

Numerical Methods Lecture 3 Nonlinear Equations and Root Finding Methods

Lecture covers two things:

- 1) Solving systems of linear equations symbolically
- 2) Using Mathcad to solve systems of nonlinear equations
- 3) Investigating algorithms to find roots of equations

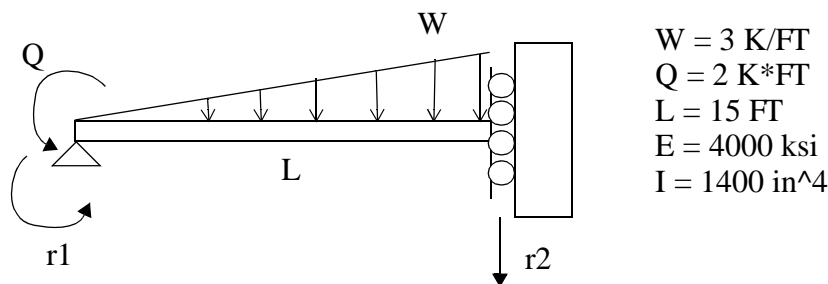
Solving Systems of Linear Equations Symbolically

Let's take a look at a very powerful tool in Mathcad that started a revolution in computational analysis. It started in the late 1980's when I was an undergraduate. A company called Wolfram created a computer program called Mathematica. This was the first computer code that could solve algebraic and calculus equations **symbolically**. That is, if I had an equation that said $x*y = z$, Mathematica could tell me that $y = z / x$, without ever needing me to assign numbers to x , y , or z . It also was able to solve integrals, differential equations, and derivatives symbolically. This was an incredible advance, and opened the doors to a whole new world of programming, numerical methods, pure mathematics, engineering, and science.

Since then, a competing code called Maple was developed and sold itself to other software companies to include in their programs.

The end result: Mathcad uses Maple as a solving engine in the background (you don't see it) to solve problems symbolically. Here we will look at a brief example of how to use this capability in the context of solving a system of linear equations.

Example: The structural system below is something you will see in CES 3102 or CES 4141. r_1 and r_2 are labels that indicate how the ends of the beam are allowed to move. Q and W represent the external loads (a couple and a distributed load, respectively), and material properties are given as E and I . L is the length of the beam. The goal is to solve for the amount or rotation at r_1 , and the deflection at r_2 that occurs for the given loads. This would help us to solve for internal stresses, allowing us to design the beam to survive these internal forces.



Solution: The way we learn to solve this problem in CES 4141 is using a Matrix-based solution procedure. The generic form of the solution is

$$K * r = R$$

K is a 2×2 'stiffness' matrix that contains information about the structures shape, boundary conditions and material properties. THE information needed is L , E , and I

r is a 2×1 vector that contains the unknown rotation and displacement quantities sought.

R is a 2×1 vector that contains only information about the external loads (W and Q for this problem).

Solution continued:

So we will have K as a known 2 x 2 stiffness matrix
 We will have R as a known 2 x 1 load vector
 We will solve for the unknown displacement vector r

We will not go into any detail on HOW we fill in the values for K and R, that's for another class. Here we will just consider how to solve the system of matrix equations symbolically.

Define Stiffness K as a function of E, I, L

Define Load Vector R as a function of Q, W, L

$$K(E, I, L) := \begin{pmatrix} \frac{4 \cdot E \cdot I}{L} & \frac{6 \cdot E \cdot I}{L^2} \\ \frac{6 \cdot E \cdot I}{L^2} & \frac{12 \cdot E \cdot I}{L^3} \end{pmatrix}$$

$$R(Q, W, L) := \begin{pmatrix} Q - \frac{W \cdot L^2}{30} \\ \frac{7}{20} \cdot W \cdot L \end{pmatrix}$$

known information

The variables in the parenthesis (E,I,L) and (Q,W,L) is telling mathcad that the variables K and R, respectively, will be a function of those variables.

Show me the inverse of the stiffness matrix
 Not a necessary step, but pretty cool see the matrix version of 1/K

$$K(E, I, L)^{-1} \rightarrow \begin{bmatrix} \frac{1}{(E \cdot I)} \cdot L & \frac{-1}{(2 \cdot E \cdot I)} \cdot L^2 \\ \frac{-1}{(2 \cdot E \cdot I)} \cdot L^2 & \frac{1}{(3 \cdot E \cdot I)} \cdot L^3 \end{bmatrix}$$

Symbolic Inverse

Note the arrow --> is the way to initiate a symbolic operation. Here we are asking what is the inverse of K ? The arrow comes from the view => toolbars => symbolic pad

Calculate the displacement vector as $K^{-1} \cdot R$

Note that we DO need to keep writing the names of the independent variables next to K, r, and R to tell Mathcad what symbols to look for

$$r(E, I, L, Q, W) := K(E, I, L)^{-1} \cdot R(Q, W, L)$$

Symbolic Calculation

Display the resulting displacement vector calculated in terms of W, Q, L, E, I

$$r(E, I, L, Q, W) \rightarrow \begin{bmatrix} \frac{1}{(E \cdot I)} \cdot L \cdot \left(Q - \frac{1}{30} \cdot W \cdot L^2 \right) - \frac{7}{(40 \cdot E \cdot I)} \cdot L^3 \cdot W \\ \frac{-1}{(2 \cdot E \cdot I)} \cdot L^2 \cdot \left(Q - \frac{1}{30} \cdot W \cdot L^2 \right) + \frac{7}{(60 \cdot E \cdot I)} \cdot L^4 \cdot W \end{bmatrix}$$

Symbolic Solution

Note the arrow displays a symbolic result

We can now send specific values into the function for r by first assigning numbers to the parameters. Note that units are not being declared, so its up to the user to be consistent.

