CHEMICAL ENGINEERING 2E04

Chapter 2 – Nonlinear Algebraic Equations

Module 2A: Introduction & Bracketed Methods for Univariate Equations

Dr. Jake Nease

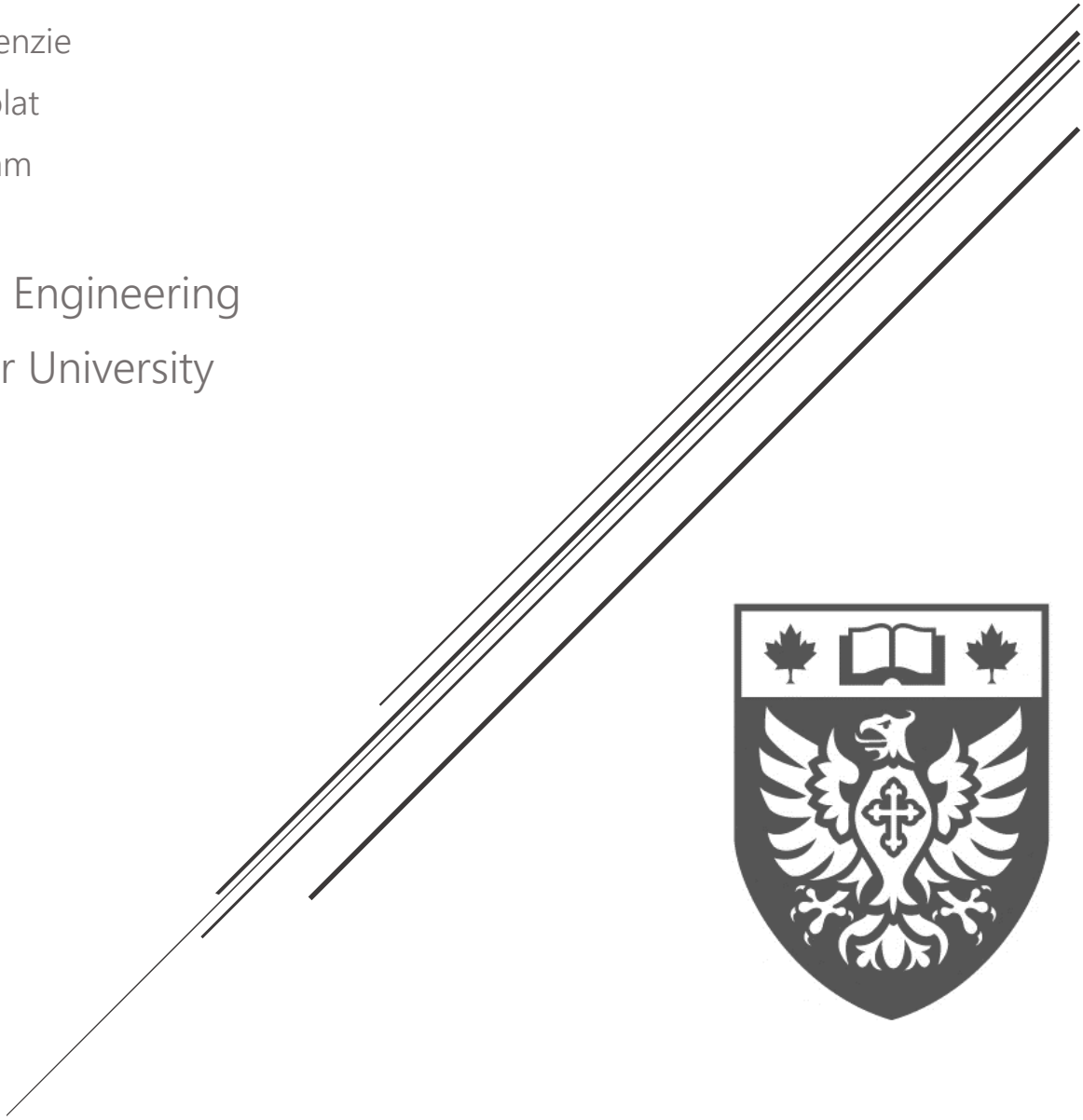
Kieran McKenzie

Steven Karolat

Cynthia Pham

Chemical Engineering

McMaster University



Updated September 11, 2019

Contents

Supplementary Material

Suggested Readings

Gilat, V. Subramaniam: *Numerical Methods for Engineers and Scientists.* Wiley. Third Edition (2014)

- Bisection Method
  - Chapter 3 → *3.3*
- Regula-Falsi
  - Chapter 3 → *3.4*
- Using `MATLAB` Built-in Functions to Solve Nonlinear Equations (Useful for Checking Your Work!)
  - Chapter 3 → *3.8*

Online Material

Math Dr. Bob > The Bisection Method *(Click to follow hyperlink)*

- References the Intermediate Value Theorem and applies the Bisection Method to a simple example

Introduction and Perspective

Now that we have discussed linear algebraic systems, we'll dive into the topic of nonlinear equations, and nonlinear systems. We previously defined a nonlinear equation as *any equation that is not linear* ☺! It is any equation *without a constant slope* (i.e. the rate of change in the dependent variable is not constant with changes in the independent variable), resulting in a *nonlinear shape*.

