CHEMICAL ENGINEERING 2E04

Chapter 2 – Nonlinear Algebraic Equations

Module 2B: Open Methods for Nonlinear Equations

Dr. Jake Nease

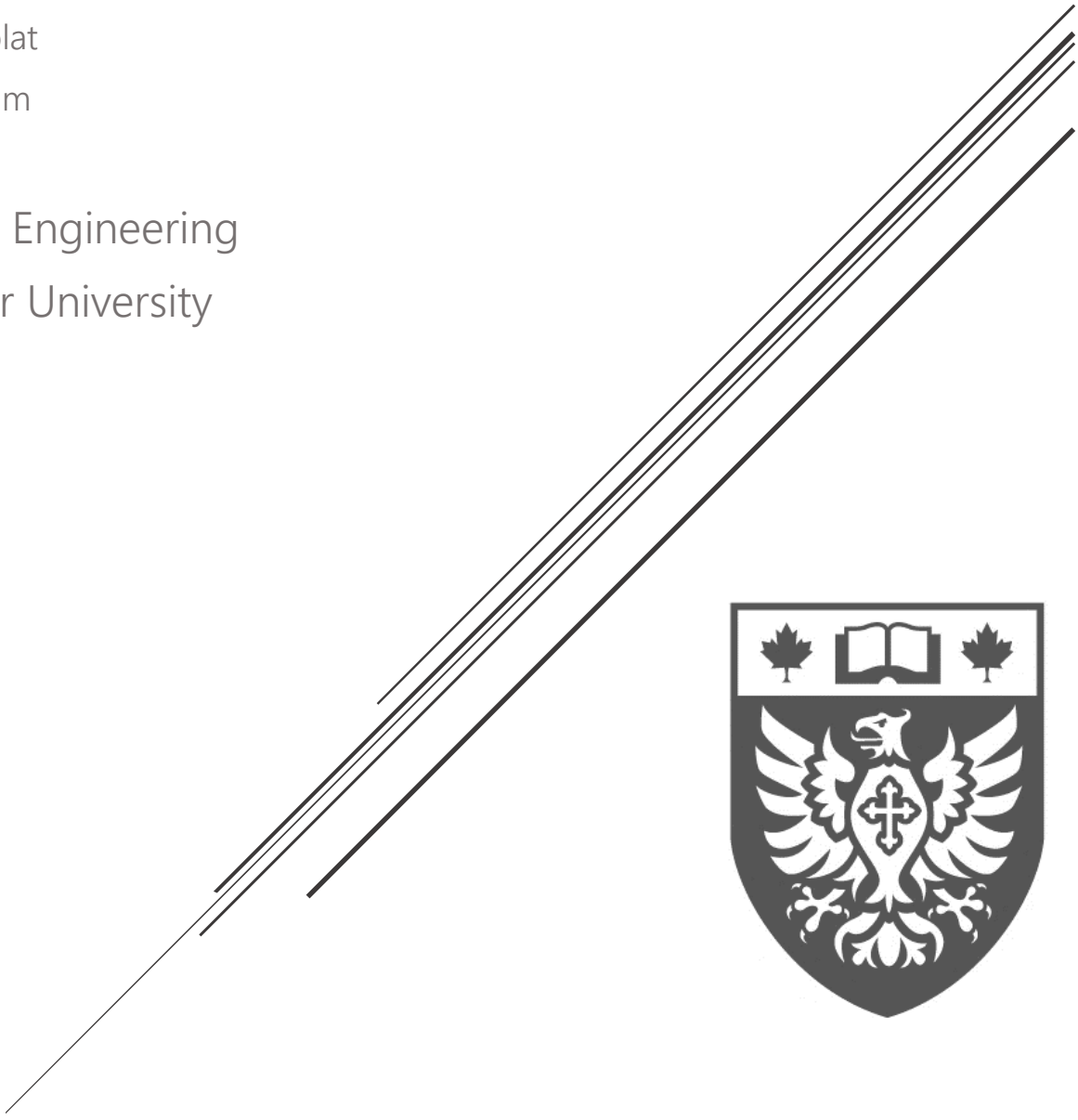
Kieran McKenzie

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)

- Secant Method
  - Chapter 3 → *3.6*
- Newton-Raphson Method
  - Chapter 3 → *3.5*
- Multivariate Newton-Raphson Method
  - Chapter 3 → *3.10, 3.10.1*

Online Material

Ilectureonline > What is the Taylor Series? *(Click to follow hyperlink)*

- Derives the Taylor Series expansion formula from the general power series formula

Oscar Veliz > Secant Method *(Click to follow hyperlink)*

- Visually demonstrates Secant Method, discusses rate of convergence, compares Secant to Bisection and NR

Ilectureonline > Newton's Method, Lecture 1 *(Click to follow hyperlink)*

- Derives the Newton's Method algorithm and theory

Overview

We learned that bracketing methods typically require more iterations to find a solution, and require that an interval containing a solution is bounded with our initial guesses. Recall, in the previous module we defined *open methods* as iterative methods that are not confined within an initial bound. Let's look at this alternative to solving nonlinear systems of equations using two open methods: Secant and Newton-Raphson method!

Learning Outcomes for this Module

- Introduce *Secant Method* and apply it to a Chemical Engineering example.
- Discuss the theory behind *Newton-Raphson Method* and derive it from a *Taylor Series Expansion*.
- Analyze the *advantages* and *disadvantages* of both open methods.
- Compare *rates of convergence* of the open methods.
- Introduce *Multivariate Newton-Raphson Method* to solve multivariate systems of nonlinear equations.

Secant Method for Univariate Equations

This method has similarities to the Regula-Falsi bracketing method. Making use of the secant between two points is a clever way to isolate and solve for a root provided that the function changes *monotonically* between those points. Two initial points must be given to create the first secant; however, the root is *not required* to be between those two initial points.

Question: How close do the initial points have to be to the solution that we're solving for?

Answer: There is *no general rule* for how close "close enough" is! The range of acceptable distances and locations *vary* with the nature of the function. For example, a *monotonic function* will converge with the Secant Method regardless of where the two initial points are taken, since the secant will always lead toward the root. However, some other functions may diverge away from the root if the points are too far away from it. We'll touch on more scenarios where the Secant Method fails later.

