4. Roots of equations: open methods

- fixed point iteration
- Newton-Raphson
- secant method
- modified Newton-Raphson
- MATLAB `fzero` function

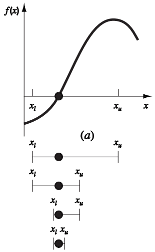
Open versus bracketing methods

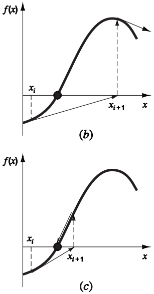
Bracketing methods

- start with $[x_l, x_u]$ where $f(x_l)f(x_u) < 0$ (root within interval)
- repeated halving/refinement $\Rightarrow$ always convergent (moves closer to the true root)

Open methods

- use formulas with one start value or two that need not bracket the root
- may *diverge* (move away from the true root)
- when they converge, they typically do so *faster* than bracketing methods





Fixed-point iteration

rearrange $f(x) = 0$ into *fixed-point* form

$$x = g(x)$$

- for example
  - algebraic manipulation: $x^2 - 2x + 3 = 0 \;\Rightarrow\; x = \dfrac{x^2 + 3}{2}$ or $x = \sqrt{2x - 3}$
  - add x to both sides: $\sin x = 0 \;\Rightarrow\; x = x + \sin x$

- given an initial guess x_0, iterate

  $$x_{i+1} = g(x_i), \quad i = 0, 1, \ldots$$

  called *fixed point iteration* (or *one-point iteration* or *successive substitution*)

- monitor the approximate relative error

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

Example

$$f(x) = e^{-x} - x$$

true root: $x^\star \approx 0.567143140453502$

separate directly and iterate $x_{i+1} = g(x_i) = e^{-x_i}$ with $x_0 = 0$

i	x_i	$x_{i+1} = g(x_i)$	ε_a	ε_t	$\varepsilon_{t,i}/\varepsilon_{t,i-1}$
0	0.0000	1.0000		76.32%	
1	1.0000	0.3679	171.83%	35.13%	46.03%
2	0.3679	0.6922	46.85%	22.05%	62.76%
3	0.6922	0.5005	38.31%	11.76%	53.31%
4	0.5005	0.6062	17.45%	6.89%	58.65%
5	0.6062	0.5454	11.16%	3.83%	55.62%
6	0.5454	0.5796	5.90%	2.20%	57.34%
7	0.5796	0.5601	3.48%	1.24%	56.36%
8	0.5601	0.5711	1.93%	0.71%	56.91%
9	0.5711	0.5649	1.11%	0.40%	56.60%
10	0.5649	0.5684	0.62%	0.23%	56.78%

error proportional to that in previous iteration (about 60%), called *linear convergence*

Example

choose alternative formulation $x = g(x) = -\ln(x)$ with starting point $x_0 = 0.5$

Iteration	x_i	$g(x_i)$	ε_a	ε_t	$\varepsilon_{t,i}/\varepsilon_{t,i-1}$
0	0.5000	0.6931		11.84%	
1	0.6931	0.3665	27.87%	22.22%	1.8766
2	0.3665	1.0037	89.12%	35.38%	1.5923
3	1.0037	–0.0037	63.48%	76.98%	2.1760

diverges in this case

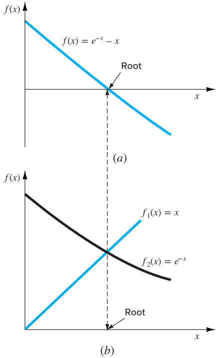
Two-curve graphical method

- split $f(x) = f_1(x) - f_2(x) = 0$ into $y_1 = f_1(x)$ and $y_2 = f_2(x)$
- the intersections give roots

Example: for $e^{-x} - x = 0$, take $y_1 = x$, $y_2 = e^{-x}$

x	$y_1 = x$	$y_2 = e^{-x}$
0.0	0.000	1.000
0.2	0.200	0.819
0.4	0.400	0.670
0.6	0.600	0.549
0.8	0.800	0.449
1.0	1.000	0.368

intersection near $x \approx 0.57$



Convergence via two-curve method

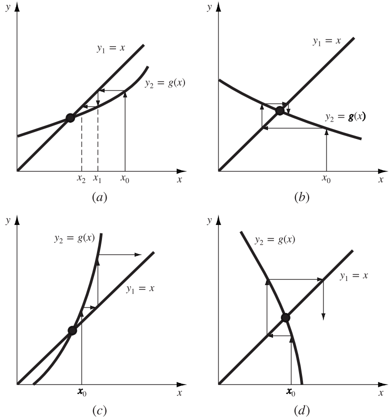
the two-curve method can be used to illustrate convergence and divergence