Recall the table from the previous Chapter, outlining some examples of nonlinear expressions to one-dimensional linear expressions:

Linear	Linear	Nonlinear	Nonlinear	Nonlinear	Nonlinear	Nonlinear
$3 + x_1$	$3x_1$	$3 + x_1^2$	x_1/x_2	$x_1^{x_2}$	$\sin(x_1)$	e^{x_1}

Learning Outcomes for this Module

- Introduce the *Bisection Method* for solving univariate nonlinear equations and develop a `MATLAB` algorithm for it.
- Derive the *Regula-Falsi Method* based on the Bisection Method for univariate nonlinear equations.
- List the benefits, drawbacks and potential failures of bracketed methods.
- Introduce the concept of *rates of convergence* for iterative methods.

Applications of Nonlinear Equations/Systems to Chemical Engineering

Nonlinear equations are the most abundant type of equation that practicing engineers are faced with. As a matter of fact, as chemical engineers we face the highest proportion of nonlinearities in all of engineering! Some examples:

- Solving the *Navier Stokes Equation* (2O04).
- Nonlinear *process dynamics* due to reactions and other nonlinearities (3P04, 3K04).
- Equations of state, reaction rates, chemical/phase *equilibria* (3M04, 3D03).

Solutions of Nonlinear Equations

Solutions to nonlinear equations are different than solutions to linear systems. In a LSoE, we solve for a solution vector, $\boldsymbol{x}$, which contains the solved parameters which satisfy all equations in the LSoE. When solving nonlinear equations, we look for the single x value that will make that equation equal to zero, otherwise known as the roots/x-intercepts.

Definition: Solution of a Nonlinear Equation

Solving a nonlinear equation, defined as $\boldsymbol{f}(\boldsymbol{x})$, is defined as finding its roots/x-intercepts by finding a numerical value of x that satisfies $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$.

The value(s) of x that satisfy this are called the solution(s) to the nonlinear equation.

Nonlinear equations can have one, many or no solutions:

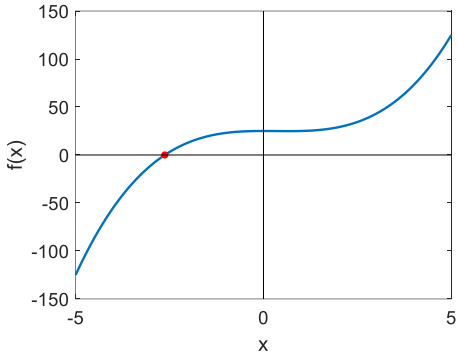


Figure 1: Plot of $f(x) = x^3 - x^2 + 25$

One Solution

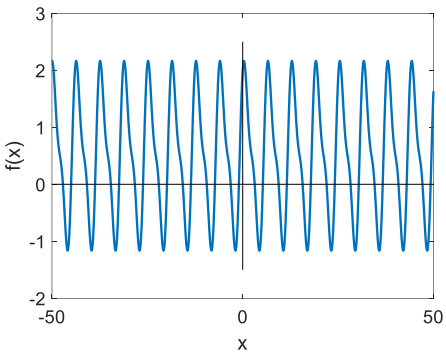


Figure 2: Plot of $f(x) = sin(x) + cos(x) + (cos(x))^2$

Many Solutions

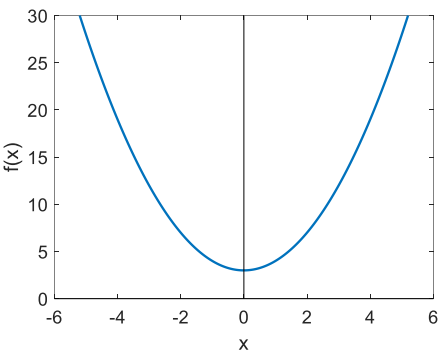


Figure 3: Plot of $f(x) = x^2 + 3$

No Solutions

Most nonlinear equations are extremely tedious (sometimes impossible) to solve by hand.