Fun Fact: A *monotonic* function refers to a function that is either always increasing or decreasing. The origin of this word comes from the Greek prefix 'mono-', which means 'single'. The suffix, 'tonic' refers to a 'tone', the word used to describe a pitch, quality, and strength of a sound. A 'monotone' is a sound with constant tone, and unchanging pitch. From this, we can see why the name for this function makes sense – a monotonic function is a function with a constant, unchanging direction, either always increasing or always decreasing!

Secant Method

The Secant Method takes two points, $x^{(k-1)}$ and $x^{(k)}$, and forms a secant between $f\left(x^{(k-1)}\right)$ and $f\left(x^{(k)}\right)$. In general, the procedure to iterate through $x^{(k+1)}$ is:

1. Select initial guesses $x^{(1)}$ and $x^{(2)}$. Set the iteration counter $k = 2$.

2. Use the intercept of the secant between $f\left(x^{(k-1)}\right)$ and $f\left(x^{(k)}\right)$ to approximate the next value $x^{(k+1)}$:

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)\left(x^{(k-1)} - x^{(k)}\right)}{f\left(x^{(k-1)}\right) - f\left(x^{(k)}\right)}$$

3. Update $k = k + 1$. IF a *stopping condition* is met, terminate $x^{\star} = x_{NS}^{(k)}$. ELSE, return to (2) and continue updating the numerical solution using the most recently calculated points and slopes.

- The numerical solutions will (hopefully) grow closer/converge toward the true solution, and the secant will (hopefully) shorten as this happens.
- The process terminates once a desired tolerance or stopping condition is met – when termination occurs, the final (n^{th}) numerical solution will be used as the final root approximation: $x^{\star} \approx x^{(n)}$.

Example Tolerance/Stopping Condition:

- $\left|f\left(x^{(n)}\right)\right| < \epsilon$ (the function evaluation is sufficiently close to zero)
- $n > \mathcal{M}$ (the maximum number of iterations has been reached)

Great! Let's crank out an example to solidify the concept on the next page.

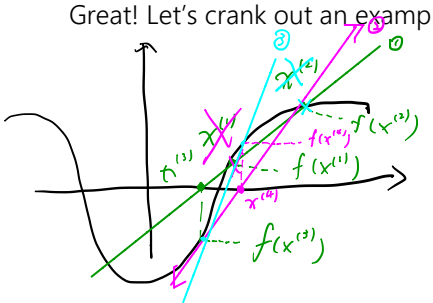
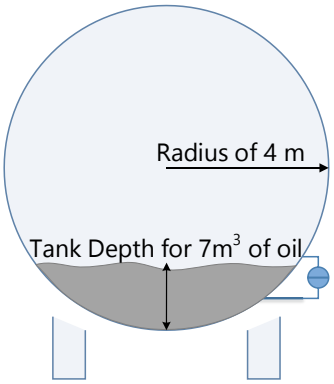


Figure 1: Cranking out another

Workshop: Applying Secant Method to an Oil Tank

(Adapted from MathforCollege.com/Physical Problem for Nonlinear Equations: Chemical Engineering)

Oil is stored in a spherical tank with a radius of $4\ m$. What is the depth of the oil in the tank when it holds a volume of $7\ m^3$?



The equation for oil volume, $V = \frac{\pi h^2(3r-h)}{3}$, ($h$ represents the height of the oil in the tank, and r represents the radius of the tank) can be solved to find a nonlinear equation for height:

$$h^3 - (3r)h^2 + \frac{3V}{\pi} = 0\ .$$

Solve for at least the fourth numerical solution of h using the initial points as $h^{(1)} = 6\ m$, $h^{(2)} = 4\ m$.

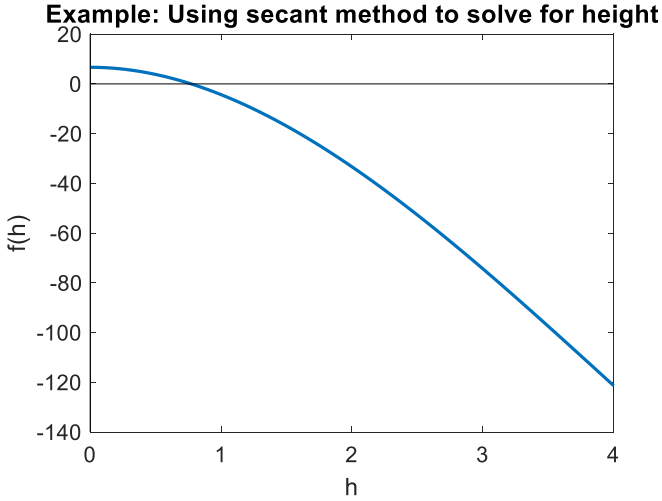
$f(h) = h^3 - 3(4)h^2 + \frac{21}{\pi} = 0$

$f(h^{(1)}) = -209$

$f(h^{(2)}) = -121$

$f(h^{(3)}) = h^{(2)} - \frac{f(h^{(2)})(h^{(1)} - h^{(2)})}{f(h^{(1)}) - f(h^{(2)})} = 1.25 \ldots$

True solution: $h_{root} = 0.7716$ *(Other roots found at* $h = (-0.7248)$ *and* $h = (11.9532)$ *, however, height cannot be negative, and our sphere only has a diameter of 8 m).*



Rate of Convergence: Secant Method

- Follows a *super-linear rate* of convergence ($\mathcal{R}_A = 1.618$ ... *the Golden Ratio!*)
  - Faster than linear convergence, slower than quadratic.
  - Converges more rapidly when near a root.
- Typically requires more iterations than NR. However, *each iteration takes less time*.
  - Only one function evaluation needs to be performed ($f(x^{(k)})$), and the calculation of the first derivative is not necessary.
  - *Two iterations* of Secant Method are as computationally expensive as one iteration of NR!

Advantages: Secant Method

- The root is not solely required to be within the brackets of the initial interval (like all open methods ☺)
- No calculations of derivatives required (unlike the Newton-Raphson method, which we will see next).
- Uses a secant line in its process to find roots of a function.
  - As successive iterations are performed, a converging secant method will begin to narrow in on a solution, thus shortening the length of the secant line.
  - In this scenario, a close approximation of the tangent line will eventually be formed, resulting in an approximation of the function's derivative (this is used in the later unit on Numerical Differentiation!).
- This method is simple in nature and easy to implement.
- Can be used as a casual topic to tell friends at small get-togethers, and as a conversation starter at networking events.

