Roots of Equation

Learning Outcomes

At the end of this session, the students should be able to:

1. understand the concept of false position, secant and mullers' method in root finding;
2. differentiate the algorithms used in false position, secant and mullers' method in root finding;
3. create an R script that will automate false position or secant and mullers' method in root finding;

Content

I. Root-Finding: A Review of Prerequisites

The root of a function $f(x),\ (f : \mathbb{R} \rightarrow \mathbb{R})$ is a value $r \in \mathbb{R}$, for which the value of the function when evaluated is zero (0). It can also be said that $f(r) = 0$. For root-finding, we shall focus on finding roots of equations of a single variable.

II. False Position Method

The False Position Method, also known as *regula falsi*, is a root-finding method where roots is derived from the equation of the line formed by the limits of the current interval of the iteration, as well as their evaluations over the function. This method is an example of closed method where inputs must bracket the desired root of the function.

A line can be formed by using the points $(a, f(a))$ and $(b, f(b))$, and thus we will employ the point-slope form of linear equations, which has the formula $y - y_1 = m\,(x - x_1)$. Since we are given two points, we need to compute for the slope first.

As a recap, we can find the slope of the line by using $m = \frac{y_1 - y_2}{x_1 - x_2}$. Plugging the values above, we can have the following formula:

$$y - f(b) = \frac{f(b) - f(a)}{b - a}\,(x - b).$$

A graph of the line can already be made, and as a result, two right triangles will be formed, as shown below in Figure 1. These two triangles are similar, since they are both right triangles. Therefore, a proportion can be made in these two triangles, because of the Angle-Angle-Angle (AAA) similarity theorem.

The proportion will be done by the following:

$$\frac{f(a)}{c - a} = \frac{f(b)}{c - b},$$

after which, solving for c will yield

$$c = \frac{b \cdot f(a) - a \cdot f(b)}{f(a) - f(b)}.$$

This will be used in finding the next potential root of the function.

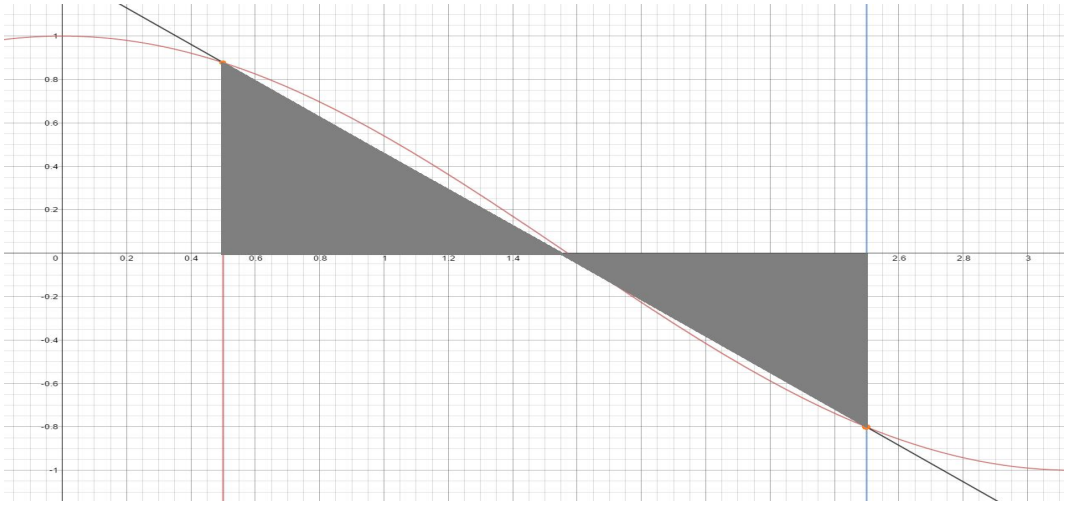

Figure 1. Graph of $cos(x)$, in the interval $x \in [0.5, 2.5]$. Two right triangles are shown in the bounds of these intervals.

Algorithm

The False Position Method are going to bound the solution on a given interval until it reaches a terminating condition. Given a function $f(x)$, we must find c which is between the values a and b inclusive, such that $f(c) = 0$.