- Some nonlinear equations have analytical solutions (ex. The quadratic formula for 2nd degree polynomials, or an equation that is easy to rearrange and factor such as $f(x) = 2x^6 - 4$).
- NO general formula exists to solve polynomials of the 5^{th} degree or higher.

It is typically simpler to find a *numerical solution* to the roots of these equations, which *approximately* gives $\boldsymbol{f(x) = 0}$.

Making Hard things Easy

The underlying goal of numerical methods is to take *hard problems* and solve them *simply and algorithmically*. This allows us to use *computer algorithms* to tackle these hard problems, obtaining an answer that is *within tolerance* to the true solution!

This is the difference between the field of numerical methods and the field of calculus, which uses rigorous approaches to find *exact* solutions. In the engineering world, exact solutions are rarely required.

Algorithmic Methods to Solve Univariate Nonlinear Equations

In general, algorithms used to find values of x such that $f(x) = 0$ follow the following steps:

1) Begin with an initial guess to the solution, x.
2) Use an iterative process to update previously approximated solutions. Each successive approximation will (hopefully) become closer and closer to a true solution.
3) Repeat step 2 until our solution approximation is accurate within a desired tolerance.

We will cover two strategies to perform this process:

Bracketed Methods	Open Methods
Based on the *intermediate value theorem*. An interval is defined in which a solution is *contained* (the root is 'bracketed' or 'trapped' within the interval). During each step, the interval is *reduced* while ensuring the solution still resides within. This proceeds until a desired accuracy is met and the process is terminated.	Based on exploitation of *curvature behavior*. We begin with an *initial guess* to the solution(s), and then update them in a series of iterative steps to obtain a more accurate numerical solution. Open methods are *not confined* to a bracketed interval and can travel *outside* the bounds of initial guesses.

Advantages	Advantages
+ Will always converge to a solution *(if the function crosses the x-axis during the initial interval).*	+ Faster rates of convergence than bracketing methods. + Allowed to travel outside the bounds of initial guesses to find a solution. + Can be extended to multiple dimensions with little additional math.
Disadvantages	Disadvantages
– Slower rate of convergence (typically a linear rate). – The initial interval must contain the desired solution – an appropriate interval may be hard to guess when dealing with complex functions.	– Not guaranteed to converge. – May need to know additional information about the function (derivatives).
Methods covered in 2E	Methods covered in 2E
• Bisection • Regula-Falsi	• Secant • Newton-Raphson

Setting up Nonlinear Equations to be Solved

Following is a note on the terminology we will be using throughout this module:

Phrase	Meaning
"Solving a univariate nonlinear equation":	Finding the value of x such that $f(x) = 0$.
"Solving a system of nonlinear equations (multivariate)":	Finding the value of $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = \mathbf{0}$.

In this section, we'll gain practice setting up nonlinear equations and systems of nonlinear equations to be solved.

Let's begin with saying we have the specific heat formula you may all know and love: $Q = mc(T_2 - T_1)$ and we want to solve for our final temperature (T_2). You *could* easily rearrange the equation to isolate $T_2 = \frac{Q}{mc} + T_1$ to solve for the final temperature since this is a nice, little equation.

What if you have something a little uglier, such as the simplified Clausius-Clapeyron equation (coming up in ChE 3D03): $ln\left(\frac{p_2}{p_1}\right) = \frac{\Delta H}{R}\left(\frac{T_2 - T_1}{T_1 T_2}\right)$. If you again want to solve for the final temperature (T_2) in this situation, it's a little tougher to isolate for T_2. A format this module exploits, is setting a function equal to $\mathbf{0}$ and solving that instead. Let's try this for our two equations!

Equation	Unknown	Isolation Method	Setting to $\mathbf{0}$
$Q = mc(T_2 - T_1)$	T_2	$T_2 = \frac{Q}{mc} + T_1$	$0 = Q - mc(T_2 - T_1)$
$ln\left(\frac{p_2}{p_1}\right) = \frac{\Delta H}{R}\left(\frac{T_2 - T_1}{T_1 T_2}\right)$	T_2	$T_2 = \frac{\Delta H T_1}{R\left(\frac{\Delta H}{R} - ln\left(\frac{p_2}{p_1}\right) T_1\right)}$ Ew!	$0 = ln\left(\frac{p_2}{p_1}\right) - \frac{\Delta H}{R}\left(\frac{T_2 - T_1}{T_1 T_2}\right)$ Ah! That's better.