- equation $x = g(x)$ is re-expressed as a pair of equations: $y_1 = x,\ y_2 = g(x)$
- function $y_1 = x$ and $y_2 = g(x)$ are plotted
- roots of $f(x) = 0$ correspond to the abscissa at the intersection of the two curves

Iteration procedure

- start with initial guess x_0
- determine the corresponding point on y_2 curve: $(x_0, g(x_0))$
- move horizontally to y_1: the point (x_1, x_1)
- this corresponds to the first fixed-point iteration: $x_1 = g(x_0)$
- repeat starting from x_1 (moving to $(x_1, g(x_1))$ and then to (x_2, x_2))

Convergence via two-curve method



- (a) and (b): solution converges — estimates move closer to the root
- (c) and (d): iterations diverge away from the root

Convergence

fixed-point iteration converges locally (in a region of interest) if

$$|g'(x)| < 1$$

meaning the slope of $y_2 = g(x)$ is less than the slope of $y_1 = x$

- $0 < g'(x^\star) < 1$: monotone convergence
- $-1 < g'(x^\star) < 0$: oscillatory (spiral) convergence
- $|g'(x^\star)| > 1$: divergence

Proof: mean-value theorem with $x_{i+1} = g(x_i)$, $x^\star = g(x^\star)$:

$$x^\star - x_{i+1} \;=\; g(x^\star) - g(x_i) \;=\; g'(\xi)\,(x^\star - x_i)$$

thus for the true error $E_{t,i} = x^\star - x_i$,

$$E_{t,i+1} = g'(\xi)\,E_{t,i} \quad \Longrightarrow \quad |g'(\xi)| < 1 \;\Longrightarrow\; |E_{t,i+1}| < |E_{t,i}|$$

if converges, error decreases proportional to each step (linear convergence)

MATLAB implementation of our example

```
clear,clc,format compact
g=@(x) exp(-x);
x0 = 0;
[xr,ea,iter] = fixpt(g,x0,1e-6);
X = ['The root = ',num2str(xr),' (ea = ',num2str(ea),'% in ',...
num2str(iter),' iterations)'];
disp(X)
```

with function

```
function [x1,ea,iter] = fixpt(g,x0,es,maxit)
% fixpt: fixed point root locator
if nargin < 2, error('at least 2 arguments required'), end
if nargin < 3||isempty(es),es = 1e-6;end % if es is blank set to 1e-6
if nargin < 4||isempty(maxit),maxit = 50;end % if maxit is blank set to 50
iter = 0; ea = 100;
while (1)
x1 = g(x0);
iter = iter + 1;
if x1 ~= 0, ea = abs((x1 - x0)/x1)*100; end
if (ea <= es || iter >= maxit), break, end
x0 = x1;
end
end
```

running code gives

```
The root = 0.56714 (ea = 7.6775e-07% in 35 iterations)
```

Outline

- fixed point iteration
- **Newton-Raphson**
- secant method
- modified Newton-Raphson
- MATLAB `fzero` function

Newton-Raphson method

- first-order Taylor approximation around x_i

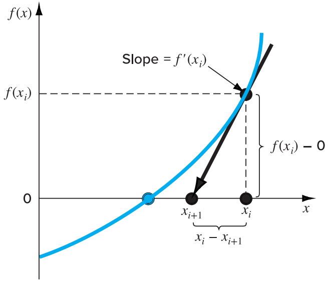
$$f(x) \approx \hat{f}(x) = f(x_i) + f'(x_i)\,(x - x_i)$$

- set to zero $\hat{f}(x) = 0$:

$$0 = f(x_i) + f'(x_i)\,(x - x_i)$$

- solution is our next estimate:

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



called *Newton-Raphson method* or just *Newton method*

Example

use the Newton-Raphson method to estimate the root of

$$f(x) = e^{-x} - x$$

starting with an initial guess of $x_0 = 0$ (true root: $x^\star \approx 0.56714329$)

