrow the last root-bracketing interval to $[-1, 0]$ and see how that affects the number of iterations for the various weights.

Table 5 shows the number of iterations for various sums of weights. Figure 4 shows the plots for the results in Table 5.

<i>W1 + W2</i>	<i>Iterations</i>
1.1	40
1.15	27
1.2	22
1.25	19
1.414213562	21
1.618033989	21
2	28
2.302585093	24
3.141592654	20
4.605170186	23

Table 5. Results for $f(x)=\exp(x)-3*x^2$ for $[-1, 0]$ and tolerance of $1e-8$.

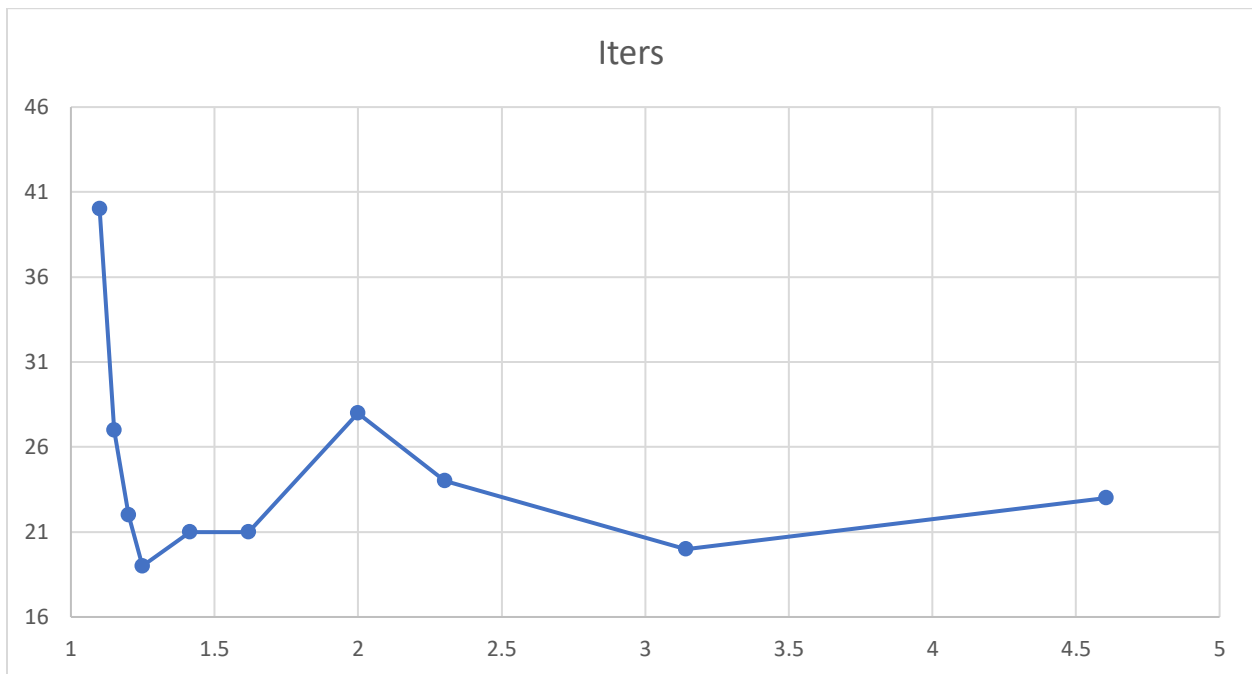


Figure 4. Results for $f(x)=\exp(x)-3*x^2$ for $[-1, 0]$ and tolerance of $1e-8$.

Table 5 and Figure 4 indicate that the classical Bisection method can easily experience faster convergence when the denominator 2 in equation 1 is replaced by weights whose sum is less than 2 (except 1.1) or greater than 2, so long these weights

obey equation 2 and the implementation of the algorithm uses the If statement appearing in Listing 1.

The conclusions we can draw that for the function $f(x)=\exp(x)-3*x^2$, the weights that sum up to $\sqrt{2}$ and π give optimum number of iterations for the sum of weights that are below 2 and above 2, respectively. The extreme sum of weights of 1.1 does not seem to reliably yield fewer iterations than the classical Bisection Method.

Testing Another Function

As stated earlier the function $f(x)=\exp(x)-3*x^2$ has roots near -0.4590 , 0.9100 , and 3.7331 . It has a maximum and a minimum. I will use a simpler linear function $f(x)=\ln(100)-x$ with a single root at 4.60517 .

Let's start with root-bracketing interval of $[0, 5]$, and how it affects the number of iterations for the various weights.

Table 6 shows the number of iterations for various sums of weights. Figure 5 shows the plots for the results in Table 6.

<i>W1 + W2</i>	<i>Iterations</i>
1.1	31
1.15	24
1.2	28
1.25	26
1.414213562	20
1.618033989	23
2	29
2.302585093	25
3.141592654	21
4.605170186	22

Table 6. Results for $f(x)=\ln(100)-x$ for $[0, 5]$ and tolerance of $1e-8$.