After setting the equation to 0, we are now left with solving a system $f(x) = 0$. You will soon find, this way should work for any complicated equation that is impossible to isolate and it is also the way the computer loves! Let's try some equations for ourselves:

Workshop: Setting Up Nonlinear Equations to be Solved

Fill in the "Rearrange to Solve" and "DOF" columns of the following table!

Equation(s)	Unknown(s)	Rearrange to Solve	DOF
$x = \quad x^3$	x	$f(x) = 0 =$	
$v^2 = \quad v_0^2 + 2a\Delta x$	v	$f(v) = 0 =$	
$x_1^2 = \quad -x_2^3$	x_1, x_2	$f(x_1, x_2) = 0 =$	
$0 = \quad \ln(sin(x_1))$ $x_2 = \quad \exp(x_1)$	x_1, x_2	$f_1(x_1, x_2) = 0 =$ $f_2(x_1, x_2) = 0 =$	

Great, now that we've warmed up, let's have an application example:

Workshop: Setting up to Solve the Catalytic Conversion of Gas Oil

Imagine we have a MATLAB program that computes where $f(x) = 0$ for a given function using a bracketing method we are about to learn. You are now given the following problem:

The catalytic conversion of gas oil (X) is possible in a moving-bed catalytic reactor. For a certain catalyst and reactor setup, the reaction follows a first-order rate law, and the conversion (X) of component A (gas oil) to B (long-chain hydrocarbons; the desired products) can be described via the following expression:

$$\frac{X}{1-X} = \frac{kC_{A0}^2 U_s}{F_{A0} k_d}\left(1 - e^{-\frac{k_d W}{U_s}}\right)$$

The following are *constants:* k is the reaction rate constant, C_{A0} is the concentration of gas oil in the inlet stream, W is the weight of catalyst contained by the reactor at any point in time, U_s is the mass flow rate of the moving catalytic bed, F_{A0} is the initial flow rate of the gas oil compound, k_d is the reaction decay constant.

1) Arrange this equation so it is ready to be solved for the catalytic conversion of gas oil (X) by a numerical method. (*Hint:* Rearrange the equation!)

2) If all other design parameters for the reactor are now held constant, how would we set up the equation to determine the bed flow rate U_s that should be used to achieve a conversion of 60%?

Bracketing Methods for Univariate Equations

When using an algorithmic method to solve a nonlinear equation, we must set up our equation in the form of $f(x) = 0$, as we just did!

Bracketing methods can be visualized well and exploit the *Intermediate Value Theorem (IVT)* to solve for a numerical solution of a function (*i.e. an approximation of the 'true solution'*).

Intermediate Value Theorem (IVT)

A continuous function $f(x)$ evaluated at two points $f(a)$ and $f(b)$ with $f(a) \leq f(b)$ *must* take on the value $f(c)$ where $f(a) \leq f(c) \leq f(b)$ for some value of c such that $a \leq c \leq b$. The opposite is true if $f(a) \geq f(b)$. Why is this important? Well…

If a continuous function has *opposite signs* at the endpoints of a given interval (for example, $f(a) < 0$ and $f(b) > 0$), the function *must cross* the x-axis *at least once* (*i.e.* $f(x) = 0$) at some point c where $a < c < b$.

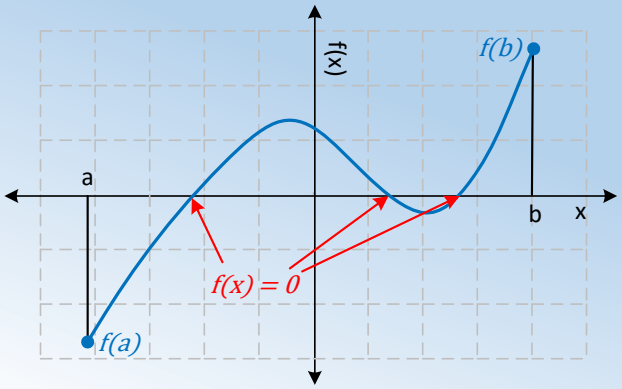


Figure 4: A plot of a function $f(x)$ showing IVT.

Below is a summarized process for solving $f(x) = 0$ using bracketing methods:

Bracketing Methods

Begin with an initial interval (a, b), within which the function is *known to have a solution and be continuous*. If $f(a) \times f(b) < 0$, then the IVT states that the solution x such that $f(x^*) = 0$ is 'bracketed' between the lower bound a and the upper bound b:

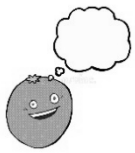
- An initial numerical solution (x_{NS}) is guessed using the lower and upper bounds.
- The interval is iteratively reduced towards the root – we're narrowing in on the solution.
- Goal: Iterate until a desired tolerance (ε) in the numerical solution is reached ($f(x_{NS}) = 0 \pm \varepsilon$), or a specified maximum number of iterations are completed.