Disadvantages: Secant Method

The secant method will fail under certain circumstances. Recall the equation used to find a more refined numerical solution:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})(x^{(k-1)} - x^{(k)})}{f(x^{(k-1)}) - f(x^{(k)})}$$

SCENARIO 1: $f(x^{(k-1)}) - f(x^{(k)}) = 0$ occurs when the slope of the secant between the two points on the function is equal to zero, and therefore never intersects the x-axis. In this case, the Secant Method will encounter *numerical failure*. This error can be common with functions that are symmetric about the y-axis.

SCENARIO 2: When each iteration draws the numerical solution further away from the actual root. For example, imagine if the points $x^{(1)} = 3$ and $x^{(2)} = 5$ were chosen as initial points for the function in Figure 2. This function has a limit at $y = 1$; the secant calculated with these initial points would have a slope very close to zero, and would intersect the x-axis at a very large negative value. The next iteration would produce a secant with slope even closer to zero, and so on, resulting in *divergence* of the Secant Method. The same situation will occur in any case where the function is 'very flat.'

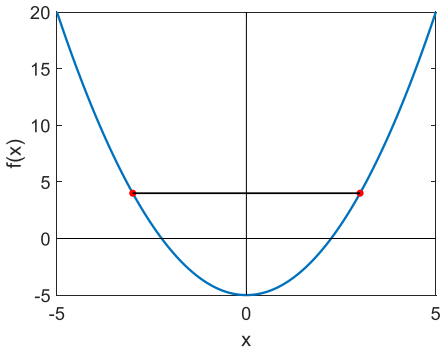


Figure 1: Plot of $f(x) = x^2 - 5$, showing scenario 1.

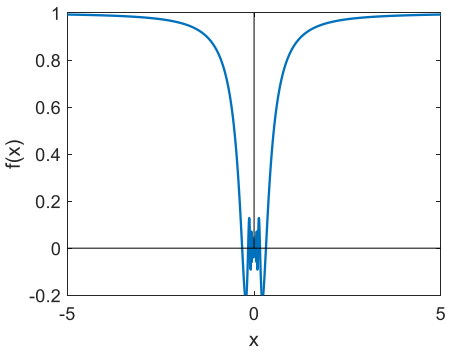


Figure 2: Plot of $f(x) = x \times sin\left(\frac{1}{x}\right)$, showing scenario 2.

FUN IDEA: "Spice up the Secant" using Hybrid Methods

There are open methods, and there are bracketing methods. But guess what?! There are also hybrid methods! Hybrid methods combine open and bracketing methods together to create a more efficient root finding algorithm. Consider an algorithm that takes an initial interval, uses the Bisection Method to gain an x-value that is close to the root of a function, then, after a certain trigger condition is met, switches to the Secant Method to finish the process. This method has potential to achieve a faster rate of convergence than if the Bisection Method or Secant Method were used alone!

Taylor Series Expansion

Yes, "everyone's favorite topic." However, this sarcasm might be because they are misunderstood. Hopefully after this section, you will realize the *power* and *simplicity* of the Taylor Series.

A function $f(x)$ may be approximated as an n^{th} order polynomial in a *neighborhood* (close immediate area) of a known point a by using n derivatives for that function evaluated at point a. The polynomial form is convenient, since they are relatively easy to understand and work with for integration, derivation and so-on.

Taylor Series

An n^{th} order Taylor Series for a function $f(x)$ centered on (or referenced to) a point $x = a$ is:

$$f(x) \approx T_n(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \frac{f'''(a)(x-a)^3}{3!} + \cdots + \frac{f^{(n)}(a)(x-a)^n}{n!}$$

nth order of Taylor series

Theoretically, it is possible to *perfectly* represent any function $f(x)$ using a Taylor Series with infinite terms T_∞:

$$f(x) \equiv T_\infty(x) = f(a) + \sum_{i=1}^{\infty} \frac{f^{(i)}(a)(x-a)^i}{i!}$$

Note: There is a difference between $f(x) \approx T_n(x)$ and $f(x) \equiv T_\infty(x)$. The more terms used in the expansion, the more accurate the series will be! Using the resulting equation, any values of $f(x)$ can be *approximated*.

Each successive term in a Taylor Series has diminishing returns on the result of $T_n(x)$. This means that we often do not need to obtain a full Taylor Series to get a close approximation. In fact, we usually just cut it off at two or three terms.

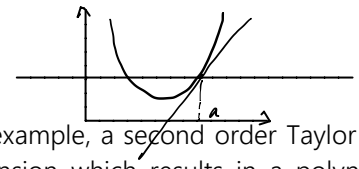
Requirements for Taylor Series:

- All $f^{(i)}$ must exist, where i is the derivative number (i.e. $f'''(x) \equiv f^{(3)}(x)$)
  - This is the standard notation to denote the derivative number of f. Do not confuse this with our $x^{(k)}$ notation for iteration number!
- $f(x)$ must be continuous
- $f(x)$ must be smooth (non-smooth functions are not differentiable!)

Note: Terms like $f''(a)$ just refer to the second derivative *evaluated* at the point a - they're just numbers!

V corner
X 不可以.

Grade
Study time



For example, a second order Taylor Polynomial $T_2(x)$ (i.e. an expansion which results in a polynomial of degree 2) that represents e^x is:

$$f(x) = e^x \approx T_2(x) = \quad f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!}$$

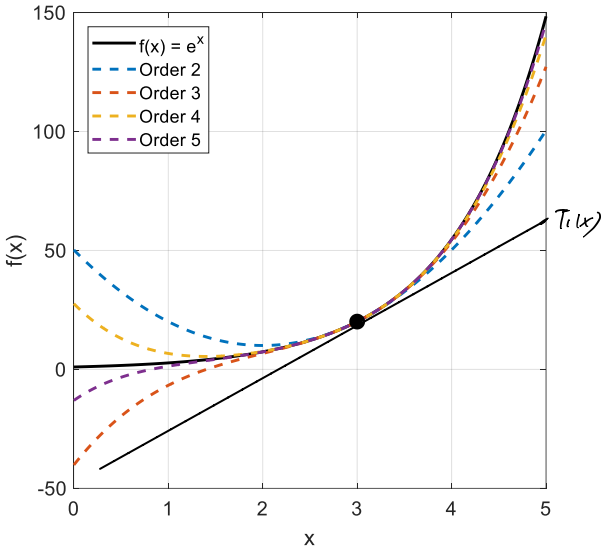
$$T_2(x) = \quad e^a + e^a(x-a) + \frac{e^a(x-a)^2}{2!}$$

If $a = 3$ is used as our centering point for the Taylor Series:

$$T_2(x) = e^3 + e^3(x-3) + \frac{e^3(x-3)^2}{2!}$$

After expansion and collecting like terms (which you don't need to do when using a computer!), this becomes:

$$T_2(x) = e^3\left(\frac{x^2}{2} - 2x + 7\right)$$



Plotting the true function against the second order Taylor Series, the improved accuracy achieved from higher order Taylor Series expansions is illustrated. The higher-order Taylor Series expansions do not add a lot of accuracy very close to the centering point $a = 1$, but they do increase the accuracy range of the approximation.