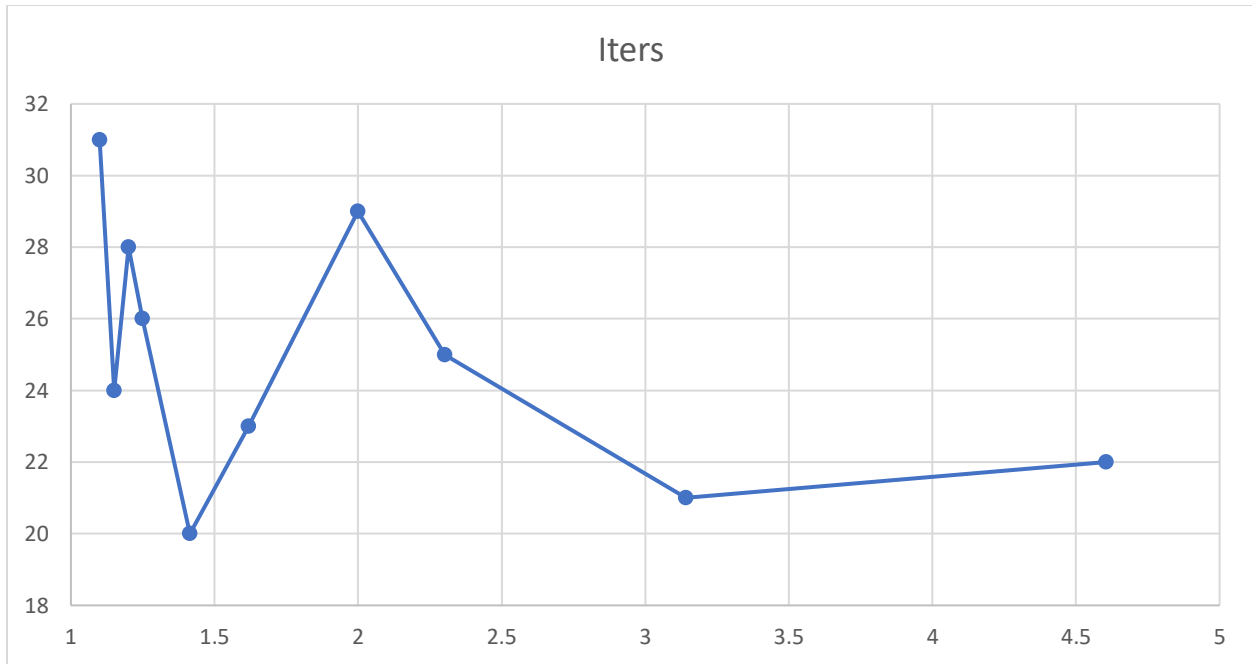


Figure 5. Results for $f(x)=\ln(100)-x$ for $[0, 5]$ and tolerance of $1e-8$.

Table 6 and Figure 5 indicate that the classical Bisection Method can easily experience faster convergence when the denominator 2 in equation 1 is replaced by weights whose sum is less than 2 (except 1.1) or greater than 2, so long these weights obey equation 2 and the implementation of the algorithm uses the If statement appearing in Listing 1.

Let’s narrow the root-bracketing interval to $[2, 5]$, and observe its effect on the number of iterations for the various weights.

Table 7 shows the number of iterations for various sums of weights. Figure 6 shows the plots for the results in Table 7.

<i>W1 + W2</i>	<i>Iterations</i>
1.1	36
1.15	24
1.2	22
1.25	21
1.414213562	21
1.618033989	22
2	29
2.302585093	24

<i>W1 + W2</i>	<i>Iterations</i>
3.141592654	23
4.605170186	23

Table 7. Results for $f(x)=\ln(100)-x$ for $[2, 5]$ and tolerance of $1e-8$.

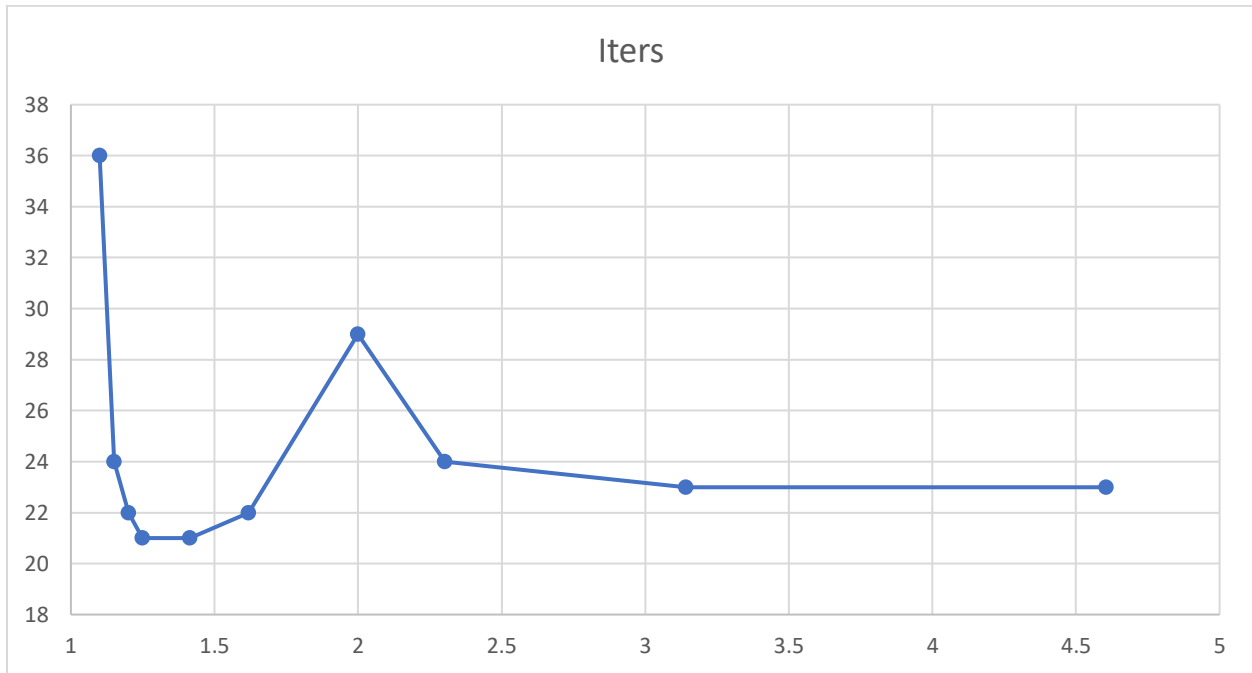


Figure 6. Results for $f(x)=\ln(100)-x$ for $[2, 5]$ and tolerance of $1e-8$.

