Roots of Equations

Ch. 5 &6

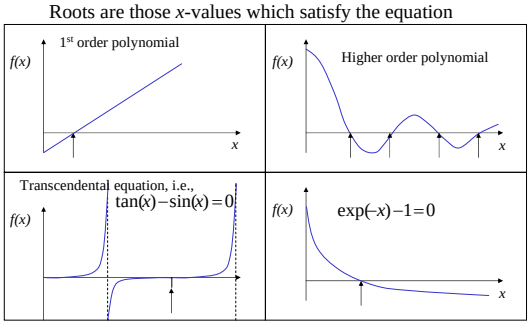
Lecture Objectives

- To understand why we as Engineers are interested in finding the roots of algebraic equations
- To understand the difference between *Open* and *Bracketing* methods & to know <u>why</u> and <u>when</u> we would choose to apply one method over another
- To be able to implement your own root finding tools

Roots of Equations

Many Mathematical functions of Engineering relevance can be expressed as: $f(x)=0$

Roots are those *x*-values which satisfy the equation



Roots of Equations – "Zeros"

Examples :

1. Quadratic Formula – roots for 2nd order polynomials.

$$f(x) = ax^2 + bx + c = 0$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

2. Most Practical Equations are more complicated:

$$\tan(x) - x = 0$$

$$e^{-x} - x = 0$$

$$v - \frac{gm}{c}\left(1 - e^{-c/mt}\right) = 0$$

Roots of Equations

- Explicit Equation – the unknown is specified explicitly in terms of the other variables & parameters.

$$y - mx + b = 0$$

- Implicit Equation – The desired variable *can not* be isolated on the LHS of an equation.

$$v - \frac{gm}{c}\left(1 - e^{-c/mt}\right) = 0$$

Roots of Equations

- Explicit Equation – the unknown is specified explicitly in terms of the other variables & parameters.

$$y - mx + b = 0$$

- Implicit Equation – The desired variable can not be isolated on the LHS of an equation.

$$v - \frac{gm}{c}\left(1 - e^{-c/mt}\right) = 0 \longrightarrow$$

Numerical methods must be applied to obtain roots of implicit equations

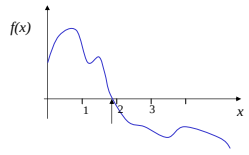
Ch. 5 – Bracketing Methods

TWO guesses are required to bracket either side of a root.

- Graphical Method
- Bisection Method
- False Position Method

(A) Graphical Method

1. Make a plot of *f(x)*
2. Observe where it crosses the x-axis
3. Plug the estimate back into the original equation to check
   - "Rough Estimate" approach
   - Limited due to imprecision
   - Provides some understanding of the functions behavior
   - May be hard to find all roots due to varying ranges



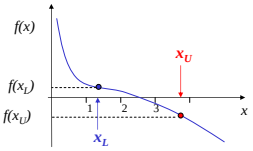
(B) Bisection Method – Brute-Force

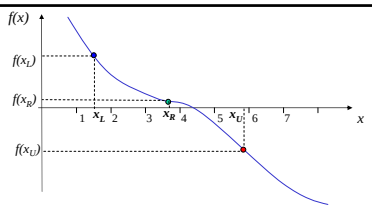
Utilize the notion that the function must change signs on either side of the root.

"If f(x) is real & continuous on the interval from x_L to x_U and $f(x_L)$ & $f(x_U)$ have opposite signs, then

$$f(x_L) \cdot f(x_U) < 0$$

And there is at least one root between x_L to x_U





Bisection Method:

1. Choose lower (x_L) & upper (x_U) guesses of the root, such that the function changes signs over the interval. This can be checked by ensuring
$$f(x_L)\cdot f(x_U)<0$$
2. Estimate the root (x_r) at the midpoint of the guesses: $x_r = \frac{x_U + x_L}{2}$
3. Determine where the root lies:
   - If $f(x_L)*f(x_r)<0$ the root lies in the lower sub interval. Set $x_U = x_r$ → *goto step 2*
   - If $f(x_L)*f(x_r)>0$ the root lies in the upper sub interval. Set $x_L = x_r$ → *goto step 2*
   - If $f(x_L)*f(x_r)=0$ → the root is x_r