Workshop: Representing an n^{th} order polynomial with an n^{th} order Taylor Series

Represent $f(x) = 3x^2 + 8x - 3$ centered at $a = 2$ with a second-order Taylor Series expansion.

$f(a) = 3(2)^2 + 8(2) - 3 = 25$

$f'(x) = 6x + 8$

$f'(a) = 6(2) + 8 = 20$

$f''(x) = 6$

$f''(a) = 6$

$f(x) \approx T_2(x)$

$= f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2}$

$= 25 + 20(x-2) + \frac{6}{2}(x-2)^2$

$= 25 + 20x - 40 + 3(x^2 - 4x + 4)$

Obtaining Appropriate Representations of your Function using Taylor Series

- The only way for the Taylor Series to *perfectly* match the true function is if *every* non-zero derivative possible is used in the Taylor expansion.
    - For example, a full Taylor Series for a quadratic equation will have three non-zero function evaluations ($f(a),\ f'(a),\ f''(a)$) since the 3rd and all further derivatives of a quadratic is 0.

- In other words, *an* n^{th} *order Taylor Polynomial* will represent *any given* n^{th} *order polynomial* function perfectly!

- However, it is usually impractical to use every non-zero derivative possible.
  - Infinite derivatives exist for functions such as logarithmic and trigonometric function!
- Thankfully, we can obtain an appropriate representation of the function for our purposes without a full Taylor Series. However, excluding (truncating) valid terms reduces overall accuracy.
- Imagine if your plot looks like this:

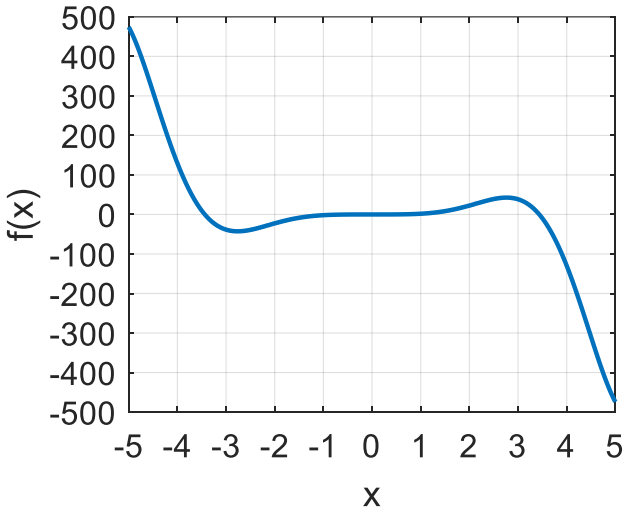


Figure 3: Unknown function!

You don't necessarily know the underlying function/relation that caused it *(is it a cubic? 4th degree? Some weird combination of* $\sin(x)$ *and/or* e^x*, which have infinite derivatives?)*. This brings up the question of how many terms are needed to get a 'good' approximation? Again, this comes back to *you as an engineer* to decide what level of accuracy is appropriate for the given situation.

In terms of *how bad* our approximation is, we can always attempt to estimate (or at least bound) the *remainder* of a Taylor Series.

Choosing the Correct Value for a:

You may ask, does my choice of a matter? As shown above, the Taylor Series expansion more accurately represents the true function *near the chosen point*, a. After a certain range, the Taylor Series veers off track, and requires a higher order Taylor Polynomial to be calculated (i.e. more terms used in the Taylor expansion) to continue to accurately represent the true function. It is for this reason that you often hear the term "Taylor Series centered at a."

Computational efficiency is important in practical applications. If function values are desired around the neighbourhood of a specified point, it will be more computationally efficient to choose a as that point so that fewer terms in the Taylor Series expansion are required for accurate approximations.

Taylor Series Truncation & Remainder

A full Taylor Series is not always ideal to calculate. As a result, we truncate the series by only taking a n^{th} order approximation, leaving R^{n+1} as the remainder/error term.

$$f(x) = f(a) + \sum_{i=1}^{n} \frac{f^{(i)}(a)(x-a)^i}{i!} + \underbrace{R^{n+1}}_{Remainder}$$

Where the *remainder* R^{n+1} is the infinite sum of all terms $n+1$ and above:

$$R^{n+1} = \sum_{i=n+1}^{\infty} \frac{f^{(i)}(a)(x-a)^i}{i!}$$

If we ignore the remainder, $f(x)$ is *approximately:*

$$f(x) \approx T_n(x) = f(a) + \sum_{i=1}^{k} \frac{f^{(i)}(a)(x-a)^i}{i!}$$

And we can combine the above two equations to get:

$$f(x) = T_n(x) + R^{n+1}$$

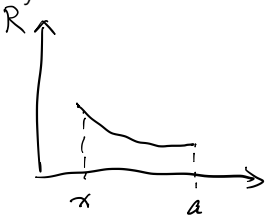
Looking at the remainder term will help us *quantify potential errors* and *anticipate the rate of convergence* of solution methods based on a Taylor Series expansion. We can represent the remainder term in a new way (the proof for this is complicated so we'll skip it):

$$R^{n+1} = \frac{f^{(n+1)}(\xi)(x-a)^{n+1}}{(n+1)!}$$

Where ξ is a number between x and a that represents the true error of our approximation.

Although ξ is unknown, we can use it to form an *upper bound on the error* of our n^{th} order Taylor Series. As a matter of fact, the error at point ξ must be between the remainder of a and x:

$$\underbrace{|R^{n+1}|}_{\substack{Absolute\ value \\ of\ error\ term}} = \underbrace{\left|\frac{f^{(n+1)}(\xi)(x-a)^{n+1}}{(n+1)!}\right|}_{\substack{Error\ term\ exactly \\ using\ \xi}} < \underbrace{\left\lceil\left|\frac{f^{(n+1)}(x,a)(x-a)^{n+1}}{(n+1)!}\right|\right\rceil}_{\substack{Error\ term\ found\ by\ computing \\ it\ using\ a\ \mathbf{or}\ x}}$$



So, although we don't know the exact value of the error, we know it is definitely *less* than the higher value of terms contained in the ceiling $\lceil\cdot\rceil$ operator. If this value is relatively small, we know that our Taylor Series serves as a reasonable approximation! If the remainder is large, we may require a higher-order Taylor Series.