$E := 4000$ $I := 1400$ $L := 15 \cdot 12$ Here are the material/geometric properties

$W := \frac{3}{12}$ $Q := 2 \cdot 12$ Here are the external load values

Finally, we can now ask Mathcad to evaluate the results of sending the parameters into the function we created above. Now that E, I, L, Q, W have numbers, they will be substituted into the equations to get answers. Be sure to use the same order for the parameters as defined above.

$$r(E, I, L, Q, W) = \begin{pmatrix} -0.053 \\ 6.179 \end{pmatrix}$$

Now let's change the external load values and get new answers

$W := \frac{-5}{12}$ $Q := 4 \cdot 12$

$$r(E, I, L, Q, W) = \begin{pmatrix} 0.092 \\ -10.553 \end{pmatrix}$$

Solving Systems of Nonlinear Equations

We won't go into the algorithms themselves here. We will just focus on how to use Mathcad to solve the problem.

Use Mathcad help and use the keywords 'nonlinear equations' to get some information. Follow the link to the 'Find' function in Mathcad. 'Find' is the workhorse that gets the solution. All you need to do is give it the correct description of the problem. This includes the equations to solve, any possible constraints to the solutions you are interested in, and a place to start looking for a solution (i.e., initial guess for the solution). Here we will go straight into an example to show how it's done.

ORIGIN≡ 1

Using the Find function to solve systems of nonlinear equations
 This example will solve for the intersecting values of the following system of 2 equations

$$x^2 + (y - 2)^2 = 8$$

$$x + .15x^2 + y = 1$$

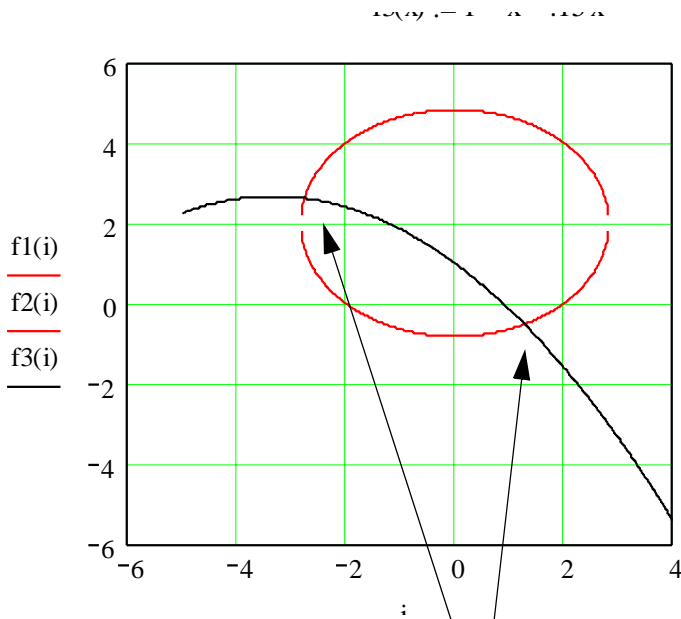
Let's first take a look at the solution visually by plotting
 We will create some functions that allow us to plot the two equations above

$$f1(x) := \sqrt{8 - x^2} + 2$$

$$f2(x) := -\sqrt{8 - x^2} + 2$$

$$f3(x) := 1 - x - .15x^2$$

i := -5, -4.99.. 4



We can see that there are two solutions to the two equations above.
 We'll try to find both solutions by using the 'find' function

Use the help desk with the keyword 'find function'

Step one: start by guessing the solution to the set of two equations we are trying to solve. Let's guess that $x=4$ and $y=-1$

$$x := 4 \quad y := -1$$

Next we create what is called a 'solve block'. This defines the equations to be solved.

- 1) type the word 'Given'
- 2) type below 'Given' all the equations to be solved

Note that we use the 'boolean' tab to get the equal sign, **not** the keyboard stroke.

Given

$$x^2 + (y - 2)^2 = 8$$

$$x + .15 \cdot x^2 + y = 1$$

Finally, we use the 'Find' function to solve by sending in the two guesses we made above for x and y

$$\text{Find}(x, y) = \begin{pmatrix} 1.278 \\ -0.523 \end{pmatrix}$$

'Find' starts looking at the initial guess values, and iteratively updates the values of x and y until a pair is found that solves the equations defined in the solve block.

The results give the x and y values corresponding to **one** of the solutions. find where $x=1.278$ and $y=-0.523$ in the graph above 'Find' found this solution because the initial guesses were closer to this solution.