Bisection Method – Termination criteria

ε_a = Approximate relative % error for the root x_r

$$\varepsilon_a = \left|\frac{x_r^{new} - x_r^{old}}{x_r^{new}}\right| \times 100\%$$

If ε_a < some stopping criteria → terminate calculation

While ε_a is only an estimate for the true error ε_t, it turns out That for the Bisection Method:

$\varepsilon_a > \varepsilon_t$

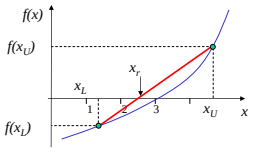
→The Number of iterations required to get a desired absolute error $E_{a,d}$ is:

$$n = \frac{\log\left(\Delta x^o / E_{a,d}\right)}{\log 2}$$

$$\Delta x^o = x_U^o - x_L^o$$

(C) False Position Method

- Take Advantage of the magnitude of $f(x_L)$ & $f(x_U)$
- The root will most likely be closest to the value of x for which $f(x)$ is closer to zero.
- Draw a straight line between $f(x_L)$ & $f(x_U)$, the intersection with the x-axis represents an improved root estimate.

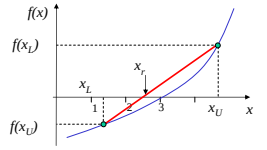


(C) False Position Method

- From similar triangles:

$$\frac{f(x_L)}{x_R - x_L} = \frac{f(x_U)}{x_R - x_U}$$

$$x_R = x_U - \frac{f(x_U)(x_L - x_U)}{f(x_L) - f(x_U)}$$



Linear interpolation method

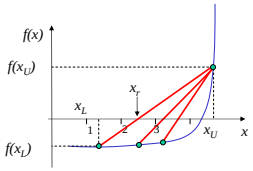
(C) False Position Method – Implementing

```
iter = 0                                    %initialize iteration counter
While ea>es and iter <= imax
    xrold = xr
    xr = (xl + xu)/2                        % Estimate new root
    iter = iter + 1                         % increment iteration counter
    if xr ≠ 0 then
        ea = abs((xr – xrold)/xr)*100       % calculate approx. rel. error
    endif
    test = f(xl)*f(xr)                      % good spot to use a Matlab function
    if test < 0 then
        xu = xr
    else if test > 0 then
        xl = xr
    else
        ea = 0
    endif
end while loop
Bisect = xr
```

How might we modify our Bisection Method to implement the False Position Method?

(C) False Position Method

- To check your root substitute your estimate back into the original equation – see if it is satisfied.
- Problem with False Position Method – Certain types of functions can have slow convergence:



Ch. 6 –Open Methods

1 guess is required as value to determine a root. Method Can Diverge! Usually much faster convergence.

- Successive Substitution
- Newton-Raphson Method
- Secant Method

(A) Successive Substitution Iterative Technique

- Rearrange the function f(x) = 0 so that x is explicit:

$$x = g(x)$$

- This gives us the iteration formula

$$x_{i+1} = g(x_i)$$

- Error Estimate

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \times 100\%$$

- Choice of g(x) is important, as Divergence can occur for the "wrong" formulation.

(A) Successive Substitution Iterative Technique

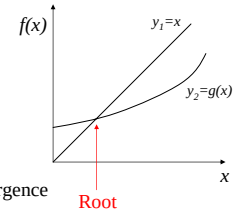
- Example:

$$f(x) = 3x^3 + x - 4x + 1 = 0$$

See Matlab suciter.m code

(A) Successive Substitution
Possibility of Convergence

- Let $y_1 = x$
  $y_1 = g(x)$ Plot these function on a graph



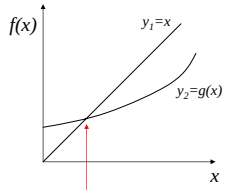
Linear Convergence Rate:

$\varepsilon_{t,i+1} \propto \varepsilon_{t,i}$

Root

(A) Successive Substitution
Possibility of Convergence

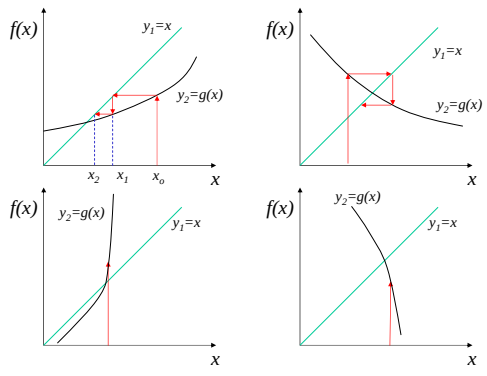
- Let $y_1 = x$
  $y_1 = g(x)$ Plot these function on a graph



Root

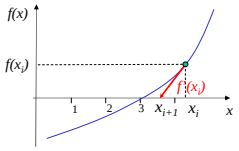
We will obtain convergence when The slope of g(x) is less than x.
i.e. $|g'(x)| < 1$

Convergence & Divergence of Simple Fixed Point Iteration Technique



(B) Newton-Raphson Method

- A very commonly used Root Finding Method
- Uses the slope of the function to Zoom in on a Root



$$f'(x_i)=\frac{f(x_i)-0}{x_i-x_{i+1}}$$ First order TS expansion

$$\boxed{x_{i+1}=x_i-\frac{f(x_i)}{f'(x_i)}}$$ ← Newton-Raphson

(B) Newton-Raphson Method: Error

- Using a TS expansion, the true error can be shown to have quadratic convergence:

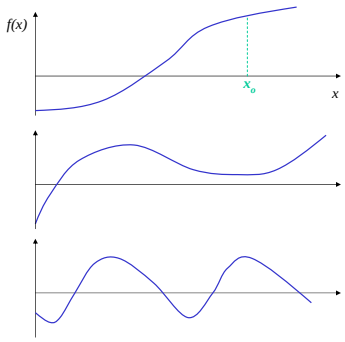
$$E_{t,i+1}=O\left(E^2_{t,i}\right)$$

- The current error is proportional to the square of the previous error → Rapid convergence

(B) Newton-Raphson Method: Problems

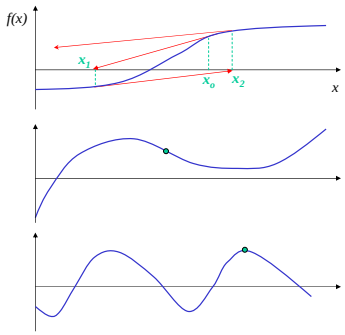
1. Slow convergence for certain functions
2. Poor Convergence - inflection points near a root – progressively diverges
3. Local Maxima or minima can lead to oscillations or divergence (Near Zero Slope Problem)
4. Jumping away from a local root
5. Multiple roots

(B) Newton-Raphson Method: Examples of functions with Poor Convergence



$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

(B) Newton-Raphson Method: Examples of functions with Poor Convergence



$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$f'(x_i) \to 0$$

Fig. 6.6 text

(B) Secant Method

- How do we implement the Newton-Raphson method if we do not have an expression for *f'(x)*

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- *Use a finite difference method (backward diff.)*

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

- *Secant Method*

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

Pseudocode for Newton-Raphson technique

```
es=eo                          % initialize error stopping criteria with some value eo
xr=x0                          % initialize the root with some value x0
iter =0                        % initiliaze the iteration counter
ea = 999                       % initiliaze absolute relative error

% Begin iteration loop

while ea < es & iter < itermax     % test to see if relative error is less then stopping criteria
   xrold = xr                      % update old value of root
   xr=xrold-f(xrold)/f'(xrold)     % calculate new root (How do you implement this line?)
   iter=iter+1                     % update the iteration counter
   if xr ~= 0                      % update absolute relative error
      ea=abs((xr-xrold)/xr)*100
   end
end

root=xr                        % final Root
```

Pseudocode for Newton-Raphson technique

```
es=eo                          % initialize error stopping criteria with some value eo
xr=x0                          % initialize the root with some value x0
iter =0                        % initiliaze the iteration counter
ea = 999                       % initiliaze absolute relative error

% Begin iteration loop

while ea < es & iter < itermax     % test to see if relative error is less then stopping criteria
   xrold = xr                      % update old value of root
   xr=xrold-f(xrold)/f'(xrold)     % calculate new root (How do you implement this line?)
   iter=iter+1                     % update the iteration counter
   if xr ~= 0                      % update absolute relative error
      ea=abs((xr-xrold)/xr)*100
   end
end

root=xr                        % final Root
```

How would you implement the Secant method?