Note:

- If the function has more than one root, only a root within the given interval will be found.
- If an interval contains more than one root, only one of those will be found (which one depends on the bounds and the method being used).

Bisection Method

The Bisection Method is an intuitive bracketed method used for solving nonlinear equations.



Imagine This:

I'm thinking of a number between 0 and 100 and you want to guess this number! You could either:

- Start from 0, and go up one-by-one until you reach the number I'm thinking of. (Linear Search)
- Bisect your interval at the middle (this would be 50 for the first guess), and check if the number I am thinking of is greater than or less than this middle value.
    - If my number is lower than 50, your guessing bounds has now become $(0, 50)$
    - If my number is higher than 50, your guessing bounds has now become $(50, 100)$
    - Your guessing interval becomes smaller and smaller! Keep bisecting/reducing your interval until you reach the number! (Binary Search)

You can probably guess that the binary search sorting method will be more computationally efficient for your algorithmic computer-mind! It turns out, binary search is analogous to Bisection Method.

All of the methods to find where $f(x) = 0$ are simply just searching algorithms – all we're doing is searching for a point on a numbered axis! If we have a function and we want to find its solution(s), how does the Bisection Method work?

General Procedure: Bisection Method

Example: Find a root of $f(x) = x^3 - x^2 + x - 5$

Steps	Our Example (FITB)	Explanation
1. Define an initial interval, within which a solution is known to be contained:	$(a, b) = (-1, 3)$ $k = 1$	By inspection, we observe that $f(a) < 0$ and $f(b) > 0$. Therefore, we can assume that the function crosses the x-axis between these points. The iteration number (k) is 1.
2. Calculate the midpoint of the interval. This will be your first approximate numerical solution:	$x_{NS}^{(k)} = x_{NS}^{(1)} = \square = 1$ $f(x_{NS}^{(1)}) = \square$	(Bisection step)

if [ ]

$set\ b = x_{NS}^{(1)}$

New interval: $(a, b) = (-1, 1)$

3. Determine if the function crosses the x-axis on the interval $(a, x_{NS}^{(k)})$ or the interval $(x_{NS}^{(k)}, b)$. This new, reduced interval will replace the old one:

elseif [ ]

$set\ a = x_{NS}^{(1)}$

New interval: $(a, b) = (1, 3)$

This step reduces the current interval by ½ each time!

else

$\boldsymbol{x^* \approx x_{NS}^{(1)}}$

end

New interval = [ ]

4. Repeat steps two and three until you have reached your desired tolerance:

Example tolerance: Continue until $|f(x_{NS}^{(k)})| < \epsilon,\ \epsilon = 10^{-5}$.

Using the initial bounds and tolerance above, this example will have a final numerical solution $x_{NS}^{(21)} = 1.8812$, where $f(x_{NS}^{(k)}) = -3.64 \times 10^{-6}$. This satisfies the desired tolerance, and thus the process is terminated.

Workshop: Understanding the IF Statements from Bisection Method

Having trouble understanding or filling in the IF-statement boxes from the procedure above? Fill in the indicated information for following plots to figure them out! :

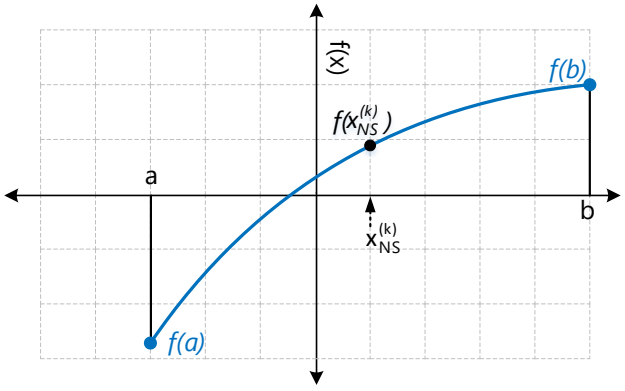


Figure 5

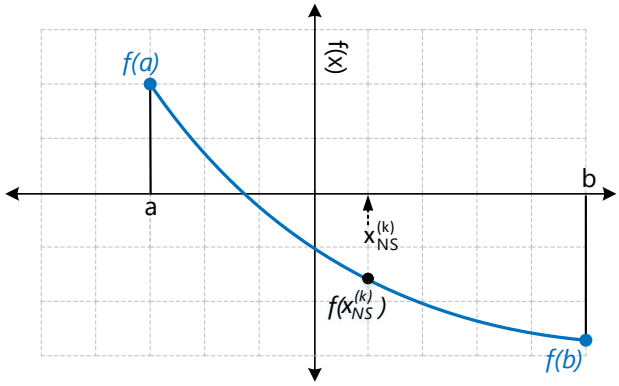


Figure 6

(True or False?)