Algorithm 1. False Position Method

```
FalsePositionMethod(f(x) [the function to root-find],
          a [lower limit],
          b [upper limit],
          macheps [machine epsilon],
          max [maximum number of iterations]){
    if(f(a) * f(b) < 0) //interval does bound the root
        while(ea >= macheps AND num_iterations != max)
            c = (b * f(a) - a * f(b)) / (f(a) - f(b))
            if(f(c) = 0) return c
            if(f(c) < 0 AND f(a) > 0) b = c
            else a = c
            if num_iterations != 0
                ea = | (c - c_old) / c | * 100
            num_iterations += 1
    return c, num_iterations, ea
}
```

Example

If we are to consider the function $f(x) = cos(x)$, let us compute for the root of $f(x)$ at the interval $[0, 2]$. Let us use 1×10^{-5} as our machine epsilon. (In this example, the numbers are formatted to R's options(digits=4) command.)

a	b	f(a)	f(b)	c	f(c)	Update	Error (%)
0	2	1	-0.4161	1.412	0.1579	a = c	

1.412	2	0.1579	-0.4161	1.574	-0.00311	b = c	10.27
1.412	1.574	0.1579	-0.00311	1.571	1.28e-05	a = c	0.1988
1.571	1.574	1.28e-05	-0.00311	1.571	-2.056e-11	b = c	0.0008152
1.571	1.571	1.28e-05	-2.056e-11	1.571	6.123e-17	a = c	1.309e-9

After doing the false position method to the function in 5 iterations, we have found out that the approximate value of the root of $f(x)$ is $x = 1.571$, in the machine epsilon of 1×10^{-5}. If we define our machine epsilon to be smaller, as well as the output format of R, we can get a well-defined root.

Consequences

In this algorithm, as the interval get closer by the reassignment of the variable c, the zero of the line decides where to place the next interval. Because of this, if the function is asymptotic, such as $f(x) = ln\,(x)$, convergence might be slower.

III. Secant Method

The false position method forms a secant line which has the points of the interval that the root is bracketed upon. In this method, it also forms a secant line, but a new number is used to approximate the next potential root of the function, and it is also being used to form a new secant line on the next iteration. Secant method is an open method where inputs are not necessarily bracket the desired root of the function.

Theory

The secant method requires two initial approximations for the root of $f(x)$, which will be referred to as x_0 and x_1. These variables should not necessarily bracket the root, but it should be close to the actual root. We will then have the initial function evaluations for these two variables creating two points: $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

We can now construct a secant line by using the point-slope form:

$$y - f(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1).$$

Since we are finding the roots of the function, we will set y to zero, then we will solve for x. Doing so will yield:

$$x = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}.$$

We will then use the above formula to approximate for the next approximation of the root, x_2. Doing so will yield:

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}.$$

We shall continue this process in finding the next approximation of the root, replacing the current approximation with the previous one. In the above equation, we shall find x_3, replacing x_0 with x_1 and x_1 with x_2. Doing so will yield:

$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)}.$$

Finding a general formula for the above equation in the n^{th} iteration will yield the following final formula:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Algorithm

Algorithm 2. Secant Method

```
SecantMethod(f(x) [the function to root-find],
          x_0 [first approximate],
          x_1 [second approximate],
          macheps [machine epsilon],
          max [maximum number of iterations]){
    y0 = f(x_0)
    y1 = f(x_1)
    while(ea >= macheps AND num_iterations != max)
        x = x_1 - (x_1 - x_0) * y1 / (y1 - y0)
        y = f(x)
        if(f(x) = 0) return x
        if num_iterations != 0
            ea = | (x - x_old) / x | * 100
        x0 = x1
        y0 = y1
        x1 = x
        y1 = y
        num_iterations += 1
    return x, num_iterations, ea
}
```

Example

If we are to consider the function $f(x) = cos(x)$, let us compute for the root of $f(x)$ with the initial guesses of 0 and 2. Let us use 1×10^{-5} as our machine epsilon. Take note what happens if the initial guesses bracket the root.

Iteration	x_0	x_1	$f(x_0)$	$f(x_1)$	x	f(x)	Error (%)
1	0	2	1	-0.4161468	1.412283	0.1578504	
2	2	1.412283	-0.4161468	0.1578504	1.573906	-0.00311	10.26893
3	1.412283	1.573906	0.1578504	-0.00311	1.570784	1.280485e-5	0.1988054
4	1.573906	1.570784	-0.00311	1.280485e-5	1.570796	-2.0557e-11	0.00081518
5	1.570784	1.570796	1.280485e-5	-2.0557e-11	1.570796	6.12303e-17	1.308678e-9

After doing the secant method to the function in 5 iterations, we have found out that the approximate value of the root of $f(x)$ is $x = 1.570796$, in the machine epsilon of 1×10^{-5}. If we define our machine epsilon to be smaller, as well as the output format of R, we can get a well-defined root.