Now let's find the other solution seen in the graph, we'll redefine the initial guesses to somewhere near the other solution and use 'Find' again

$$x := -4 \quad y := 2$$

Given

$$x^2 + (y - 2)^2 = 8 \quad x + .15 \cdot x^2 + y = 1 \quad \text{Find}(x, y) = \begin{pmatrix} -2.76 \\ 2.617 \end{pmatrix}$$

Notice now that the solution changes to the other one shown in the earlier graph

What else can we do with this solve block?
Suppose we want only to find solutions if $x > 0$
that is, we want to tell Mathcad to not even bother with solutions if $x < 0$

We can do this by adding **constraints** to the solve block

$$x := -4 \quad y := 2$$

Given

$$x^2 + (y - 2)^2 = 8 \quad x + .15 \cdot x^2 + y = 1$$

$$x > 0 \quad \text{Adding the constraint } x > 0$$

$$\text{Find}(x, y) = \begin{pmatrix} 1.278 \\ -0.523 \end{pmatrix}$$

Solution changes to the first one we found earlier, even though the initial guess is closer to the solution with the negative x value

Solving for Roots of Equations

What?

- Locating where some function crosses the x-axis

$$\text{Find } x \text{ such that } f(x) = 0$$

Why?

- We can then find where $f(x)$ crosses any value

$$\text{Find } x \text{ such that } f(x) = 49.35$$

or

$$\text{Find } x \text{ such that } g(x) = f(x) - 49.35 = 0$$

When?

- Numerical methods in general are:

1) less accurate

2) slower

than using an analytical (exact) solution.

- Use numerical methods when analytical answers not available

Example:

Given: $y = 5 + 6x$, Find x such that $y = 0$

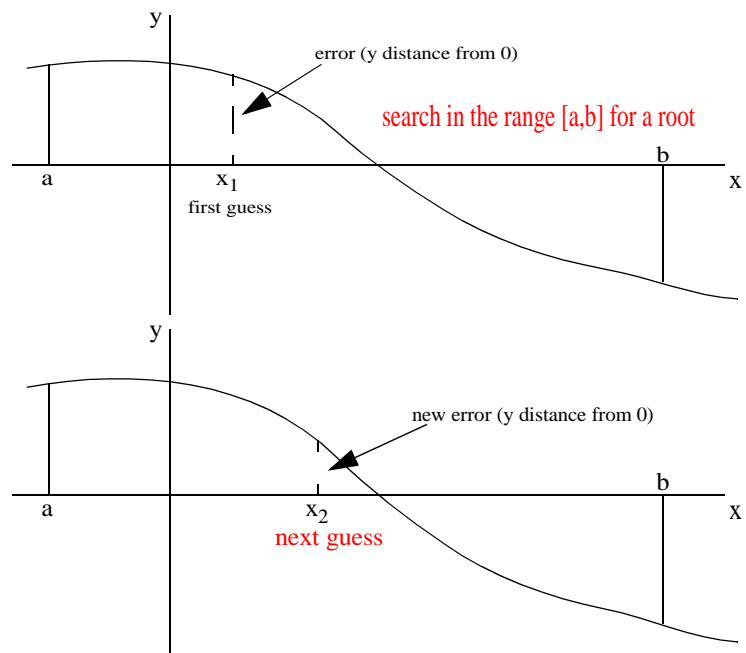
analytical solution: $0 = 5 + 6x \implies x = -5/6$ no need for

Given: $y = \exp(\sin(x)) - 3 \sin(1.5x) - 5$, Find x such that $y = 0$

No idea? Good, a prime candidate for a numerical solution

We will consider several methods, each does the following in some way:

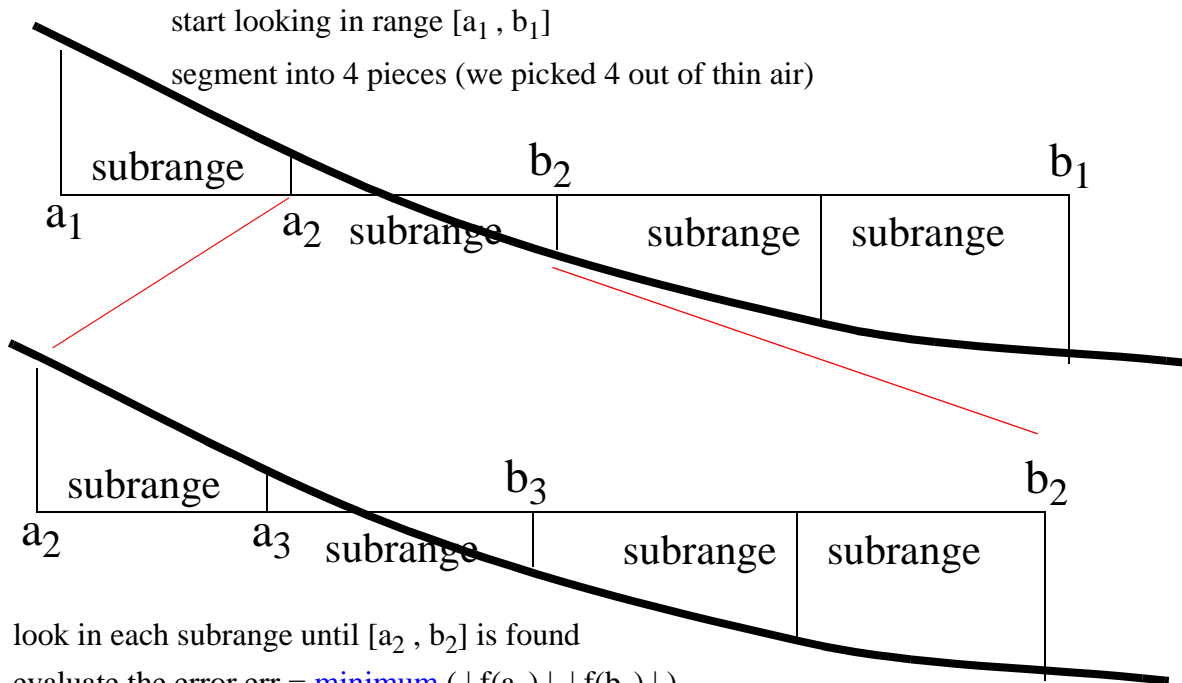
- 1) Start with some initial guess (or range in which to look)
- 2) Use some information about the function $f(x)$ to make another, closer guess
- 3) Stop when our current guess is 'close enough' to a solution