Table 7 and Figure 6 indicate that the classical Bisection method can easily experience faster convergence when the denominator 2 in equation 1 is replaced by weights whose sum is less than 2 (except 1.1) or greater than 2, so long these weights obey equation 2 and the implementation of the algorithm uses the If statement appearing in Listing 1.

Let’s further narrow the root-bracketing interval to $[4, 5]$, and examine its effect on the number of iterations for the various weights.

Table 8 shows the number of iterations for various sums of weights. Figure 7 shows the plots for the results in Table 8.

<i>W1 + W2</i>	<i>Iterations</i>
1.1	25
1.15	31

<i>W1 + W2</i>	<i>Iterations</i>
1.2	22
1.25	25
1.414213562	19
1.618033989	22
2	27
2.302585093	24
3.141592654	23
4.605170186	18

Table 8. Results for $f(x)=\ln(100)-x$ for $[4, 5]$ and tolerance of $1e-8$.

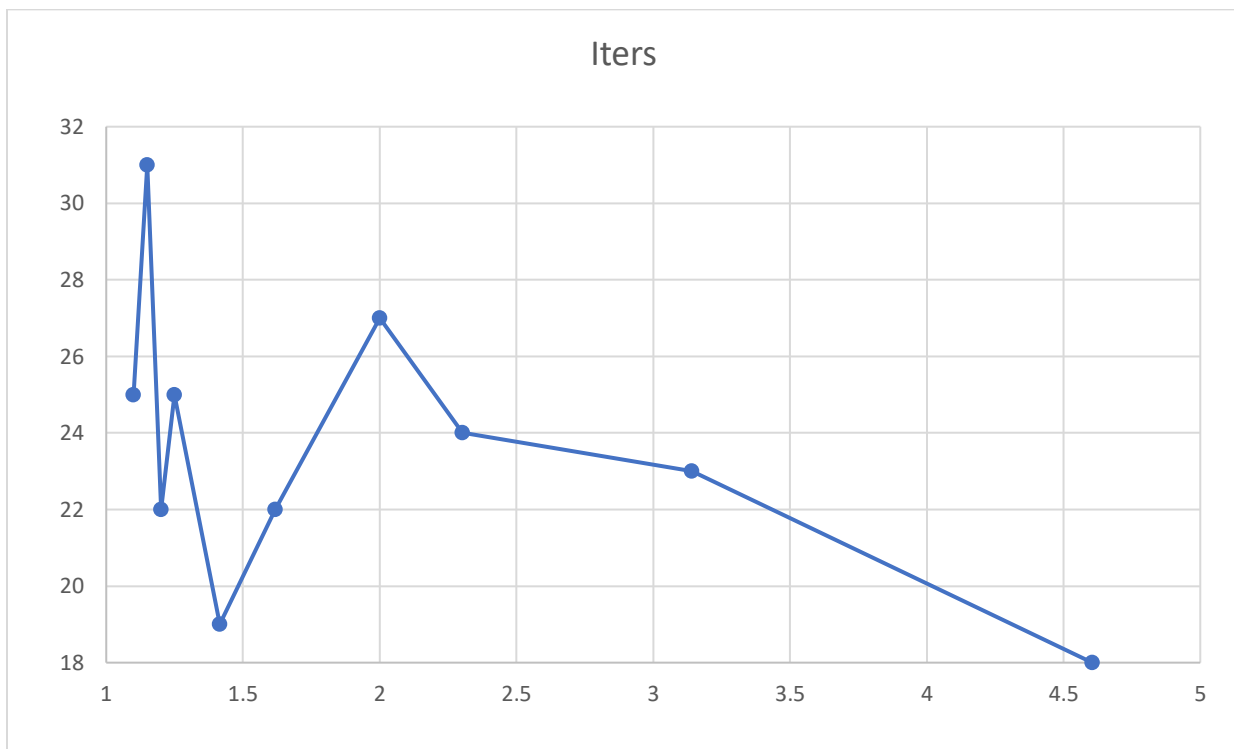


Figure 7. Results for $f(x)=\ln(100)-x$ for $[4, 5]$ and tolerance of $1e-8$.

Table 8 and Figure 7 indicate that the classical Bisection Method can easily experience faster convergence when the denominator 2 in equation 1 is replaced by weights whose sum is less than 2 (except 1.15) or greater than 2, so long these weights obey equation 2 and the implementation of the algorithm uses the If statement appearing in Listing 1.

Conclusion So Far!

Moving from the classic Bisection Method and how it uses equation 1 to refine the guess for the root to using equation 2 with the addition of the code snippet in Listing 1 pays off in reducing the number of iterations. The pseudo code for the first variant of the Bisection Method is:

Given:

- Function $f(x) = 0$.
- Root-bracketing interval $[A, B]$.
- Tolerance for the refined guess Toler
- Sum of weights $\Phi = w_1 + w_2$ and $\Phi < 2$.

```

Fa = f(A)
Fb = f(B)
Do
  If Abs(Fa) < Abs(Fb) Then
    C = (A + (Phi - 1) * B) / Phi
  Else
    C = (B + (Phi - 1) * A) / Phi
  End If
  Fc = f(C)
  If Fa * Fc > 0 Then
    A = C
    Fa = Fc
  Else
    B = C
    Fb = Fc
  End If
Loop Until Abs(A - B) < Toler

```

Listing 2. The first general variant of the Bisection Method.