Plug x_r back into the original equation to check solution

(B) Secant Method

Newton-Raphson Matlab Example

Systems of Non-Linear Equations

- We would like Roots of N-equations:

$$f_1(x_1, x_2, x_3, \ldots, x_n) = 0$$
$$f_2(x_1, x_2, x_3, \ldots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, x_2, x_3, \ldots, x_n) = 0$$

- We would like the x_i's that result in all equations simultaneously = 0.
- Non-linear Example:

$$x^2 y + y = 0 \longrightarrow d(x, y) = 0$$
$$y\sin(x) + xy^3 = 0 \longrightarrow e(x, y) = 0$$

Systems of Non-Linear Equations

- See Graphical Explanation using Matlab figure

Systems of Non-Linear Equations

- Roots correspond to $d_{i+1} = e_{i+1} = 0$

$$\frac{\partial d_i}{\partial x} x_{i+1} + \frac{\partial d_i}{\partial y} y_{i+1} = \frac{\partial d_i}{\partial x} x_i + \frac{\partial d_i}{\partial y} y_i - d_i$$
$$\frac{\partial e_i}{\partial x} x_{i+1} + \frac{\partial e_i}{\partial y} y_{i+1} = \frac{\partial e_i}{\partial x} x_i + \frac{\partial e_i}{\partial y} y_i - e_i$$

- 2 Equations & 2 unknowns (solve using Cramer's Rule), Our root at the next iteration is:

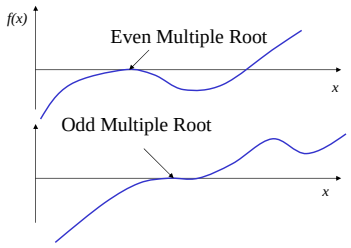
$$x_{i+1} = x_i - \frac{d_i \frac{\partial e_i}{\partial y} - e_i \frac{\partial d_i}{\partial y}}{\frac{\partial d_i}{\partial x}\frac{\partial e_i}{\partial y} - \frac{\partial d_i}{\partial y}\frac{\partial e_i}{\partial x}}$$

$$y_{i+1} = y_i - \frac{e_i \frac{\partial d_i}{\partial x} - d_i \frac{\partial e_i}{\partial x}}{\frac{\partial d_i}{\partial x}\frac{\partial e_i}{\partial y} - \frac{\partial d_i}{\partial y}\frac{\partial e_i}{\partial x}}$$

see Ex6.11 in text

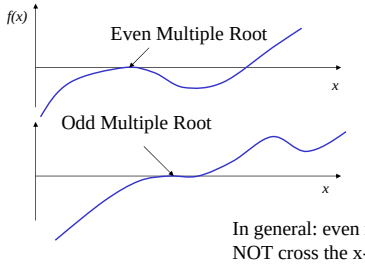
Multiple Root Problems

- Occurs when a function is tangent to the x-axis



Multiple Root Problems

- Occurs when a function is tangent to the x-axis



In general: even roots DO NOT cross the x-axis, but odd roots DO

Difficulties with Numerical Methods and Multiple Root Problems

1. Function does not change signs at an even multiple root (Bisection Problem)
2. Both $f(x)$ and $f'(x)$ are zero (although $f \rightarrow 0$ before f')
3. Newton-Raphson Method becomes linearly convergent for multiple root cases.

Newton Raphson Multiple Root Modification

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

- Define a new function

$$u(x) = \frac{f(x)}{f'(x)}$$

- Differentiate:

$$u'(x) = \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2}$$

- The function *u(x)* has the same roots as the original function *f(x) (See multiroot Matlab Example).*

Newton Raphson Multiple Root Modification

- Replace *f(x)* and *f'(x)* in the original NR formulation with *u(x)* and *u'(x)*

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \longrightarrow x_{i+1} = x_i - \frac{u(x_i)}{u'(x_i)}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \frac{[f'(x_i)]^2}{[f'(x_i)]^2 - f(x_i)f''(x_i)}$$

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)}}$$

- Preferred for multiple root problems
- Less Efficient, takes more iterations than the standard NR method