- first derivative:
$$f'(x) = -e^{-x} - 1$$

- iterative equation:
$$x_{i+1} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1}$$

Example

first iteration:

$$x_1 = x_0 - \frac{e^{-x_0} - x_0}{-e^{-x_0} - 1} = 0 - \frac{e^{-0} - 0}{-e^{-0} - 1} = 0.5$$

repeating gives the results

i	x_i	$\varepsilon_t(\%)$	E_t
0	0.000000000	100.0	0.567143290
1	0.500000000	11.8	0.067143290
2	0.566311003	0.147	0.000832287
3	0.567143165	0.0000221	0.000000125
4	0.567143290	$< 10^{-8}$	0.000000000

converges much faster than fixed-point iteration

Error analysis of Newton-Raphson

second-order Taylor approximation:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{f''(\xi)}{2!}(x - x_i)^2$$

setting $x = x^\star$ and using $f(x^\star) = 0$ gives

$$0 = f(x_i) + f'(x_i)(x^\star - x_i) + \frac{f''(\xi)}{2!}(x^\star - x_i)^2$$

subtracting from Newton update $0 = f(x_i) + f'(x_i)(x_{i+1} - x_i)$ gives

$$0 = f'(x_i)(x^\star - x_{i+1}) + \frac{f''(\xi)}{2!}(x^\star - x_i)^2$$

define error $E_{t,i+1} = x^\star - x_{i+1}$ and assume convergence ($x_i, \xi \to x^\star$):, we get

$$E_{t,i+1} \approx -\frac{f''(x^\star)}{2f'(x^\star)}\, E_{t,i}^2$$

- error decreases quadratically (called *quadratic convergence*)
- number of correct digits roughly doubles with each iteration

Example

examine error of Newton-Raphson for $f(x) = e^{-x} - x$, at $x^\star = 0.56714329$

we have

$$f'(x^\star) = -1.56714329, \quad f''(x^\star) = 0.56714329$$

- error factor: $E_{t,i+1} \approx -\frac{f''(x^\star)}{2f'(x^\star)} E_{t,i}^2 = 0.18095 E_{t,i}^2$
- initial error with $x_0 = 0$: $E_{t,0} = 0.56714329$
- predicted errors using $E_{t,i+1} \approx 0.18095\, E_{t,i}^2$:

$$E_{t,1} \approx 0.0582 \quad \text{(true: } 0.0671\text{)}$$
$$E_{t,2} \approx 8.16 \times 10^{-4} \quad \text{(true: } 8.32 \times 10^{-4}\text{)}$$
$$E_{t,3} \approx 1.25 \times 10^{-7}$$
$$E_{t,4} \approx 2.84 \times 10^{-15}$$

- confirms quadratic convergence: error shrinks $\propto E_{t,i}^2$

Example

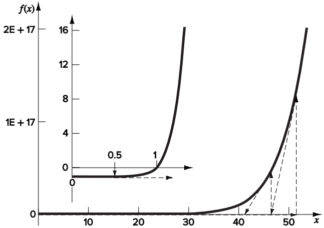
determine the positive root of $f(x) = x^{10} - 1$ using Newton-Raphson with $x_0 = 0.5$

Newton-Raphson update

$$x_{i+1} = x_i - \frac{x_i^{10} - 1}{10x_i^9}$$

Iterations

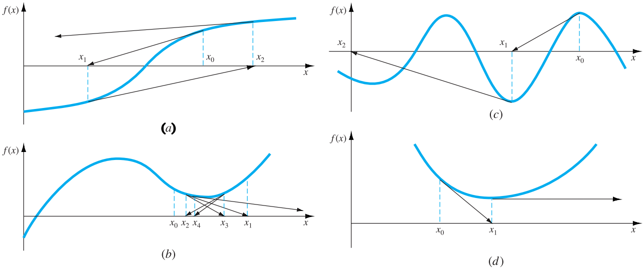
i	x_i	$\lvert\varepsilon_a\rvert$ (%)
0	0.5	
1	51.65	99.032
2	46.485	11.111
3	41.8365	11.111
4	37.65285	11.111
⋮	⋮	⋮
40	1.002316	2.130
41	1.000024	0.229
42	1	0.002



- very slow convergence
- first guess is in a region where the slope is near zero (flings the solution far away from the initial guess)