The If statement in Listing 1 empowers the various weights (whose sum is not equal to 2) to reduce the number of iterations. Figures 1 to 7, show that there is a general trend favoring the sum of weights at $\sqrt{2}$ and π . The curves in these figures are similar but do not follow a strictly identical patterns. The figures are influence by the client function, the width and values if initial root-bracketing interval, and the tolerance value. While these factors are somewhat few, there is much complexity associated with them to enable me to derive an equation predicting the number of iterations

Twaking the Bisection when the Sum of Weights is 2

This part of this study looks back at the classic Bisection Method and inquires about tweaking the sum of the weights w_1 and w_2 is 2 but the weights are not equal to 1. The pseudo-code for this second general variation for the Bisection Method is: Given:

- Function $f(x) = 0$.
- Root-bracketing interval $[A, B]$.
- Tolerance for the refined guess Toler
- $0 < \text{Phi} < 1$.

```

Fa = f(A)
Fb = f(B)
Do
  If Abs(Fa) < Abs(Fb) Then
    C = (Phi * B + (2 - Phi) * A) / 2
  Else
    C = (Phi * A + (2 - Phi) * B) / 2
  End If
  Fc = f(C)
  If Fa * Fc > 0 Then
    A = C
    Fa = Fc
  Else
    B = C
    Fb = Fc
  End If
Loop Until Abs(A - B) < Toler

```

Listing 3. The second general variant for the Bisection Method.

The code in Listing 3 shows that the value of the inferior end point is multiplied by Phi (which is less than 1) while the superior end point is multiplied by $(2 - \text{Phi})$ setting its value above 1.

I decided to vary the values of Phi between 0.1 and 0.9 in increments of 0.1. I want to answer the following questions:

1. Does any value of $\text{Phi} < 2$ yield few iterations?
2. If (1) is true, what are the best values of Phi?
3. Is there a systematic pattern that appears regardless of the size and values of the root-bracketing interval and of the targeted function?

Let's start with $f(x)=\exp(x)-3*x^2$ and the initial root-bracketing interval of $[3, 7]$, with the tolerance value of $1e-8$.

Table 9 shows the number of iterations for various sums of weights. Figure 8 shows the plots for the results in Table 9.

<i>W1 + W2</i>	<i>Iterations</i>
0.1	33
0.2	24
0.3	18
0.4	21
0.5	24
0.6	22
0.7	22
0.8	24
0.9	26
1 (classic Bisection)	29

Table 9. Results for $f(x)=\exp(x)-3*x^2$ for $[3, 7]$ and tolerance of $1e-8$.

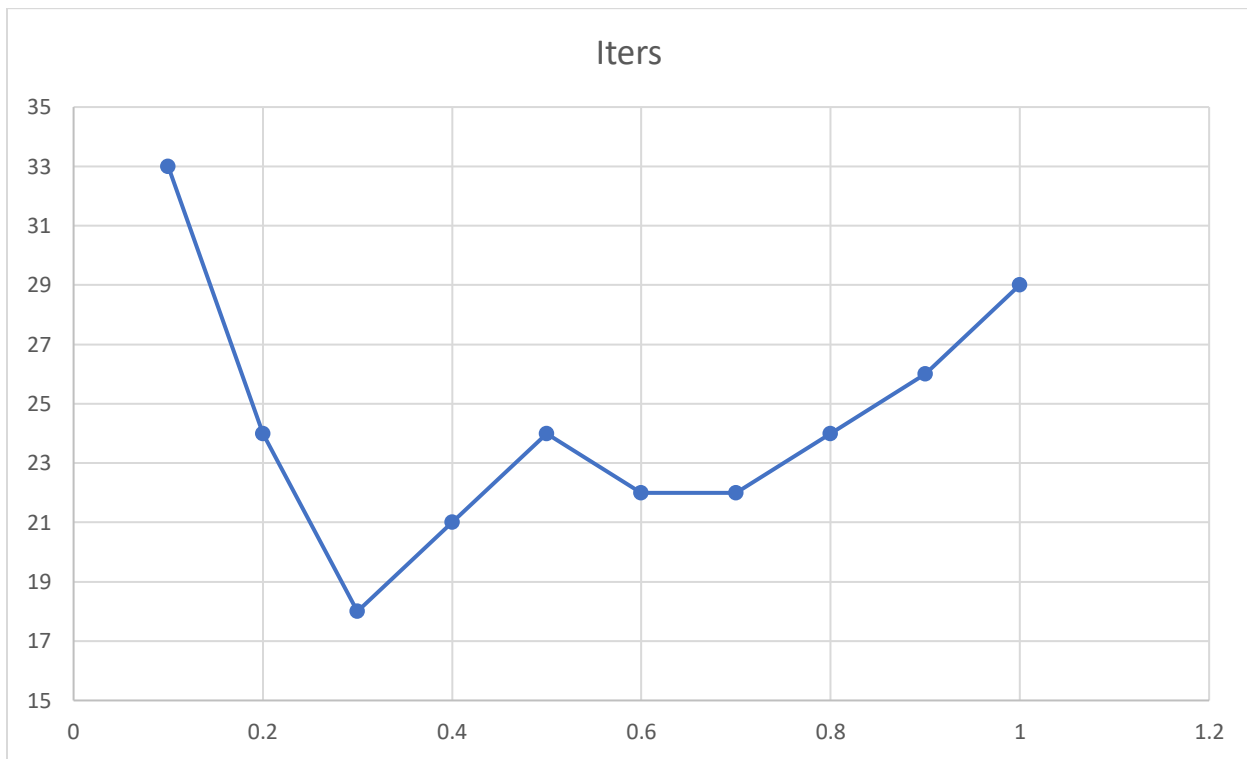


Figure 8. Results for $f(x)=\exp(x)-3*x^2$ for $[3, 7]$ and tolerance of $1e-8$.