Note: *The ceiling operator* $\lceil\cdot\rceil$ *for a set of numbers returns the largest number (ceiling) of that set. For example, the largest number in the set* $v = \{1 \quad 5 \quad 12 \quad 3 \quad -5\}$ *is* $\lceil v \rceil = 12$.

Workshop: Taylor Series Remainder

1. What happens to R^{n+1} as $n \to \infty$?

As $n \to \infty$, $T_N(x) \to f(x)$.

$\Rightarrow R^{(n+1)} \to 0$

$\frac{1}{(n+1)!} \to \frac{1}{\infty} = 0$ as $n \to \infty$

Which term is $f^{(n+1)}(S)(x-a)^{n+1}$

2. For the same a and x, if I half the distance between them, what happens to R^{n+1}?

As distance cut in half, $\Rightarrow (x-a) = \frac{1}{2}$ as much.

$(x-a)^{n+1}$

↑ distance away from centre.

Assume T_2 & $(x-a) = 2$ $R^{(3)} = \square \cdot (2)^3$

cut half & $(x-a) = 1$ $R^{(3)} = \square (1)^3$

3. What is R^{n+1} for the Taylor series expansion of an n^{th} order polynomial?

Newton-Raphson Method

Newton-Raphson (NR), AKA Newton's Method – you were likely first introduced to this method in first year calculus.

Derivation of Newton-Raphson Method from Taylor Series

The mechanics behind this method comes from the Taylor Series expansion, hence the past section's Taylor Series review. For NR, we only need the first-order expansion for the function at a desired point a:

$$f(x) = f(a) + f'(a)(x - a) + R^2$$

As x approaches a (*i.e.* $(x - a) \to 0$), the value of $R \to 0$ at a *quadratic rate*. This is because our Taylor Series becomes more representative of our given function when the distance between our starting point, a and x are close together, and EXACT once $a = x$. See for yourself in the graph below which uses various order Taylor Series to model e^x.

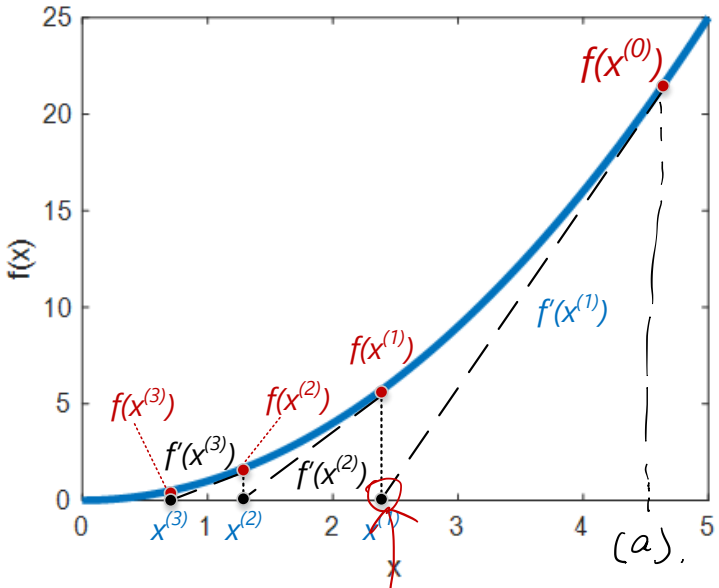


Figure 4: A representation of how Newton-Rhapson method works to come nearer to the solution.

Therefore, assuming R^2 is small (which may or may not be true) *we desire* $f(x) = 0$, which means we can approximate when our function equals zero using the Taylor Series with the LHS = 0. In other words:

$$0 = f(a) + f'(a)(x - a)$$

We can rearrange the above equation to be:

$$x = a - \frac{f(a)}{f'(a)}$$

This will give us the x-value where the *first-order Taylor Series approximation* of $f(x)$ hits the x-axis. This calculation will then be repeated until we achieve an x-value within our pre-determined tolerance.

Newton-Raphson Method

This method uses a *tangent line* at a single point. Where that tangent crosses the x-axis will be the location of the approximated solution to $f(x) = 0$, and serves as the approximated numerical solution.

1. Select an initial guess: $x^{(0)}$. Set the iteration counter $k = 0$.
2. Compute the tangent line at that point to approximate the next value $x^{(k+1)}$:

$$\underbrace{x^{(k+1)}}_{New\ Iteration} = x^{(k)} - \underbrace{\frac{f(x^{(k)})}{f'(x^{(k)})}}_{Newton\ Step,\ \Delta x}$$

3. Update $k = k + 1$. IF a *stopping condition* is met, terminate $x^{\star} = x_{NS}^{(k)}$. ELSE, return to (2) and continue updating the numerical solution using the most recently calculated point and tangent lines.

For NR to be performed, there are a few requirements (same requirements as Taylor Series):

- The function must be *continuous*
- The function must be *differentiable* – we need to be able to find $f'(x)$

Tolerance/Stopping Conditions:

- $|f(x^{(k)})| < \epsilon$ (the function evaluation is sufficiently close to zero)
- $k > \mathcal{M}$ (the maximum number of iterations has been reached)

Workshop: Newton-Raphson in Action

Let's write out a pseudocode for Newton-Raphson that takes in the function, the function's derivative, an initial guess, the tolerance and maximum iterations, where `xVal` is the approximated solution. We will use it for the spherical tank example, where $f(h) = h^3 - (3r)h^2 + \frac{3V}{\pi} = 0$ and draw out the first two iterations by hand.

```
function [xVal] = newtonRaphson(myFunc, myFuncDeriv, x0, tol, iter)
x = zeros(iter, 1)
xVal = x0;
x(1) = x0;
for i = 1:iter
x(i) = x0
x(i+1) = x(i) - myFunc(x(i)) / myFuncDeriv(x(i));
% stop crit?
if abs(myFunc(x(i+1))) < tol
xVal = x(i+1)
break
end.
x(i+2:end) = [ ]
x0 = x(i)
end
```