The user will decide what is 'close enough' (how big the error can be before we can stop looking)

Method 1: INCREMENTAL SEARCH METHOD

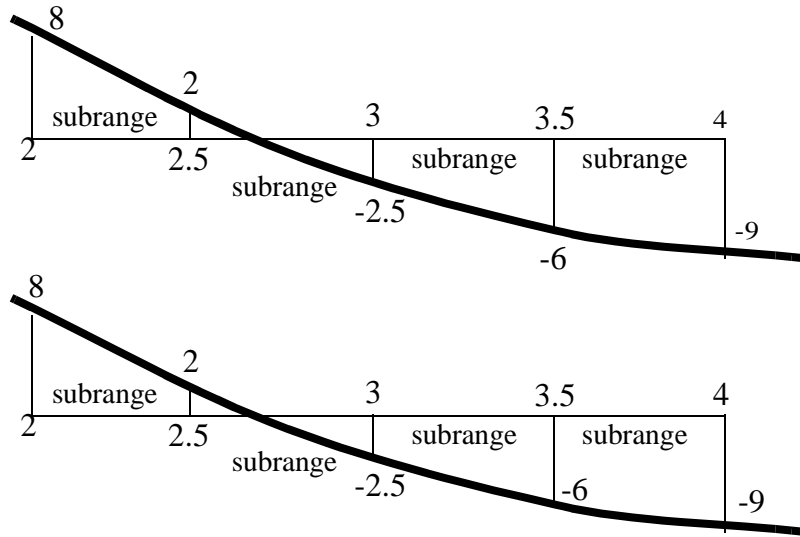
- 1) Start with an initial range that contains the root, and subdivide it into several smaller sub-ranges.
- 2) Look inside each sub-range one by one for the root. When the sub-range containing the root is identified, choose either end of the range as the guess.
- 3) Evaluate the error in your guess. If error is too big, subdivide your new range into smaller sub-ranges.
- 4) Loop back to step 1) and continue the loop until the error in step 3) is small enough



look in each subrange until $[a_2, b_2]$ is found
 evaluate the error $\text{err} = \text{minimum} (|f(a_2)|, |f(b_2)|)$
 Now divide $[a_2, b_2]$ into 4 segments and look again
 look in each subrange until $[a_3, b_3]$ is found
 evaluate the error $\text{err} = \text{minimum} (|f(a_3)|, |f(b_3)|)$
 Now divide $[a_3, b_3]$ into 4 segments and look again ...

- Out of the 4 subranges, we pick the one in which the function crosses the x-axis as the new range to subdivide even smaller.
- Visually we can see this, but how does the algorithm know how to identify the smaller range?
- Remember we have the function, so we can evaluate it at any x location we choose.

Take a moment, work out a way to identify the new smaller subrange.

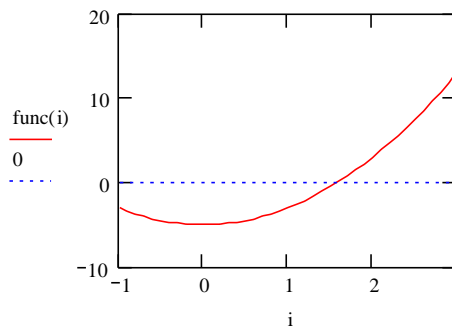


Incremental Search Root Finding Algorithm

```

incSearch(numsegs , tol, func , a, b) :=
    err ← min(|func(a)|, |func(b)|)
    while err ≥ tol
        dx ← (b - a) / numsegs
        for i ∈ 1.. numsegs
            x ← a + (i - 1) · dx
            prod ← func(x) · func(x + dx)
            if prod ≤ 0
                a ← x
                b ← x + dx
        err ← min(|func(a)|, |func(b)|)
    root ← a if |func(a)| < |func(b)|
    root ← b otherwise
    root
    
```

$func(x) := -5 + 2 \cdot x^2$
 $i := -1, -0.9..3$



IncSearch(4, .01, func, -1, 3) = 1.582

Something new to notice in how we use the above function....we'll discuss in class the idea of passing a function into another function. Note that the equation we are solving in the above example is not explicitly typed into the IncSearch function. Is this a good thing? Why?