$f(a) \times f\left(x_{NS}^{(k)}\right) =$ ☐ < 0 (T / F)

$f(b) \times f\left(x_{NS}^{(k)}\right) =$ ☐ < 0 (T / F)

Updated bound: ☐ $= x_{NS}^{(k)}$

New interval: $(a, b) =$ ☐

(True or False?)

$f(a) \times f\left(x_{NS}^{(k)}\right) =$ ☐ < 0 (T / F)

$f(b) \times f\left(x_{NS}^{(k)}\right) =$ ☐ < 0 (T / F)

Updated bound: ☐ $= x_{NS}^{(k)}$

New interval: $(a, b) =$ ☐

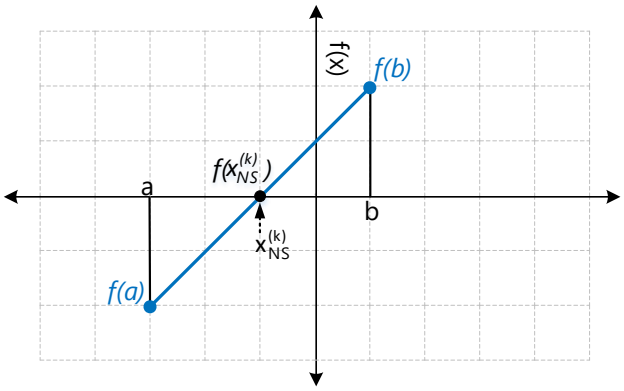


Figure 7

(True or False?)

$f(a) \times f\left(x_{NS}^{(k+1)}\right) =$ ☐ < 0 (T / F)

$f(b) \times f\left(x_{NS}^{(k+1)}\right) =$ ☐ < 0 (T / F)

$\therefore x^* \approx x_{NS}^{(k)}$

Bisection Method

The Bisection Method begins with an initial interval (a, b) of the given function $f(x)$ (within which a root is contained) and an iteration counter, $k = 1$:

1. Calculate the *midpoint* as the approximated numerical solution, $x_{NS}^{(k)}$.

$$x_{NS}^{(k)} = \frac{(a+b)}{2}$$

2. Evaluate $f\left(x_{NS}^{(k)}\right)$ and update the interval.

   - Check whether the function crosses the x-axis over the interval $\left[a, x_{NS}^{(k)}\right]$ OR $\left[x_{NS}^{(k)}, b\right]$:
     IF $f(a) \times f\left(x_{NS}^{(k)}\right) < 0 \Rightarrow b \rightarrow x_{NS}^{(k)}$
     ELSE-IF $f(b) \times f\left(x_{NS}^{(k)}\right) < 0 \Rightarrow a \rightarrow x_{NS}^{(k)}$
     ELSE terminate $x^{\star} \approx x_{NS}^{(k)}$
   - Once decided, the interval (a, b) is redefined using the new (halved) interval.

3. IF a *stopping condition* is met, terminate $x^{\star} \approx x_{NS}^{(k)}$. OTHERWISE, update $k = k + 1$, and return to (1).

Example Tolerance/Stopping Conditions:

- $|a - b| < \epsilon$ (width of the search space is sufficiently close to zero)
- $\left|f\left(x_{NS}^{(k)}\right)\right| < \epsilon$ (function evaluation is sufficiently close to zero)
- $k \geq \mathcal{M}$ (maximum iterations ($\mathcal{M}$) reached)

Note: The following conditions must be met for the Bisection Method to be useable:

- The solution to $f(x) = 0$ is known to exist within the initial interval, (a, b).
- The given function $f(x)$ is continuous.

The steps for the Bisection Method are repetitive, and therefore can be programmed simply – let's try!:

Workshop: Making a Pseudo Code for Bisection Method

You have been hired to create the pseudo code for Bisection Method used in the previous catalytic conversion problem! There is space provided for you below, where `iterations` is the predefined maximum number of iterations. The catalytic conversion function in $f(x) = 0$ form is already defined as the function `catalyticReactor(X)`.

```
for i = 1:iterations
    
    
    
    
    
    if abs(a-b) <= tolerance
        break % we have found the solution!
    end
end
```

Regula-Falsi Method

The *Regula-Falsi method* is another bracketed method - like the Bisection Method, an interval, (a, b), is defined where within which a solution is known to exist, and over which the function is continuous.