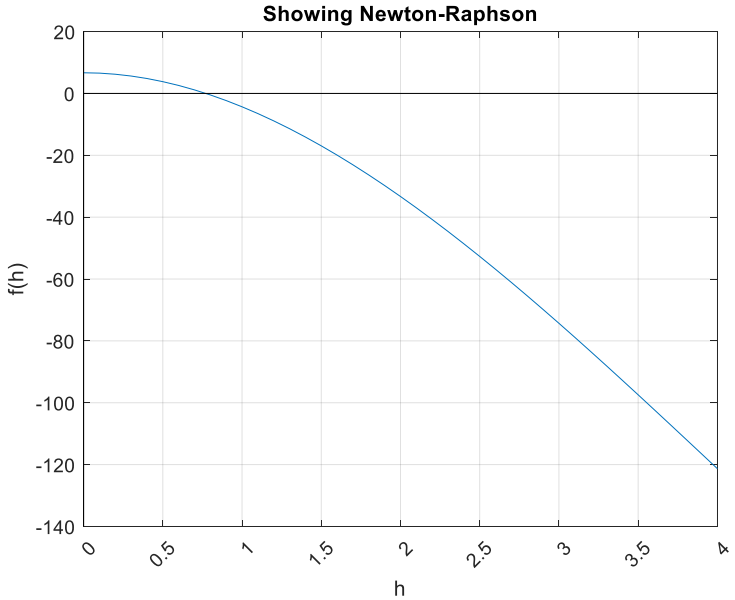
x = [2; 4; 8; 0; 0; 0]

Delete 0s:

>> x(4:7) = [ ];

x = [2; 4; 8]

Given: Volume= 7 m^3, Tank radius = 4 m, Initial guess, $h^{(0)} = 5\ m$



Rate of Convergence: Newton-Raphson

- Follows a *quadratic rate* of convergence ($\mathcal{R}_A = 2$)
  - Typically converges in fewer iterations than Secant Method, depending on the problem
  - However, recall, each iteration requires two function evaluations
- As a result, although converging in fewer iterations, NR takes longer to compute each iteration. Typically, it performs slower on a computer than Secant Method per iteration

Advantages: Newton-Raphson

- A very fast and effective method to solve nonlinear equations
  - Often, if the initial guess is good (near the solution), the system will converge quickly
  - In general, NR follows a *quadratic rate* of convergence
- Can readily be expanded to multiple dimensions
- Can exploit *numerical derivatives* to eliminate the need for analytical derivations

Disadvantages: Newton-Raphson

- Requires the calculation of a derivative, and performs two function evaluations for each iteration
  - NR needs to evaluate $f\left(x^{(k)}\right)$ and $f'\left(x^{(k)}\right)$ for each iteration. Secant only needs $f\left(x^{(k)}\right)$ and uses $f\left(x^{(k-1)}\right)$ from memory.
  - You may need to fit a curve to the data to get an equation of the function / the derivative
    - The error in approximating that curve will cascade through the NR method
- Breaks down if the derivative is equal or close to zero: $f'\left(x^{(k)}\right) \approx 0$
  - A tangent of slope 0 will not cross the x-axis, and will never give the next iteration
  - A tangent of slope ≈ 0 may draw you father away from the solution with successive iterations
  - This can either lead to *divergence* or effectively give a new starting point, making all previous iterations and their progress virtually irrelevant
- The shape and properties of the function can play a role in finding the solution
  - *Monotonic function:* no matter how far out one iteration takes us, the next will always point back to the true solution. The quality of our initial guess is nearly irrelevant except to find the answer faster!
  - *Multiple root functions:* NR will only find one of these roots and it will strongly depend on your initial guess
  - Function shapes that cause *cycling:* the plot below shows an unfortunate starting position that can lead to cyclic iterations by the NR method (although this is unlikely, it can still happen)

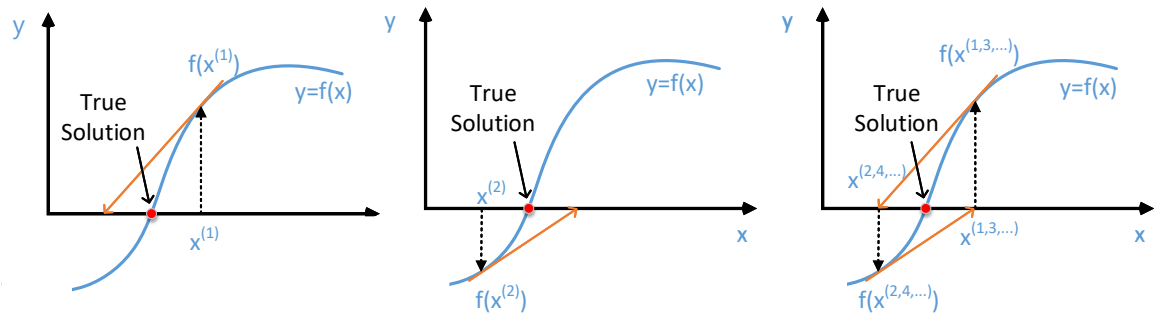


Figure 5: An example of cycling with the Newton-Raphson Method

Concluding Univariate Equations

We have covered lots of methods to solve univariate nonlinear equations! It is sometimes difficult to determine the most appropriate method to use. Common engineering software, such as `MATLAB` or `ASPEN PLUS`, use hybrids or a combination of these methods in efficient and reliable ways. One piece of advice is that if one method fails, try another! Or you can consider using a bracketing method to narrow in on a good *initial guess* for an open method, and let the open method take over from there.

Workshop: Brainstorming Numerical Methods

We have examined various numerical methods for solving single nonlinear equations. We have also briefly mentioned the use of hybrid methods, which combine open and bracketing methods to improve upon efficiency or robustness.

Together, let's see if we can use what we've learned to brainstorm ideas for a few new numerical methods.