Table 9 and Figure 8 indicate that the classical Bisection Method can easily experience faster convergence for values of Phi between 0.2 and 0.9. The value of Phi=0.3 is the optimum values in Figure 8.

Let's shorten the initial root-bracketing interval to [3, 4] and see how that affects the number of iterations and their distribution.

Table 10 shows the number of iterations for various sums of weights. Figure 9 shows the plots for the results in Table 10.

<i>W1 + W2</i>	<i>Iterations</i>
0.1	23
0.2	29
0.3	27
0.4	21
0.5	23
0.6	20
0.7	20
0.8	22
0.9	24
1 (classic Bisection)	27

*Table 10. Results for $f(x)=\exp(x)-3*x^2$ for [3, 4] and tolerance of $1e-8$.*

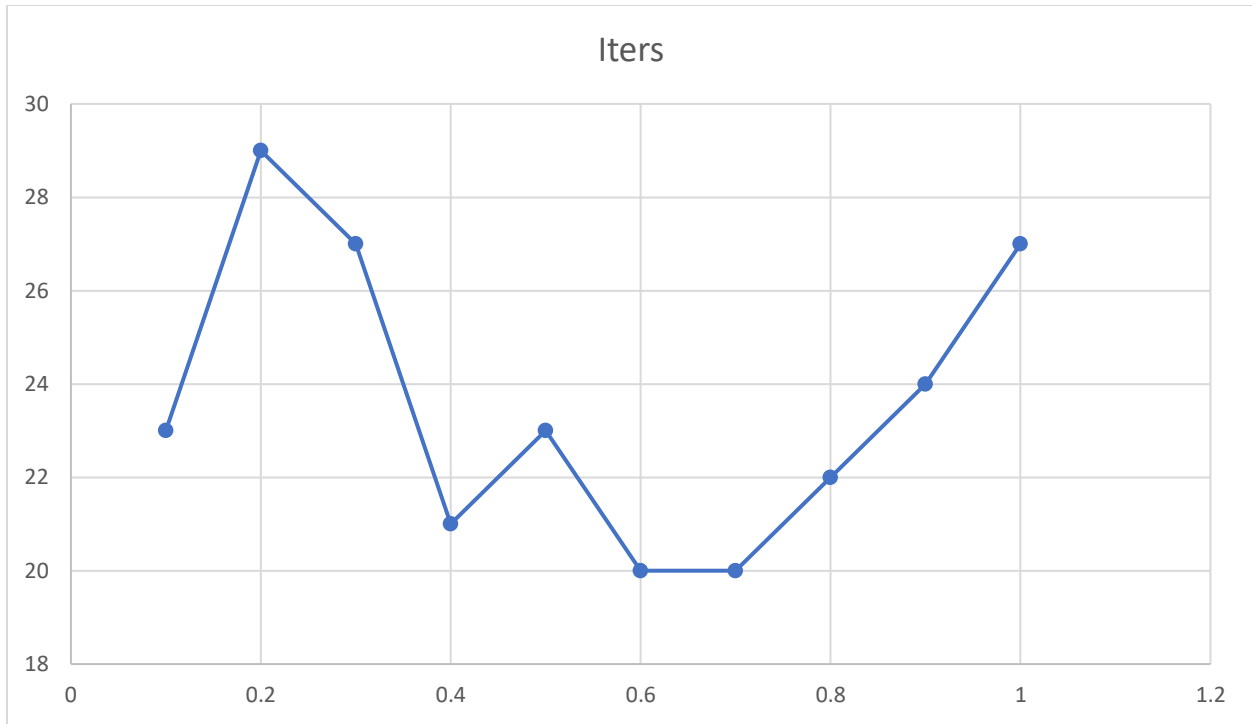


Figure 9. Results for $f(x)=exp(x)-3*x^2$ for $[3, 4]$ and tolerance of $1e-8$.

Table 10 and Figure 9 indicate that the classical Bisection Method can still easily experience faster convergence for values of Phi other than 0.2. The value of Phi=0.6 and 0.7 are the optimum values in Figure 9. Notice that the distribution of the iteration numbers is different in Figures 8 and 9, with the optimum Phi shifting from below 0.5 to above it!

Now we look at $f(x)=ln(100)-x$ with an initial root-bracketing interval of $[0, 5]$.

Table 11 shows the number of iterations for various sums of weights. Figure 10 shows the plots for the results in Table 11.

<i>W1 + W2</i>	<i>Iterations</i>
0.1	58
0.2	24
0.3	23
0.4	26
0.5	21
0.6	22
0.7	24

<i>W1 + W2</i>	<i>Iterations</i>
0.8	23
0.9	26
1 (classic Bisection)	29

Table 11. Results for $f(x)=\ln(100)-x$ for $[0, 5]$ and tolerance of $1e-8$.