Workshop: Deriving the Regula-Falsi Step

Beginning with an interval (a, b) which is known to contain a root of $f(x)$, the Regula-Falsi Method uses the x-*intercept of the secant line* between $f(a)$ and $f(b)$ to determine $x_{NS}^{(k)}$, as shown in Figure 8.

Using Figure 9, derive an algebraic formula to solve for $x_{NS}^{(k)}$. Two slopes, m_1 and m_2, such that $m_1 = m_2$, have been drawn on the plot to help you – use these to derive the formula.

The formula you arrive will be the 'Regula-Falsi step' that we'll use in our codes for Regula-Falsi.

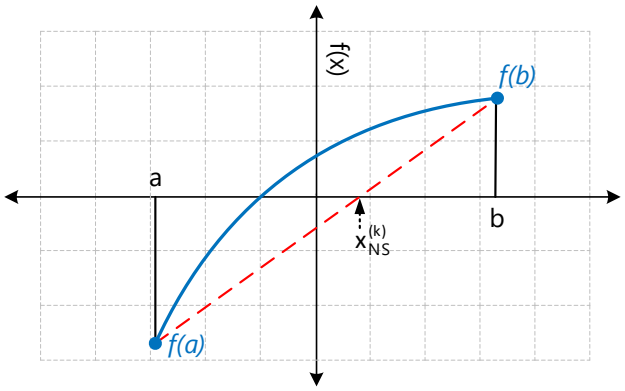


Figure 8: Secant drawn between $f(a)$ and $f(b)$. The root of this secant will be the new $x_{NS}^{(k)}$.

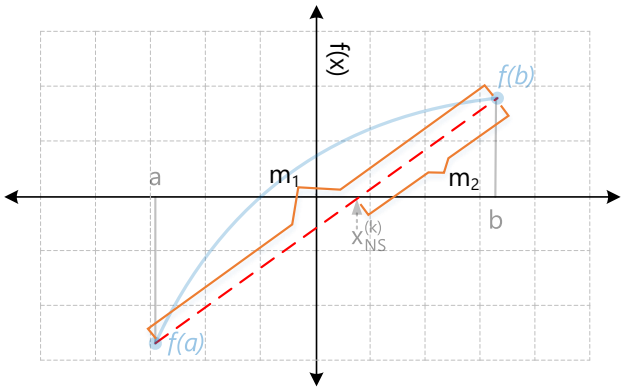


Figure 9: Two slopes drawn, m_1 and m_2, for the purposes of deriving the Regula-Falsi step.

Regula-Falsi Method

The Regula-Falsi method also begins with an initial interval (a, b), within which a solution to $f(x) = 0$ is known to exist, and the given function $f(x)$ is continuous. We start with an iteration counter $k = 1$.

1. Calculate the x-intercept of the *secant* between $f(a)$ and $f(b)$ as the approximated algorithmic solution, $x_{NS}^{(k)}$:

$$x_{NS}^{(k)} = a - f(a)\left[\frac{b-a}{f(b)-f(a)}\right] \quad \underline{\text{OR}} \quad x_{NS}^{(k)} = b - f(b)\left[\frac{b-a}{f(b)-f(a)}\right]$$

2. Evaluate $f\left(x_{NS}^{(k)}\right)$ and update the new redefined interval.
   - Check whether the function crosses the x-axis over the interval $\left[a, x_{NS}^{(k)}\right]$ OR $\left[x_{NS}^{(k)}, b\right]$:
     IF $f(a) \times f\left(x_{NS}^{(k)}\right) < 0 \Rightarrow b \to x_{NS}^{(k)}$
     ELSE-IF $f(b) \times f\left(x_{NS}^{(k)}\right) < 0 \Rightarrow a \to x_{NS}^{(k)}$
     ELSE terminate $x^{\star} \approx x_{NS}^{(k)}$

3. IF a *stopping condition* is met, terminate $x^{\star} = x_{NS}^{(k)}$. OTHERWISE, update $k = k + 1$ and return to (1).