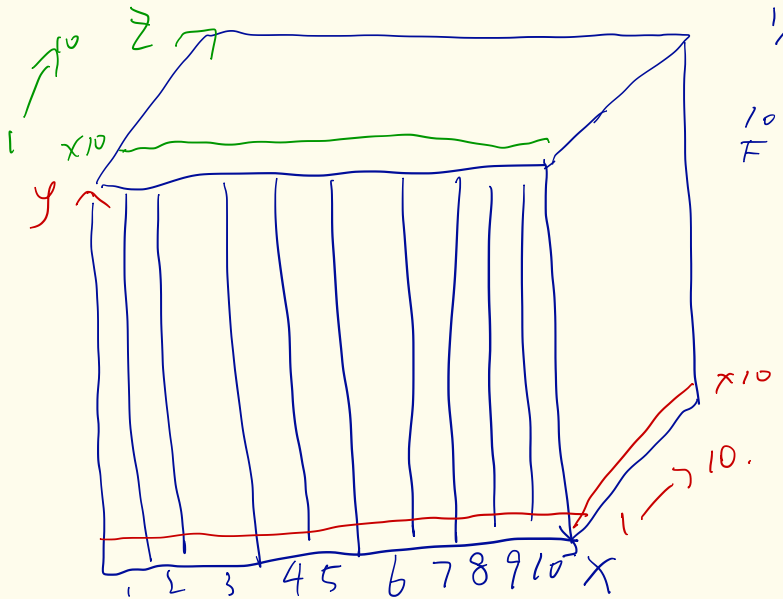
i. Can we combine certain methods to make a more efficient hybrid method? What trigger points would we use to switch from one method to another within our hybrid method?

ii. Are there any other, completely unique ideas to what we've covered that you can think of?

If you want, you can try to code some of the examples we come up with to see if they work! If they work, instant satisfaction is *guaranteed!*

100 100 100 100 100
A B C D E

100 100 100 100 100
F G H I J



10. 10 10 10 10
A B C D E

10 10 10 10 10
F G H I J

1 2 3 4 5
6 7 8 9 10.
F G H I J.

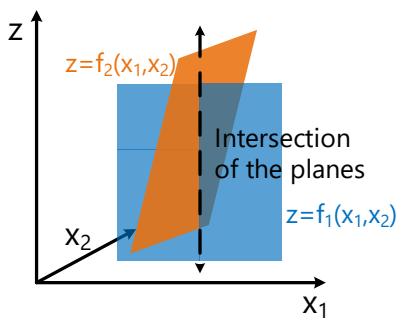
3.41.

3.22.

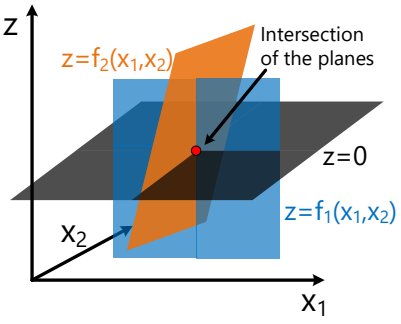
332

Solving Systems of Nonlinear Equations

Up to this point we have been trying to solve *one equation with one unknown*. However, the next question is: What if we need to solve multiple nonlinear equations at the same time? This corresponds to solving *systems* of nonlinear equations, where we have multiple variables across multiple equations.



Adding the $z = 0$ plane.



We're looking for this point shown above, where $f_1(x_1, x_2) = f_2(x_1, x_2) = 0$.

In general, this corresponds to the solution to the (*square*) equation system:

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{bmatrix} = \vec{\mathbf{0}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Luckily for us, the Taylor Series expansion and the methods derived from it (NR) are readily adaptable to multiple equations with multiple unknowns. Let's see how it works!

Multivariate Newton-Raphson Method

For a univariate system, we can approximate the value of a function near some point a, by evaluating the function and its derivative at that point. This can be extended to multivariate systems, but *we need to do it one variable at a time.* That is, we must find the Newton Step for each individual variable(Δx_i) *individually*, and it will likely be different for each variable for any given iteration.

Reminder: Newton Step Relating to Taylor Series

Recall that the Newton Step from NR, Δx, was derived by setting the Taylor Series approximation of $f(x) = 0$, and solving for the value of x that will accomplish this task. We are going to do the same thing below, but we must do it for *each possible variable* x_i one at a time.

Consider the point described by the position vector $\boldsymbol{x}^{(k)}$. Looking at $f_1\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right)$, Let's consider a Taylor Series expansion of this function centered on $\boldsymbol{x}^{(k)}$ and moving $x_1^{(k)}$ by some translation $\Delta x_1 = \left(x_1^{(k+1)} - x_1^{(k)}\right)$:

$$f_1\left(\underbrace{x_1^{(k)} + \Delta x_1}_{move}, \underbrace{x_2^{(k)}, \dots, x_n^{(k)}}_{unchanged}\right) = f_1\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right) + \left.\frac{\partial f_1}{\partial x_1}\right|_{x^{(k)}} \Delta x_1 + R_1^2$$

Again assuming R_1^2 (the Taylor Series remainder of $f_1\left(\boldsymbol{x}^{(k)}\right)$) is very small and approaches zero:

$$f_1\left(x_1^{(k)} + \Delta x_1, x_2^{(k)}, \dots, x_n^{(k)}\right) \approx f_1\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right) + \left.\frac{\partial f_1}{\partial x_1}\right|_{x^{(k)}} \Delta x_1$$

Next, consider the case where we adjust $f_1(\cdot)$ (the first system equation) for *all variables simultaneously*. We get:

$$f_1\left(x_1^{(k)}+\Delta x_1, x_2^{(k)}+\Delta x_2, \dots, x_n^{(k)}+\Delta x_n\right) = f_1\left(x_1^{(k)}, x_2^{(k)}, \dots x_n^{(k)}\right) + \left.\frac{\partial f_1}{\partial x_1}\right|_{x^{(k)}} \Delta x_1 + \left.\frac{\partial f_1}{\partial x_2}\right|_{x^{(k)}} \Delta x_2 + \dots + \left.\frac{\partial f_1}{\partial x_n}\right|_{x^{(k)}} \Delta x_n$$

Now remember, we really want to find $f\left(\boldsymbol{x}^{(k)}+\Delta\boldsymbol{x}\right)=0$ (the point where the system of ALL functions all equal zero). Therefore, make the RHS zero across ALL functions (f_n).

$$\begin{aligned}
&f_1\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right) + \left.\frac{\partial f_1}{\partial x_1}\right|_{x^{(k)}} \Delta x_1 + \left.\frac{\partial f_1}{\partial x_2}\right|_{x^{(k)}} \Delta x_2 + \dots + \left.\frac{\partial f_1}{\partial x_n}\right|_{x^{(k)}} \Delta x_n = 0\\
&f_2\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right) + \left.\frac{\partial f_2}{\partial x_1}\right|_{x^{(k)}} \Delta x_1 + \left.\frac{\partial f_2}{\partial x_2}\right|_{x^{(k)}} \Delta x_2 + \dots + \left.\frac{\partial f_2}{\partial x_n}\right|_{x^{(k)}} \Delta x_n = 0\\
&\qquad\qquad\vdots\\
&f_n\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right) + \left.\frac{\partial f_n}{\partial x_1}\right|_{x^{(k)}} \Delta x_1 + \left.\frac{\partial f_n}{\partial x_2}\right|_{x^{(k)}} \Delta x_2 + \dots + \left.\frac{\partial f_n}{\partial x_n}\right|_{x^{(k)}} \Delta x_n = 0
\end{aligned}$$