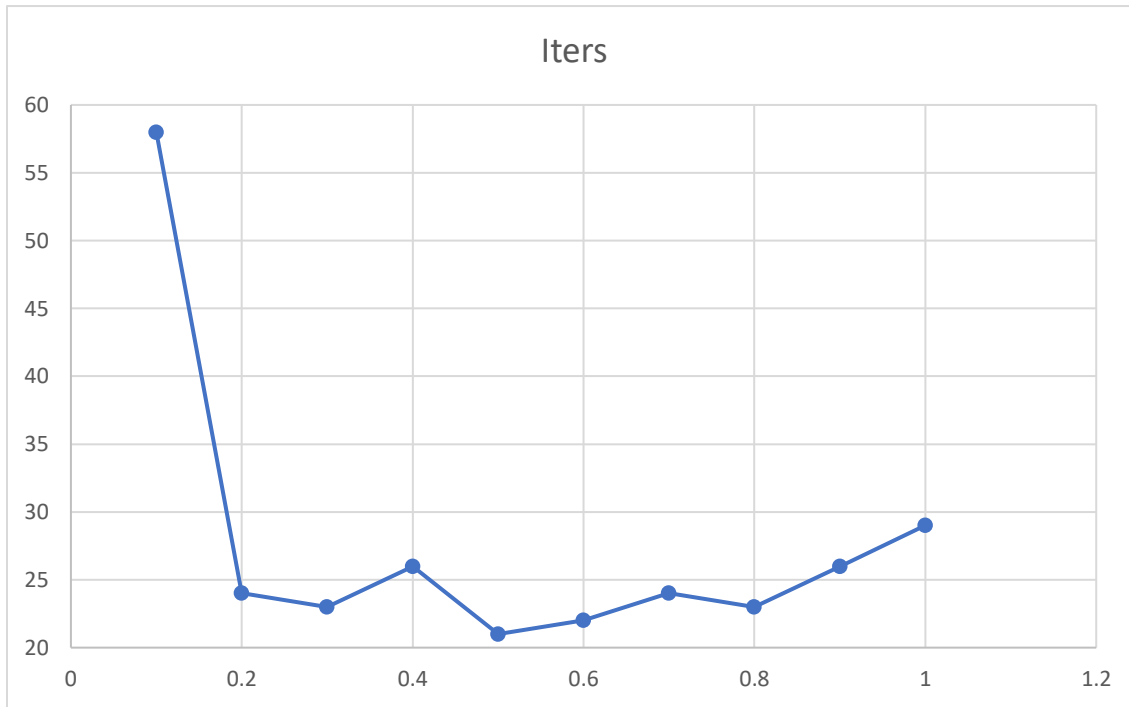


Figure 10. Results for $f(x)=\ln(100)-x$ for $[0, 5]$ and tolerance of $1e-8$.

Table 11 and Figure 10 indicate that the classical Bisection Method can still easily experience faster convergence for values of Phi other than 0.1. The value of Phi=0.5 is the optimum values in Figure 9.

Now we shorten the initial root-bracketing interval to $[4, 5]$ and examine its effect on the number of iterations.

Table 12 shows the number of iterations for various sums of weights. Figure 11 shows the plots for the results in Table 12.

<i>W1 + W2</i>	<i>Iterations</i>
0.1	34
0.2	29
0.3	25

$W1 + W2$	<i>Iterations</i>
0.4	25
0.5	19
0.6	19
0.7	22
0.8	22
0.9	24
1 (classic Bisection)	27

Table 12. Results for $f(x)=\ln(100)-x$ for $[4, 5]$ and tolerance of $1e-8$.

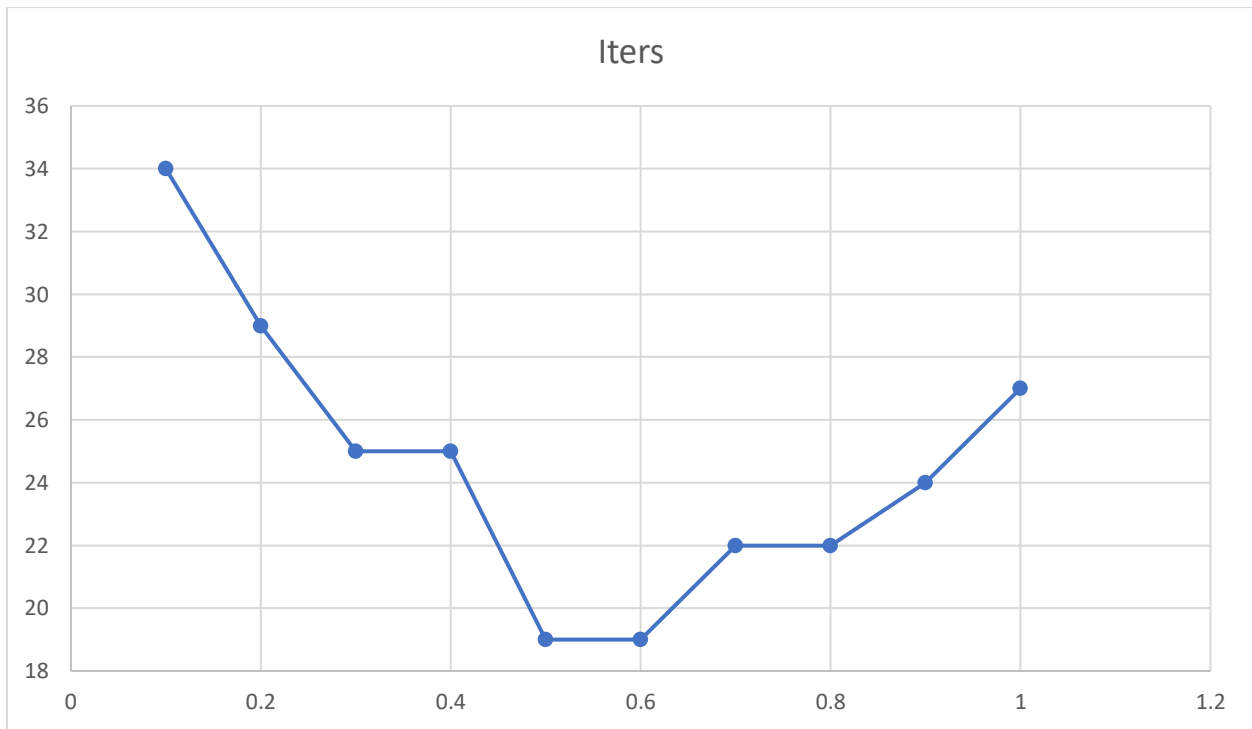


Figure 11. Results for $f(x)=\ln(100)-x$ for $[4, 5]$ and tolerance of $1e-8$.