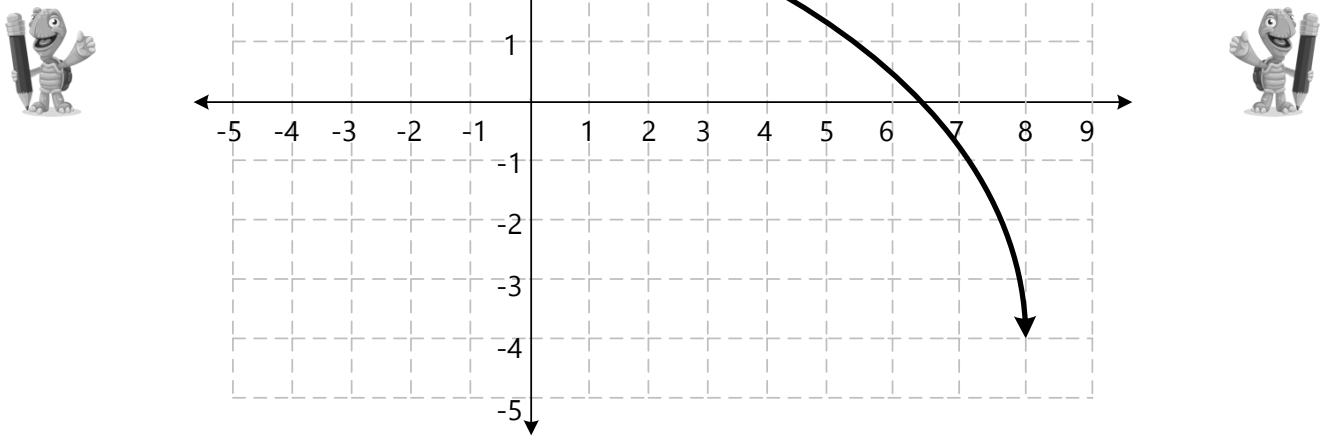
Example Tolerance/Stopping Conditions (same as before)

- $|a - b| < \epsilon$ (the width of the search space is sufficiently close to zero)
- $\left|f\left(x_{NS}^{(k)}\right)\right| < \epsilon$ (the function evaluation is sufficiently close to zero)
- $k \geq \mathcal{M}$ (maximum iterations have been reached)

Let's see this in action:

Workshop: Visualizing Regula-Falsi

Graphically show the convergence of the Regula Falsi Method on the following function starting with the given interval. Remember to indicate each: b_k, a_k, $f(a_k)$, $f(b_k)$, and $x_{NS}^{(k)}$:



Initial Interval: $(a, b) = (-3, 8)$

Advantages and Disadvantages of Bracketing Methods

Benefits	Drawbacks
• Guaranteed convergence • Requires very little computational demand per iteration.	• Typically require more iterations to achieve convergence and therefore convergence is achieved more slowly/less efficiently in *most* cases

Failures

- A function must cross the x-axis to be bounded and solved with bracketing methods
    - Bracketing methods will fail if the desired solution is tangent to the x-axis
        - Algorithms which rely on the IVT will not work in this case
- The root is required to be contained within the initially chosen interval
    - Choosing an appropriate proper interval can grow more difficult as functions grow more complex

Rates of Convergence

Rate of convergence typically refers to a method's *ability to converge* to a solution in *fewer iterations*. A method that can converge to a solution in fewer iterations than another will have a *higher rate* of convergence.

Rate of Convergence

The rate of convergence of a numerical method describes how quickly that method will theoretically converge to a numerical solution. The *rate of improvement in accuracy* ($\mathcal{R}_A$) of a candidate solution relative to the true solution (which we don't know yet) as a function of its *rate of convergence* ($\mathcal{R}_C$) is:

$$\mathcal{R}_A = (\mathcal{R}_C)^i$$

Where i is the iteration number. For example:

- *Linear* rate of convergence: $\mathcal{R}_A = (1)^i$
    - Each step can converge toward the solution at the same pace/accuracy.
- *Quadratic* rate of convergence: $\mathcal{R}_A = (2)^i$
    - The accuracy of the numerical solution can *double* with each iteration!

Theoretically, a method which follows a quadratic rate of convergence should converge in half as many iterations than a method with a linear rate of convergence would require.

Note: We say "can" or "could" because there is always the risk of a method diverging, cycling, etc.

Both the Bisection and the Regula-Falsi method follow a *linear rate of convergence* ($\mathcal{R}_C = 1$):

- $\mathcal{R}_A$ does not change with respect to the iteration number.

Conclusion and Summary

In this module we have covered:

- Two bracketed methods – Bisection and Regula-Falsi for numerically solving univariate nonlinear systems by numerical methods.
- The advantages and disadvantages of the bracketed methods.
- Rates of convergence for methods that involve iterations and convergence for the bisection and Regula-Falsi

Next up: Open Methods for solving Nonlinear Systems