Multivariate Newton-Raphson

Multivariate NR can be conveniently written in matrix form:

$$\underbrace{\begin{bmatrix}
\left.\frac{\partial f_1}{\partial x_1}\right|_{x^{(k)}} & \left.\frac{\partial f_1}{\partial x_2}\right|_{x^{(k)}} & \dots & \left.\frac{\partial f_1}{\partial x_n}\right|_{x^{(k)}}\\
\left.\frac{\partial f_2}{\partial x_1}\right|_{x^{(k)}} & \left.\frac{\partial f_2}{\partial x_2}\right|_{x^{(k)}} & \dots & \left.\frac{\partial f_2}{\partial x_n}\right|_{x^{(k)}}\\
\vdots & \vdots & \ddots & \vdots\\
\left.\frac{\partial f_n}{\partial x_1}\right|_{x^{(k)}} & \left.\frac{\partial f_n}{\partial x_2}\right|_{x^{(k)}} & \dots & \left.\frac{\partial f_n}{\partial x_n}\right|_{x^{(k)}}
\end{bmatrix}}_{Jacobian\ (J)}
\underbrace{\begin{bmatrix}\Delta x_1\\ \Delta x_2\\ \vdots\\ \Delta x_n\end{bmatrix}}_{\Delta x}
= -\underbrace{\begin{bmatrix}f_1\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right)\\ f_2\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right)\\ \vdots\\ f_n\left(x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)}\right)\end{bmatrix}}_{f(x^{(k)})}$$

$$\boxed{J\left(\boldsymbol{x}^{(k)}\right)\Delta\boldsymbol{x} = -f\left(\boldsymbol{x}^{(k)}\right)}$$

The Jacobian, J, is a matrix of partial derivatives. *Rows* distinguish which system function the derivative is taken from, and *columns* represent what the derivative was taken with respect to.

Our Newton step is now a *vector* instead of a single value: $\Delta\boldsymbol{x}^{(k)} = \left(\Delta x_1^{(k)}, \Delta x_2^{(k)}, \dots \Delta x_n^{(k)}\right)$ .To find the next iteration, we must solve for $\Delta\boldsymbol{x}$:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta\boldsymbol{x}$$

Solving for $\Delta\boldsymbol{x}$:
$J^{-1}\left(\boldsymbol{x}^{(k)}\right)$ and $-f\left(\boldsymbol{x}^{(k)}\right)$ will only contain constants, which makes $J\left(\boldsymbol{x}^{(k)}\right)\Delta\boldsymbol{x} = -f\left(\boldsymbol{x}^{(k)}\right)$ a linear system. We can solve for $\Delta\boldsymbol{x}$ in using any of the methods from the Linear Algebraic Equations unit!

Tolerance/Stopping Conditions:

- $\left|\left|F\left(x^{(n)}\right)\right|\right| < \epsilon$ (the norm of $F(\boldsymbol{x})$ is sufficiently close to zero)
- $k \geq \mathcal{M}$ (the maximum number of iterations has been reached)

Note: Multivariate NR is still susceptible to the same pitfalls of univariate NR.

Unit Relationships

Throughout this course, there will be many connections across units - the end goal of this course is to *get you from raw data points to the end solution* as effectively and efficiently as possible. Obviously, we haven't covered every link-in-the-chain yet, but as we tackle new ideas, think about how the units build-off one another.

Workshop: Jacobian

Determine the Jacobian for the following system of equations and compute $J(x^{(k)})$ and $F(x^{(k)})$ when $x^{(k)} = [1,2,1]^T$.

$$F(x) = \begin{bmatrix} x_1^3 + x_2^5 + x_3 \\ 10x_1^2x_2 + 2x_3^4 \\ 2x_1x_2^3x_3^2 \end{bmatrix} = 0$$

$x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

$$J = \begin{bmatrix} 3x_1^2 & 5x_2^4 & 1 \\ 20x_1x_2 & 10x_1^2 & 8x_3^3 \\ 2x_2^3x_3^2 & 6x_1x_2^2x_3^2 & 4x_1x_2^3x_3 \end{bmatrix} \quad J(x^{(k)}) = \begin{bmatrix} 3 & 80 & 1 \\ 40 & 10 & 8 \\ 16 & 24 & 32 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 80 & 1 \\ 40 & 10 & 8 \\ 16 & 24 & 32 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = \begin{bmatrix} -34 \\ -22 \\ -16 \end{bmatrix} \quad F(x^{(k)}) = \begin{bmatrix} 34 \\ 22 \\ 16 \end{bmatrix}$$

Conclusion and Summary

And that concludes the chapter for nonlinear systems!

In this module and chapter we have covered:

- The Secant and Newton-Raphson Methods to numerically solve nonlinear equations – there are many more out there!
- No nonlinear system model is perfect (every time) – they all have their own advantages and disadvantages.
- Bracketing methods guarantee a solution if there is one between the bounds. But time is a constraint in many situations – you may not be able to afford the computational time required for a bracketing method so open methods can come in handy!

An understanding of your available resources (speed, accuracy, and computational resources), and model (the nonlinear equation) are needed to properly pair the best nonlinear solver to the given scenario.

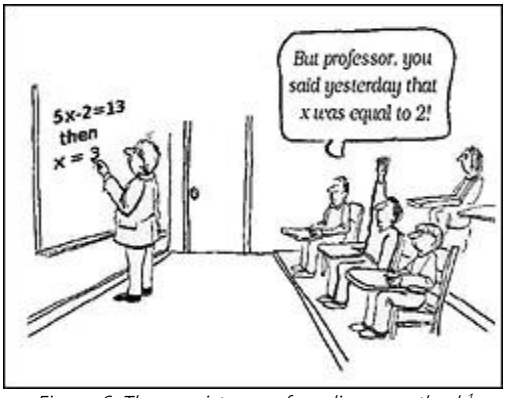


Figure 6: The consistency of nonlinear methods[1]

Next up: Curve Fitting

[1] https://web.math.princeton.edu/~seri/homepage/realtive_math1.jpg