This is a 'brute force' method that uses very little information about the function to improve the guess.

Method 2: BISECTION METHOD

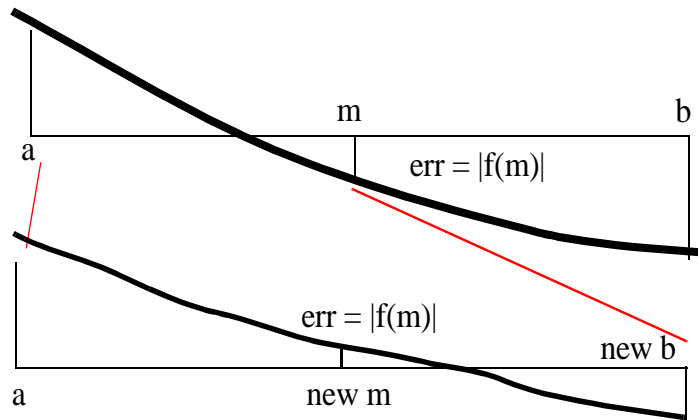
1) Start with an initial range $[a, b]$, and cut the range in half, assume this half point m is the root

2) Evaluate the error $err = |f(m)|$.

3) If error is too big, decide if the root is to the left or right of $[m]$.

- If root is to the left of m , reassign a new range $a = a, b = m$
- Else If root is to the right of m , reassign a new range $a = m, b = b$

4) Go back to step 1), stop bisecting when the error at m is 'small enough'.



evaluate the error $err = |f(m)|$

$err > tol$ - yes?

make new range:

root is to left of $m \implies a = a, b = m$

root is to right of $m \implies a = m, b = b$

divide new $[a, b]$ in half to get new m

evaluate the error $err = |f(m)|$ etc.

Three Bisection iterations

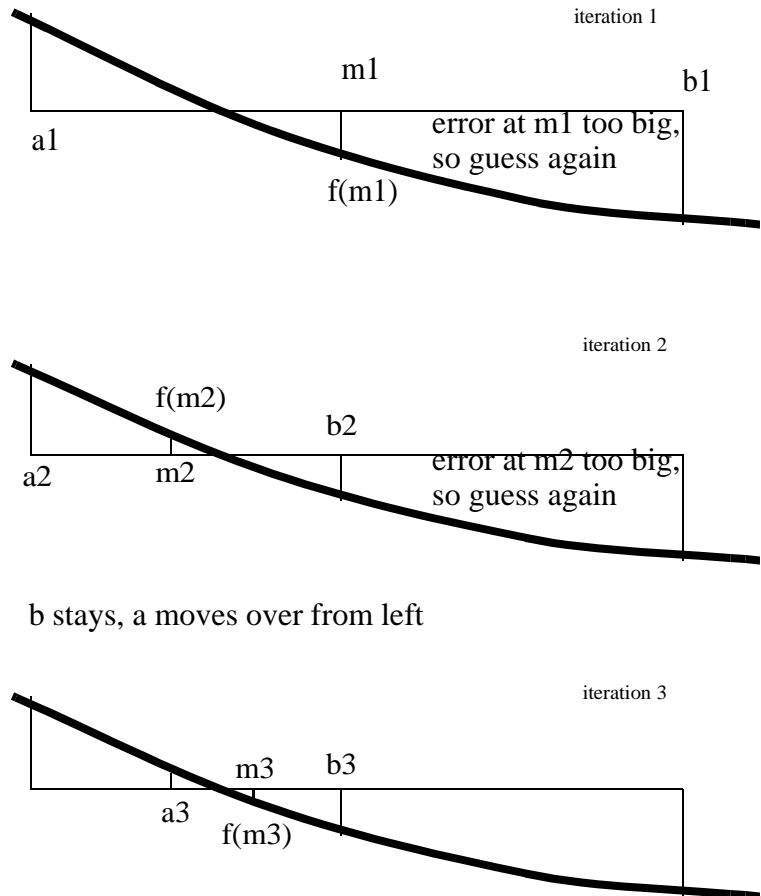
- Eventually we zero in on the root
- How do we decide where the root is relative to m for the next $[a, b]$??

Same as the incremental search:

$$prod = f(a) * f(m)$$

if $prod < 0$, root is to the left

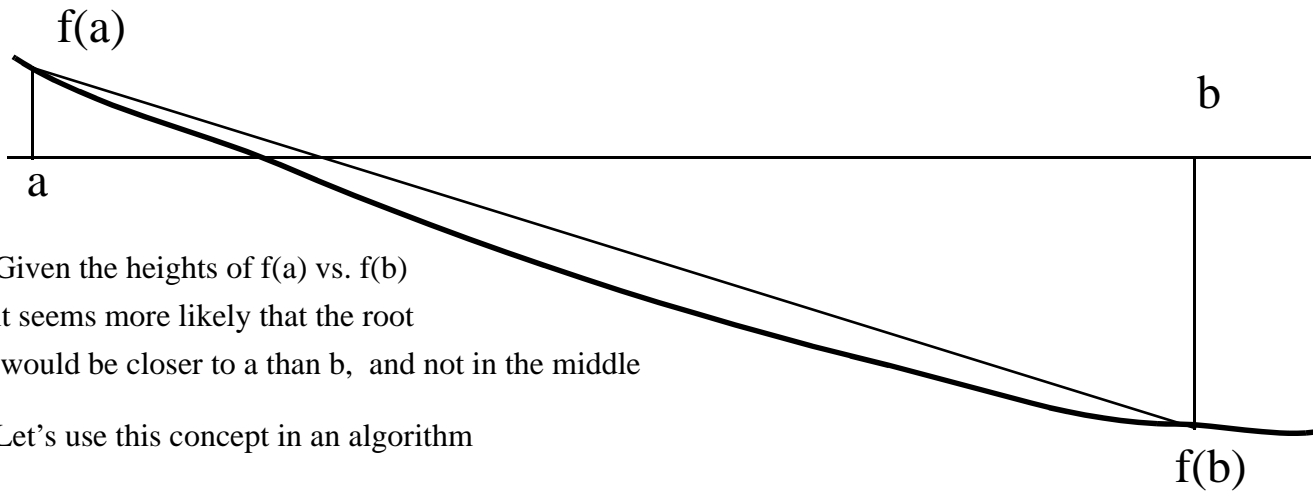
otherwise root is to the right



- This slow method does not use much function information. We can do better.

Method 3: FALSE POSITION

- Relative height of function at end points used to make better guesses



Given the heights of $f(a)$ vs. $f(b)$
it seems more likely that the root
would be closer to a than b , and not in the middle

Let's use this concept in an algorithm

- 1) Define initial range $[a \ b]$ (possibly the result of a single pass of the incremental search method).

first guess c - intersection of a line from $f(a)$ to $f(b)$ with x -axis

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)} \quad (\text{from similar triangles})$$

- 2) Evaluate the error $\text{err} = |f(c)|$

If error is too big, decide if the root is to the left or right of $[c]$.

for $f(a) * f(c) < 0$, reassign a new range $a = a$, $b = c$

for $f(a) * f(c) > 0$, reassign a new range $a = c$, $b = b$

$$\text{new guess } c = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

- 3) Stop iterating when the error is small enough

Two False Position iterations

- Eventually we zero in on the root
- Tends to zero in faster than using bisection method. why?
- This method uses more function information than bisection

bisection: only knows if f(a), f(m) are positive or negative

false position: Uses +/- of f(a) and f(c), and uses their relative magnitudes to make next guess at c

Method 3: Root finding - Newton Raphson method

Incremental search - uses sign of f(a) and f(b)

Bisection - uses sign of f(a) and f(b)

False position - uses sign and relative magnitude of f(a) vs. f(b)

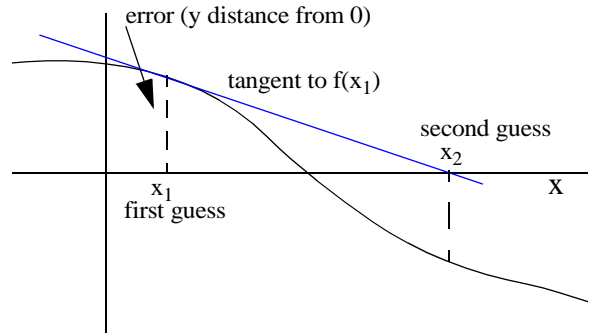
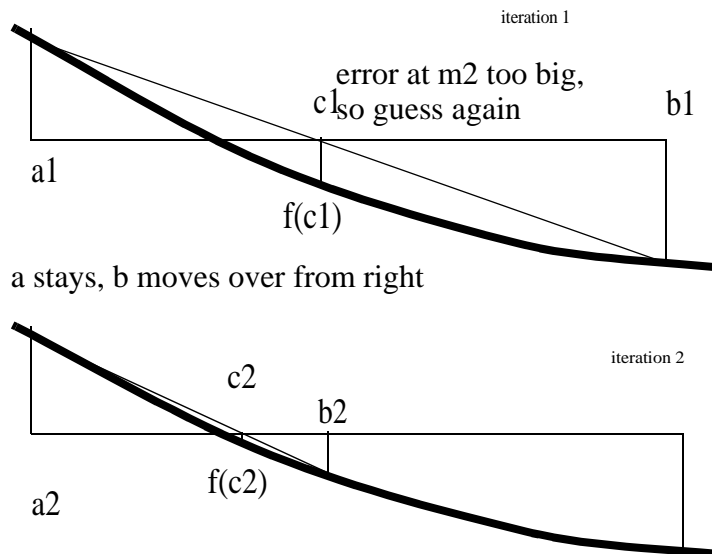
Newton Raphson - uses derivative of f(x) $df(x)/dx$

- 1) Pick a single point (not a range) as an initial guess $x(1)$
- 2) evaluate the error $err = |f(x(1))|$
- 3) Next guess - draw tangent line from initial guess to the x-axis. New guess, $x(2)$, is the intersection of the tangent line with the x-axis.
- 4) Back to step 2 to evaluate error of new guess ...etc.