Table 12 and Figure 11 indicate that the classical Bisection Method can still easily experience faster convergence for values of Phi other than 0.1. The values of Phi=0.5 and 0.6 are the optimum values in Figure 11. Again, we see that narrowing the root-bracketing interval yields a different distribution for the number of iterations.

The conclusion that we can draw is that there is no firm value of Phi that is minimum. Taking the value of 0.55 (as an average of 0.5 and 0.6) is a good guess.

Dynamic Split of Sum of Weights

Now we come back to the other method for splitting the sum of weights—the one that uses the absolute function values at A and B. The new scheme replaces the If statement in Listing 1 with the one in Listing 4.

```
If AbsFa < AbsFb Then
    w = AbsFb / (AbsFa + AbsFb)
    C = (w * B + (Phi - w) * A) / Phi
Else
    w = AbsFa / (AbsFa + AbsFb)
    C = (w * A + (Phi - w) * B) / Phi
End If
```

Listing 4. The scheme to split the sum of weights using relative function values.

Listing 4 shows that the code calculates the variable w as a ratio between an absolute function value for A or B with the sum of the absolute function values at A and B. The If statement calculates C using the values of A, B, w , and Φ . The last two entities calculate the values of the weights w_1 and w_2 (as they appear in equation 2). One big feature of this scheme is that it implements dynamic splitting of the sum of weights. Each iteration will calculate a *different* value for w , which also makes $\Phi - w$ a variable entity.

Let’s look at how using the dynamic splitting scheme in Listing 4 works with finding the root for $f(x)=\exp(x)-3*x^2$ for interval to $[3, 4]$. Table 13 shows the number of iterations for various sums of weights. Figure 12 shows the plots for the results in Table 12.

<i>W1 + W2</i>	<i>Iterations</i>
1.2	53
1.25	53
1.414213562	44
1.618033989	37
2	32
2.302585093	29
3.141592654	25
4.605170186	26
5	30

*Table 13. Results for $f(x)=\exp(x)-*x^2$ for $[3, 4]$ and tolerance of $1e-8$.*

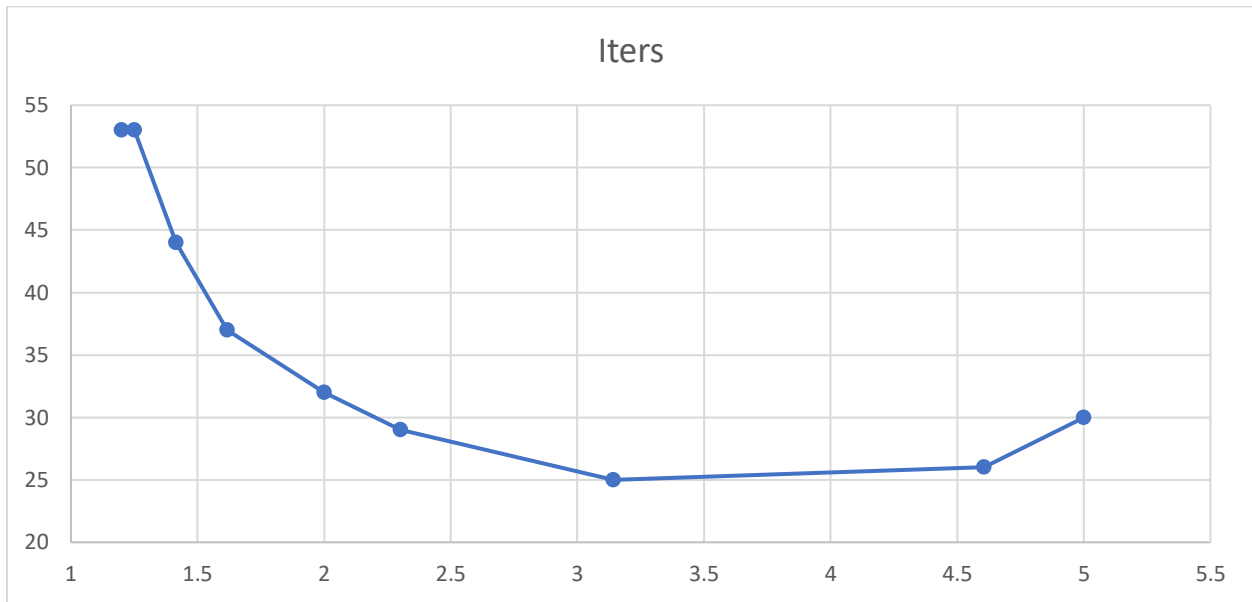


Figure 12. Results for $f(x)=exp(x)-x^2$ for $[3, 4]$ and tolerance of $1e-8$.