Pitfalls of Newton-Raphson

- can be slow and may diverge
- *inflection points:* if $f''(x) = 0$ near a root, iterations may diverge (figure a)
- *oscillations:* iterations can bounce around local maxima/minima (figure b)
- *near-zero slopes:* lead to very large jumps, possibly to other roots (figure c)
- *division by zero:* if $f'(x) = 0$, iteration fails completely (figure d)



Convergence insights

- no universal convergence guarantee
- success depends on:
  - nature of the function
  - quality of the initial guess
- remedies:
  - choose initial guesses close to the root
  - use graphical insight or physical intuition
  - use small stepsize $\alpha > 0$:

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

  - design software to detect slow convergence or divergence

MATLAB implementation of Newton-Raphson

```
function [root,ea,iter] = newtraph(func,dfunc,xr,es,maxit,varargin)
% newtraph: Newton - Raphson root location zeroes
% input:
% func = name of function
% dfunc = name of derivative of function
% xr = initial guess
% es = desired relative error (default = 0.0001%)
% maxit = maximum allowable iterations (default = 50)
% p1,p2,... = additional parameters used by function
% output:
% root = real root
% ea = approximate relative error (%)
% iter = number of iterations
if nargin<3,error('at least 3 input arguments required'),end
if nargin<4 || isempty(es),es=0.0001;end
if nargin<5 || isempty(maxit),maxit = 50;end
iter = 0;
while (1)
xrold = xr;
xr = xr - func(xr)/dfunc(xr);
iter = iter + 1;
if xr ~ = 0, ea = abs((xr - xrold)/xr) * 100; end
if ea <= es || iter >= maxit, break, end
end
root = xr;
```

Example

find mass of bungee jumper with a drag coefficient 0.25 kg/m to have a velocity

$$v(t) = \sqrt{\frac{gm}{c_d}}\ \tanh\left(\sqrt{\frac{gc_d}{m}}\, t\right)$$

of 36 m/s after 4 s of free fall

the function to be evaluated is

$$f(m) = \sqrt{\frac{gm}{c_d}}\ \tanh\left(\sqrt{\frac{gc_d}{m}}\, t\right) - 36$$

the derivative of this function must be evaluated with respect to the unknown m:

$$\frac{df(m)}{dm} = \frac{1}{2}\sqrt{\frac{g}{mc_d}}\ \tanh\left(\sqrt{\frac{gc_d}{m}}\, t\right) - \frac{g}{2m}\, t\ \operatorname{sech}^2\left(\sqrt{\frac{gc_d}{m}}\, t\right)$$

code:

```
>> y = @(m) sqrt(9.81*m/0.25)*tanh(sqrt(9.81*0.25/m)*4)-36;
>> dy = @(m) 1/2*sqrt(9.81/(m*0.25))*tanh((9.81*0.25/m) ...
^(1/2)*4)-9.81/(2*m)*sech(sqrt(9.81*0.25/m)*4)^2;
>> newtraph(y,dy,140,0.00001)
ans =
142.7376
```

Outline

- fixed point iteration
- Newton-Raphson
- **secant method**
- modified Newton-Raphson
- MATLAB `fzero` function

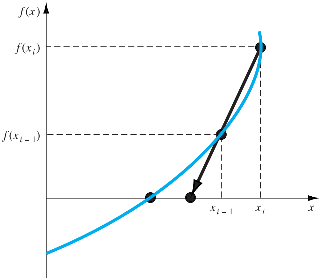
Secant method

- some functions have derivatives that are difficult or inconvenient to evaluate
- the derivative can be approximated by a backward finite divided difference

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

- this approximation can be substituted into to yield the **secant method:**

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



requires two initial estimates

Example

use the secant method on $f(x) = e^{-x} - x$ with starting points $x_{-1} = 0$, $x_0 = 1.0$

true root ≈ 0.56714329

First iteration

$$x_{-1} = 0 \quad f(x_{-1}) = 1.00000$$

$$x_0 = 1 \quad f(x_0) = -0.63212$$

so

$$x_1 = 1 - \frac{-0.63212(0-1)}{1-(-0.63212)} = 0.61270 \quad \varepsilon_t = 8.0\%$$