Consequences

Since the root is not necessarily bracketed in the secant method, there is a chance that the root may not be found based on the given initial approximations. A periodic function which always has multiple zeros on a defined interval may rectify this issue.

Let it be clarified that when a method converges into machine epsilon, it does not always mean that the root was found. The false position method always converges into a final value since it always relies on the bracketed values.

In the secant method, the iteration might not end if the function has no root.

IV. Muller's Method

The Muller's method is another root-finding algorithm which can be used when the derivative of the function is unsolvable. This is a generalization of the secant method, but instead of using two points to find a secant line, it uses three points to find a quadratic polynomial.

Theory

We are given three numbers as our initial approximations of the root, namely x_0, x_1, and x_2. Since x_2 will be our given consequent root, we shall create a quadratic equation which will use that particular variable. As you may recall, the general equation of a polynomial of degree two is:

$$f(x) = A(x - x_2)^2 + B(x - x_2) + C.$$

Since we already have our polynomial, we need our approximations and their respective function evaluations. To do so, we need to derive the coefficients of the parabola which passes through the points $(x_0, f(x_0))$, $(x_1, f(x_1))$ and $(x_2, f(x_2))$. Doing so will yield the following system of equations.

$$f(x_0) = A(x_0 - x_2)^2 + B(x_0 - x_2) + C$$
$$f(x_1) = A(x_1 - x_2)^2 + B(x_1 - x_2) + C$$
$$f(x_2) = A(x_2 - x_2)^2 + B(x_2 - x_2) + C$$

Looking at the system, we can directly solve for $f(x_2)$, which can be simplified to C. We can substitute $f(x_2)$ to $f(x_0)$ and $f(x_1)$, which will then produce the following system:

$$f(x_0) - f(x_2) = A(x_0 - x_2)^2 + B(x_0 - x_2)$$
$$f(x_1) - f(x_2) = A(x_1 - x_2)^2 + B(x_1 - x_2)$$

We can then define the following differences to be used in the above equations.

$$h_0 = x_1 - x_0, \qquad d_0 = \frac{f(x_1) - f(x_0)}{h_0}$$
$$h_1 = x_2 - x_1, \qquad d_1 = \frac{f(x_2) - f(x_1)}{h_1}$$

Substituting these into our previous equations and solving for the system, we can get the following:

$$h_0 d_0 + h_1 d_1 = (h_0 + h_1)B - (h_0 + h_1)^2 A$$
$$h_1 d_1 = h_1 B - h_1^2 A$$

We can now solve for the value of A and B, after substitution. We will get $A = \frac{d_1 - d_0}{h_1 + h_0}$, $B = Ah_1 + d_1$, and $C = f(x_2)$.

Since we now have an equation of a parabola, we can now find the root of the said parabola by employing the quadratic formula. To recall, the roots of a second-degree polynomial with a general formula $f(x) = Ax^2 + Bx + C$, when $x \neq 0$ can be defined as: $x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$, and $x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$.

We can then rationalize the numerator for us to obtain a more accurate approximation of the roots. Doing so will involve the following steps:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}\left(\frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}}\right)$$

$$x_1 = \frac{b^2 - b^2 - 4ac}{2a\,(-b - \sqrt{b^2 - 4ac})}$$

$$x_1 = \frac{-4ac}{2a\,(-b - \sqrt{b^2 - 4ac})}$$

$$x_1/x_2 = \frac{-2c}{b \pm \sqrt{b^2 - 4ac})}\,.$$

It can be observed that there are two roots of the above parabola. The root we are interested is in the one that is close to x_2. We can now find the next approximate by doing the following:

$$x_3 - x_2 = \frac{-2c}{b \pm \sqrt{b^2 - 4ac})}$$

$$x_3 = x_2 - \frac{2c}{b \pm \sqrt{b^2 - 4ac})}$$

Since we are finding x_3, which is intended to be closer to the root, we will use the sign in the equation below which agrees with the discriminant. We are to evaluate the following condition:

$$\left|b - \sqrt{b^2 - 4ac}\right| < \left|b + \sqrt{b^2 - 4ac}\right|.$$

If the above condition is true, we are to employ the positive sign form of x_3. Otherwise, we shall employ its negative sign form.