Table 13 and Figure 12 indicate that using the scheme in Listing 5 produces different convergence dynamics that in the previous parts of this study. Only the values of Phi above 2 yields fewer iterations than the classical Bisection Method. Notice that there seems to be an upward trend that yields more iterations at Phi=6 or above (I have tested these values separately).

Now we look at $f(x)=ln(100)-x$ with an initial root-bracketing interval of $[4, 5]$. Table 14 shows the number of iterations for various sums of weights. Figure 13 shows the plots for the results in Table 14.

<i>W1 + W2</i>	<i>Iterations</i>
1.2	52
1.25	37
1.414213562	38
1.618033989	31
2	29
2.302585093	29
3.141592654	23
4.605170186	20
5	27

Table 14. Results for $f(x)=ln(100)-x$ for $[4, 5]$ and tolerance of $1e-8$.

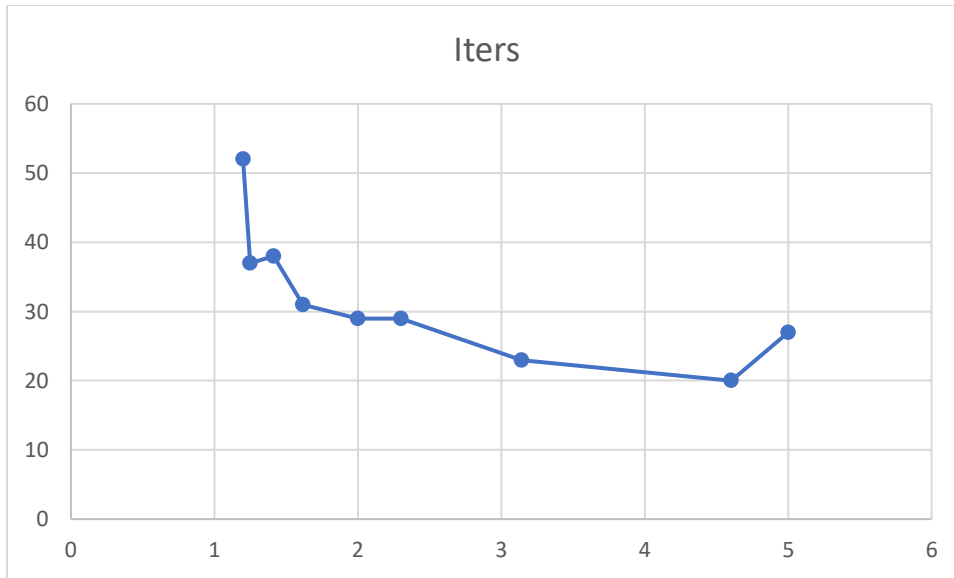


Figure 13. Results for $f(x)=\ln(100)-x$ for $[4, 5]$ and tolerance of $1e-8$.

Table 14 and Figure 13 indicate that the values for Phi above pi converge faster than the classical Bisection method. However, like in Figure 12, this reduction exists for a limited range. The number of iterations increases as Phi exceeds 5.

Using the dynamic splitting scheme in Listing 4 yields limited improvements for the number of iteration. This scheme is not as good as the manual weight selection schemes presented earlier.

General Conclusion

This study looks at using equation 2 to modify the classic Bisection Method. The denominator 2 in equation 1 can be regarded as a sum of weights used to multiply the ends, $[A, B]$, of the root-bracketing interval. The study looked at replacing 2 with other numbers as well as using it. Another design factor in fine-tuning the Bisection Method is whether to manually select the two weights w_1 and w_2 or to calculate them using the absolute function values of A and B. Calculating the weight turned out to offer limited improvements for the Bisection Method. Simple manual selection of the weights, whose sums differ from 2, yielded good reduction in the number of iterations.

Taking equation 2 in consideration, it seems that using the sum of weights w_1 and w_2 to be either $\sqrt{2}$ or π , together with the code in Listing 2 yields the most reduction in the number of iterations, compared with the classical Bisection Method.

Looking at equation 2 again:

$$C = (w_1 A + w_2 B) / (w_1 + w_2)$$

You can select any positive sum of weights and split that sum into reasonable proportional values for w_1 and w_2 . The value of these weights influences the rate of conversion to the root.

Listing 5 shows the general pseudo-code for the modified Bisection Method:

Given:

- Function $f(x) = 0$.
- Root-bracketing interval $[A, B]$.
- Tolerance for the refined guess Toler
- Weights w_1 and w_2 , such that $w_1 + w_2 \geq 2$ and $w_1 > w_2$.

```

Fa = f(A)
Fb = f(B)
Do
  If Abs(Fa) < Abs(Fb) Then
    C = (w1 * A + w2 * B) / (w1 + w2)
  Else
    C = (w1 * B + w2 * A) / (w1 + w2)
  End If
  Fc = f(C)
  If Fa * Fc > 0 Then
    A = C
    Fa = Fc
  Else
    B = C
    Fb = Fc
  End If
Loop Until Abs(A - B) < Toler

```

Listing 5. The general version of the fixed weights split Bisection Method modification.

The classical Bisection Method is the simplest form of root-bracketing algorithm that reduces the root-bracketing interval by half for each iteration. This study has show that equation 1 (used in the classical Bisection Method) can be replaced by a more general version in equation 2. The improvement reduces the number of iterations using special weights w_1 and w_2 that are multiplied by the root-bracketing

interval ends. It also requires including an If-Else statement (like in Listing 5) to effectively use these weights.

This study merely opens the door for a more thorough study of equation 2 taking in consideration a vast number of combination of weights w_1 and w_2 .

Document History

<i>Date</i>	<i>Version</i>	<i>Comment</i>
May 30, 2018	1.0.0	Initial release.