Example

Second iteration

$$x_0 = 1 \quad f(x_0) = -0.63212$$

$$x_1 = 0.61270 \quad f(x_1) = -0.07081$$

note both estimates are now on the same side of the root; we have

$$x_2 = 0.61270 - \frac{-0.07081(1 - 0.61270)}{-0.63212 - (-0.07081)} = 0.56384 \quad \varepsilon_t = 0.58\%$$

Third iteration

$$x_1 = 0.61270 \quad f(x_1) = -0.07081$$

$$x_2 = 0.56384 \quad f(x_2) = 0.00518$$

so

$$x_3 = 0.56384 - \frac{0.00518(0.61270 - 0.56384)}{-0.07081 - (-0.00518)} = 0.56717 \quad \varepsilon_t = 0.0048\%$$

Secant method versus false-position method

false position

$$x_r = x_u - \frac{f(x_u)\,(x_l - x_u)}{f(x_l) - f(x_u)}$$

secant

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

- identical first iterate if $x_u = x_0$ and $x_l = x_{-1}$
- difference
  - false-position: always brackets the root by replacing the value with same sign as $f(x_r)$
  - secant: updates sequentially (x_{i+1} replaces x_i and x_i replaces x_{i-1})
- consequence
  - false-position: guaranteed convergence (root stays in bracket)
  - secant: may converge faster but can also diverge

Example

use false-position and secant methods to estimate the root of $f(x) = \ln x$

initial guesses: $x_l = x_{-1} = 0.5$, $x_u = x_0 = 5.0$

False-position iterations

iteration	x_l	x_u	x_r
1	0.5	5.0	1.8546
2	0.5	1.8546	1.2163
3	0.5	1.2163	1.0585

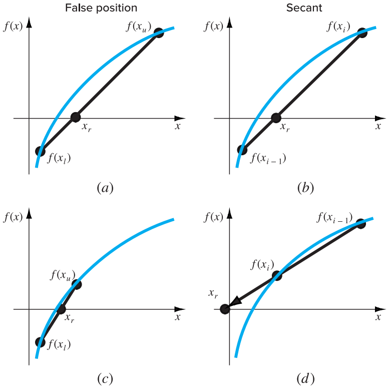
converges to true root $x = 1$

Secant iterations

iteration	x_{i-1}	x_i	x_{i+1}
1	0.5	5.0	1.8546
2	5.0	1.8546	−0.10438

result: divergence

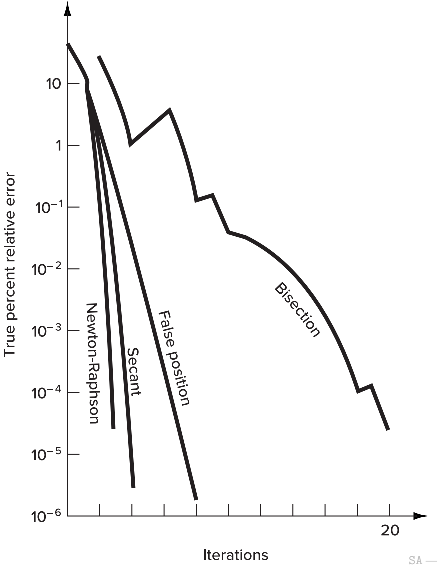
Example



- for secant both values lie on same side of the root
- false-position is robust but slower because one endpoint stays fixed
- secant usually converges faster when it works

Example: numerical comparison

comparison for $f(x) = e^{-x} - x$



Modified secant method

to avoid evaluating derivatives and two guesses

approximate derivative with fractional perturbation δ:

$$f'(x_i) \approx \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

iterative formula

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

called *modified secant method*

- δ must be chosen carefully
  - too small $\longrightarrow$ round-off errors due to subtractive cancellation
  - too large $\longrightarrow$ inefficiency or divergence
- useful when derivatives are difficult or inconvenient

Example: modified secant

use modified secant method to solve $f(x) = e^{-x} - x$, $\delta = 0.01$, $x_0 = 1.0$

- *first iteration*:

$$x_0 = 1, \quad f(x_0) = -0.63212$$
$$x_0 + \delta x_0 = 1.01, \quad f(x_0 + \delta x_0) = -0.64578$$
$$x_1 = 1 - \frac{0.01(-0.63212)}{-0.64578 - (-0.63212)} = 0.537263, \quad |\varepsilon_t| = 5.3\%$$