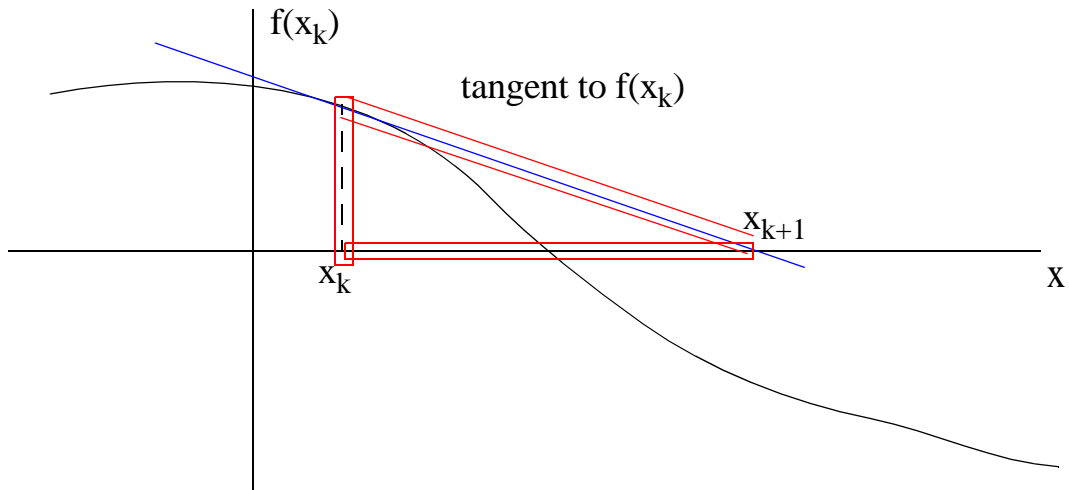
Tangent line (slope) is given by derivative of f(x)