Lastly, we will reassign the following variables: $x_0 = x_1$, $x_1 = x_2$, and $x_2 = x_3$. Error checking can be done by getting the approximate error of x_3 and x_2, since they are the later approximates in finding the root. After this, we can restart the iteration.

Algorithm

The Muller's Method is an open method. This means that the inputs may not necessarily bracket the desired root of the function. For implementation purposes, it may be best to create functions for the computations of the variables being used.

Algorithm 3. Muller's Method

```
MullerMethod(f(x) [the function to root-find],
          x_0 [first approximate],
          x_1 [second approximate],
          x_2 [third approximate],
          macheps [machine epsilon],
          max [maximum number of iterations]){
    while(ea >= macheps AND num_iterations != max)
        y0 = f(x_0)
        y1 = f(x_1)
        y2 = f(x_2)
        compute h_0 , d_0 , h_1 , d_1
        compute A , B , C
```

```
        sign = ( |b − √(b² − 4ac)| < |b + √(b² − 4ac)| ) ? positive : negative
        x3 = (sign = positive) ? positive form : negative form
        ea = |(x3 - x2) / x3| * 100
        x0 = x1
        x1 = x2
        x2 = x3
    return x0, x1, x2, x3, num_iterations, ea
}
```

Example

If we are to consider the function $f(x) = cos(x)$, let us compute for the root of $f(x)$ with the initial guesses of 0, 2 and 4. Let us use 1×10^{-5} as our machine epsilon. Take note what happens if the initial guesses bracket the root. (In this example, the numbers are formatted to R's options(digits=3) command due to space constraints.)

x_0	x_1	x_2	f(x)	$f(x_1)$	$f(x_2)$	A	B	C	x_3	$f(x_3)$	Error
0	2	4	1	-0.416	-0.654	0.147	0.176	-0.654	5.59	0.771	28.5
2	4	5.59	-0.416	-0.654	0.771	0.282	1.34	0.771	4.93	0.211	13.5
4	5.59	4.93	-0.654	0.771	0.211	-0.0604	0.879	0.211	4.69	-0.0237	5.05
5.59	4.93	4.69	0.771	0.211	-0.0237	-0.171	1.03	-0.0237	4.71	-0.000686	0.488
4.93	4.69	4.71	0.211	-0.0237	-0.000686	-0.0314	0.999	-0.000686	4.71	5.75e-07	0.0146
4.69	4.71	4.71	-0.0267	-0.000686	5.75e-07	0.00406	1	5.75e-07	4.71	1.56e-12	1.22e-05
4.71	4.71	4.71	-0.000686	5.75e-07	1.56e-12	0.000114	1	1.56e-12	4.71	-1.84e-16	3.3e-11

After doing the Muller's method to the function in 7 iterations, we have found out that the approximate value of the root of $f(x)$ is $x = 4.71$, in the machine epsilon of 1×10^{-5}. If we define our machine epsilon to be smaller, as well as the output format of R, we can get a well-defined root.

Consequences

When computed manually, due to the presence of the square root function, it can be observed that the method can find complex roots, that is when the discriminant is -1. Algorithmically, if your programming language supports complex numbers, the result may also be displayed.

Learning Experiences

1. Students will solve manually for the roots of equation using the 3 methods.
2. Students will automate the 3 methods discussed on finding the roots of equation using R.
3. Students will attempt to accomplish ***sample exercises for self-learning*** provided below.

Sample Exercises for Self-learning

1. *Required Competencies*: False Position Method, functions, loops
   Create two R functions named **FalsePositionMethod** with the following variables as parameters:
   - **f,** a variable which handles an arbitrary mathematical function;
   - **a**, a numerical variable for the lower limit of the interval;

- **b**, a numerical variable for the upper limit of the interval;
- **macheps**, a numerical variable for the machine epsilon;
- **max**, a numerical variable for the maximum number of iterations, with 1000 as default; and
- **verbose**, an optional Boolean value if the whole process is to be printed. Default value is TRUE.