- *second iteration*:

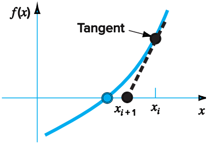
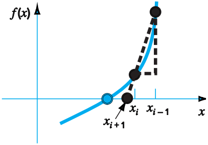
$$x_1 = 0.537263, \quad f(x_1) = 0.047083$$
$$x_1 + \delta x_1 = 0.542635, \quad f(x_1 + \delta x_1) = 0.038579$$
$$x_2 = 0.537263 - \frac{0.005373(0.047083)}{0.038579 - 0.047083} = 0.56701, \quad |\varepsilon_t| = 0.0236\%$$

- *third iteration*:

$$x_2 = 0.56701, \quad f(x_2) = 0.000209$$
$$x_2 + \delta x_2 = 0.572680, \quad f(x_2 + \delta x_2) = -0.00867$$
$$x_3 = 0.56701 - \frac{0.00567(0.000209)}{-0.00867 - 0.000209} = 0.567143, \quad |\varepsilon_t| = 2.365 \times 10^{-5}\%$$

- method converges rapidly to true root 0.56714329

Summary

Method	Formulation	Graphical Interpretation	Errors and Stopping Criteria
Newton-Raphson	$x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$		Stopping criterion: $\left\lvert \dfrac{x_{i+1} - x_i}{x_{i+1}} \right\rvert 100\% \leq \epsilon_s$ Error: $E_{i+1} = O(E_i^2)$
Secant	$x_{i+1} = x_i - \dfrac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$		Stopping criterion: $\left\lvert \dfrac{x_{i+1} - x_i}{x_{i+1}} \right\rvert 100\% \leq \epsilon_s$

Outline

- fixed point iteration
- Newton-Raphson
- secant method
- **modified Newton-Raphson**
- MATLAB `fzero` function

Multiple roots

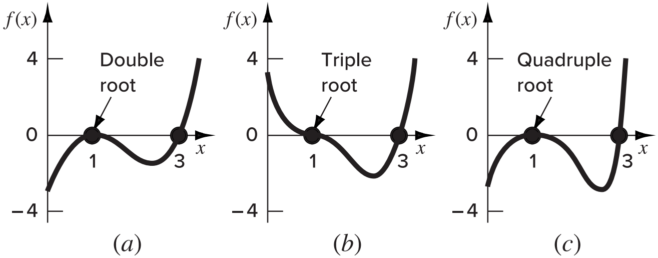
a **multiple root** occurs when a function is tangent to the x-axis at the root

- example: double root (touches the axis at $x = 1$ without crossing)

$$f(x) = (x-3)(x-1)(x-1) = x^3 - 5x^2 + 7x - 3$$

- example: triple root (tangent and crosses the axis at $x = 1$)

$$f(x) = (x-3)(x-1)^3 = x^4 - 6x^3 + 12x^2 - 10x + 3$$



- **rule of thumb**: even multiplicities touch but do not cross; odd multiplicities cross

Why multiple roots are tricky?

1. **bracketing methods fail for even multiplicities:** f does not change sign, so bracketing tests may not detect the root
2. **derivative vanishes at the root:** $f'(x^\star) = 0$ as well as $f(x^\star) = 0$
   - Newton and secant divide by f' $\Rightarrow$ possible division by values near zero
   - practical safeguard: check $|f(x_i)|$ first and terminate when below tolerance (since f reaches 0 before f')
3. **slower convergence:** standard Newton and secant become *linearly* convergent for multiple roots

Modified Newton method

Modified Newton method for known multiplicity

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

- m is root multiplicity
- modify Newton to restore quadratic convergence
- requires prior knowledge of m

Modified Newton method: applies Newton to $u(x) = \frac{f(x)}{f'(x)}$

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

- $u(x) = \frac{f(x)}{f'(x)}$ has same roots as $f(x)$
- regains fast (quadratic) convergence at multiple roots