General formula for next guess $x(k+1)$ in terms of previous guess $x(k)$

new guess
$$x_{k+1} = x_k - \frac{f(x_k)}{df(x_k)/dx}$$



this is from the definition of slope



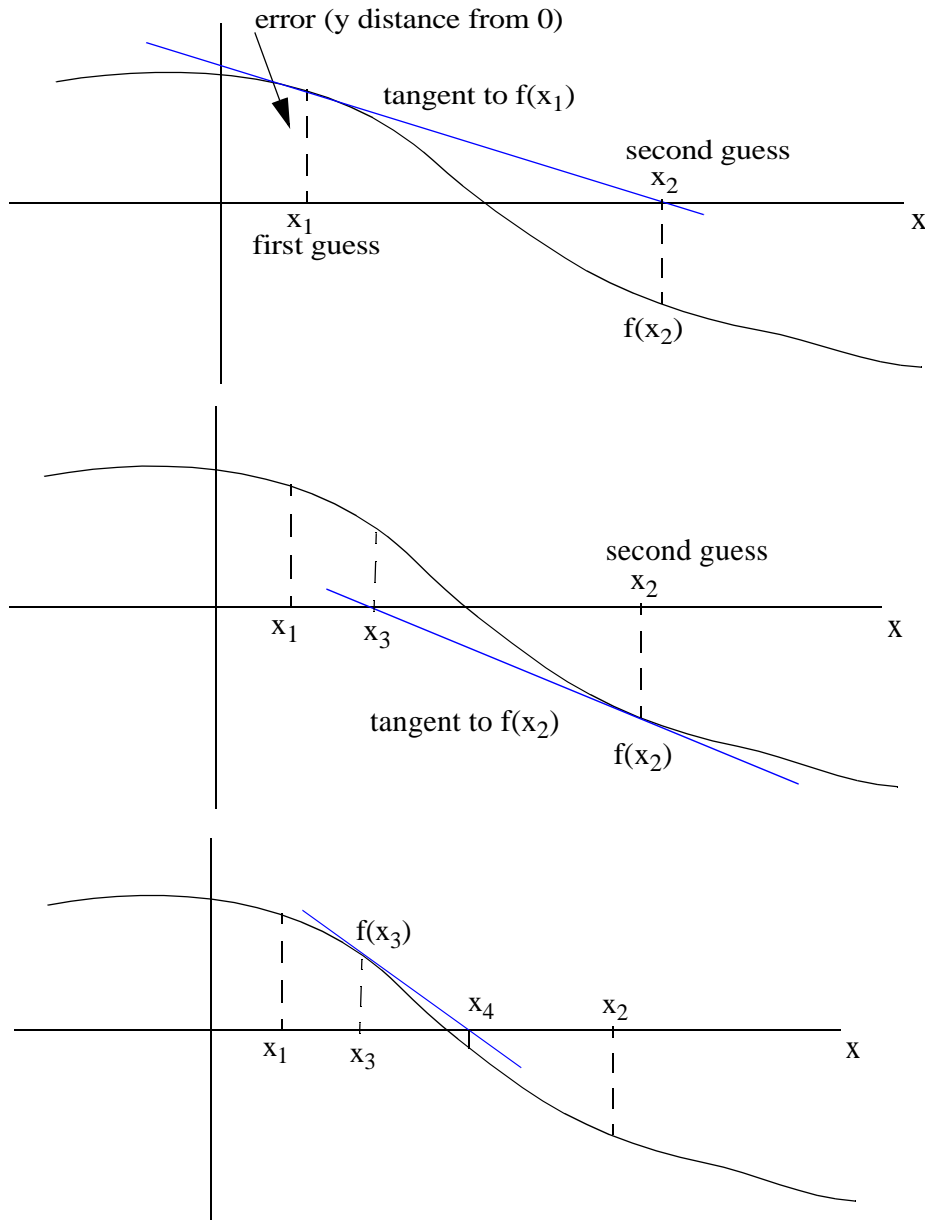
$$\text{slope} = \frac{df(x_k)}{dx}$$

$$\text{slope} = \frac{\text{rise}}{\text{run}}$$

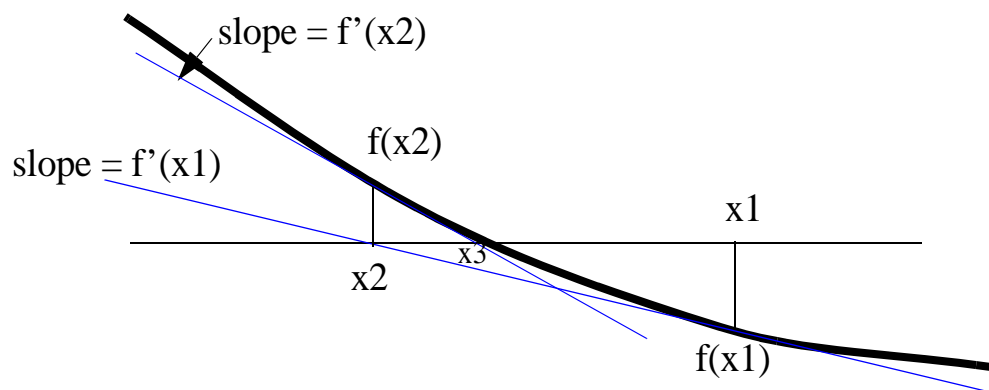
$$\text{slope} = \frac{0 - f(x_k)}{x_{k+1} - x_k}$$

solve for x_{k+1}

Let's continue with a few more iterations



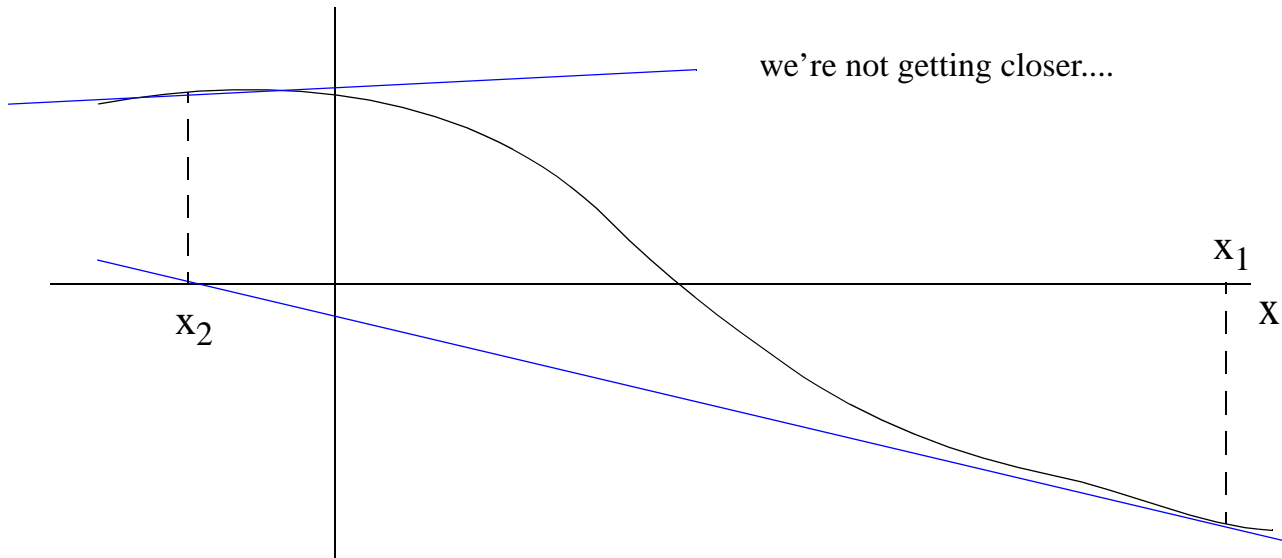
Another example, 2 iterations in one figure



Some Comments

- 1) Quickly converges to the root under the 'right' conditions
- 2) Can be divergent (a very bad word) if initial guess not close to the root
 Must have condition in the indefinite loop to stop if divergent
 may need incremental search to narrow the range

example: x_1 is not close to the root, tangent line send us off in wrong direction



- 3) Requires the function AND the function's derivative

Subtle Foreshadowing

- What if an equation for the derivative $df(x)$ is not readily available ????

Sure would be nice to have an algorithm that numerically estimates the derivative of any function ...HHmmmm...

Good test question:

Under what conditions will Newton Raphson find the exact root after a single iteration??