The function should perform the prescribed method operating on the above variables, and must return a list which contains the values of the following variables:

- **f**
- **given_a**
- **given_b**
- **c**, the converged value
- **iterations**, the iteration count when the method converged
- **ea**, the approximate error of the last iteration.

If the verbose variable is set as TRUE, the function should print the values of a, b, c and their respective evaluations over the function f, as well as the approximate error for c, for each iteration.

2. *Required Competencies*: Secant Method, functions, loops
   Create an R function named **SecantMethod** with the following variables as parameters:

- **f**, a variable which handles an arbitrary mathematical function;
- **x0**, a numerical variable for the first root approximate;
- **x1**, a numerical variable for the second root approximate;
- **macheps**, a numerical variable for the machine epsilon;
- **max**, a numerical variable for the maximum number of iterations; and
- **verbose**, an optional Boolean value if the whole process is to be printed. Default value is TRUE.

The function should perform the prescribed method operating on the above variables, and must return a list which contains the values of the following variables:

- **f**
- **given_x0**
- **given_x1**
- **x**, the current value of x at the current iteration
- **iterations**, the current iteration count
- **ea**, the approximate error of the last iteration.

If the verbose variable is set as TRUE, the function should print the values of x0, x1, x and their respective evaluations over the function f, as well as the approximate error for x, for each iteration.

3. *Required Competencies*: Muller's Method, functions, loops
   Create an R function named **MullerMethod** with the following variables as parameters:

- **f**, a variable which handles an arbitrary mathematical function;
- **x0**, a numerical variable for the first root approximate;
- **x1**, a numerical variable for the second root approximate;
- **x2**, a numerical variable for the third root approximate;
- **macheps**, a numerical variable for the machine epsilon;
- **max**, a numerical variable for the maximum number of iterations; and
- **verbose**, an optional Boolean value if the whole process is to be printed. Default value is TRUE.

The function should perform the prescribed method operating on the above variables, and must return a list which contains the values of the following variables:

- **f**

- **given_x0**
- **given_x1**
- **given_x2**
- **x3**, the current value of x3 at the current iteration
- **iterations**, the current iteration count
- **ea**, the approximate error of the last iteration.

If the verbose variable is set as TRUE, the function should print the values of x0, x1, x2, and x3 and their respective evaluations over the function f, as well as the approximate error for x3, for each iteration.

Assessment Tool

A ***programming exercise*** that implements both the False Position Method, Secant method and Muller's Method.

References

[1] Obrero, R.J. (2015). [Handout 8] Roots of Equation: Bisection Method(*CMSC 150 old handout)*
[2] Obrero, R.J. (2015). [Handout 8] Roots of Equation: False Position Method(*CMSC 150 old handout)*
[3] Obrero, R.J. (2015). [Handout 9] Roots of Equation: Secant Method (*CMSC 150 old handout)*
[4] Obrero, R.J. (2015). [Handout 10] Roots of Equation: Muller's Method (*CMSC 150 old handout)*
[5] Chapra, S. C., & Canale, R. P. (2015). ***Numerical methods for engineers*** (6th ed.). Boston ; New Delhi: McGraw-Hill Higher Education.

Exercise

After creating your function, test the accuracy of **functions** by performing the word problems below. Use $macheps = 1 \times 10^{-9}$, the maximum number of iterations to be 100000 and set the format of digits to **4 significant figures** by this R command: options(digits=4).

To describe your answers, write the last iteration's values of a, b, c and their respective values over the function f, as well as the approximate error for that iteration in a 1/2 sheet of paper, for each of the items.

1. Using False Position Method, determine the root of $f(x) = -26 + 85x - 91x^2 - 44x^3 - 8x^4 + x^5$ with initial guess $x_1 = 0$ and $x_2 = 5$.
2. Using Secant Method, determine the root of $f(x) = sin\, x + cos\,(1 + x^2) - 1$ with initial guess $x_1 = 1.0$ and $x_2 = 3.0$.
3. Using Muller's Method in the given word problem,
   When trying to find the acidity of a solution of magnesium hydroxide in hydrochloric acid, we obtain the following equation, $A(x) = x^3 + 3.5\,x^2 - 40$ where x is the hydronium ion concentration. Find the hydronium ion concentration for a saturated solution (acidity equals zero). Graph the function first, then choose your initial guesses based on the graph.