Example

use both the standard and modified Newton-Raphson to find the multiple root

$$f(x) = (x-3)(x-1)^2 = x^3 - 5x^2 + 7x - 3$$

- starting from $x_0 = 0$
- we have

$$f'(x) = 3x^2 - 10x + 7, \qquad f''(x) = 6x - 10$$

(standard) Newton

$$x_{i+1} = x_i - \frac{x_i^3 - 5x_i^2 + 7x_i - 3}{3x_i^2 - 10x_i + 7}$$

modified Newton

$$x_{i+1} = x_i - \frac{(x_i^3 - 5x_i^2 + 7x_i - 3)(3x_i^2 - 10x_i + 7)}{(3x_i^2 - 10x_i + 7)^2 - (x_i^3 - 5x_i^2 + 7x_i - 3)(6x_i - 10)}$$

Example: results

(standard) Newton

i	x_i	$\lvert\varepsilon_t\rvert$ (%)
0	0.0000000	100
1	0.4285714	57
2	0.6857143	31
3	0.8328654	17
4	0.9133290	8.7
5	0.9557833	4.4
6	0.9776551	2.2

behavior: steady but linear approach to $x^\star = 1$

modified Newton

i	x_i	$\lvert\varepsilon_t\rvert$ (%)
0	0.000000	100
1	1.105263	11
2	1.003082	0.31
3	1.000002	2.4×10^{-4}

behavior: rapid (quadratic) approach to $x^\star = 1$

Example

to estimate root $x = 3$, we start at $x_0 = 4$

standard Newton

i	x_i	$\lvert\varepsilon_t\rvert$ (%)
0	4.000000	33
1	3.400000	13
2	3.100000	3.3
3	3.008696	0.29
4	3.000075	2.5×10^{-3}
5	3.000000	2×10^{-7}

modified Newton

i	x_i	$\lvert\varepsilon_t\rvert$ (%)
0	4.000000	33
1	2.636364	12
2	2.820225	6.0
3	2.961728	1.3
4	2.998479	0.051
5	2.999998	7.7×10^{-5}

- for a simple root, standard Newton is slightly more efficient
- and requires fewer computation (no f'')

Modified secant method for multiple roots

- modified version of the secant method suited for multiple roots can also be developed
- apply secant method on $u(x) = f(x)/f'(x)$
- the resulting formula is:

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

Outline

- fixed point iteration
- Newton-Raphson
- secant method
- modified Newton-Raphson
- **MATLAB `fzero` function**

MATLAB `fzero` function

`fzero` finds the real root of a single equation

Basic syntax:

`fzero(function, x0)` or `fzero(function, [x0 x1])`

- `function`: function handle
- `x0`: initial guess or `x0, x1`: must bracket a root

Example: $x^2 - 9 = 0$

- negative root:

```
x = fzero(@(x) x^2-9,-4)
```

- positive root:

```
x = fzero(@(x) x^2-9,4)
```

- using bracketing:

```
x = fzero(@(x) x^2-9,[0 4])
```

Note: if no sign change occurs in $[x_0, x_1]$, MATLAB returns an error

More on MATLAB `fzero`

Extended syntax:

```
[x,fx] = fzero(function,x0,options,p1,p2,...)
```

- `x` : computed root
- `fx` : function value at the root
- `options` : structure created using `optimset`
- `p1,p2,...` : extra parameters required by the function
- if parameters are passed but options are not used: pass `[]` in its place

`optimset` **syntax:**

```
options = optimset('par1',val1,'par2',val2,...)
```

- we can list all available parameters via typing `optimset`
- commonly used with `fzero`:
  - `display = 'iter'` : shows iteration details
  - `tolx` : termination tolerance for x

Example

display result for finding root of $f(x) = x^{10} - 1$

```
>> options = optimset('display','iter');
>> [x,fx] = fzero(@(x) x^10 - 1,0.5,options)

Func-count    x           f(x)        Procedure
1             0.5         -0.999023   initial
2             0.485858    -0.999267   search
3             0.514142    -0.998709   search
4             0.48        -0.999351   search
5             0.52        -0.998554   search
6             0.471716    -0.999454   search
.
.
Zero found in the interval: [-0.14, 1.14].
x =
1
fx =
0
```

set tolerance

```
>> options = optimset ('tolx', 1e-3);
>> [x,fx] = fzero(@(x) x^10 - 1,0.5,options)
x =
1.0009
fx =
0.0090
```

References and further readings

- S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers* (8th edition). McGraw Hill, 2021. (Ch.6)

- S. C. Chapra. *Applied Numerical Methods with MATLAB for Engineers and Scientists* (5th edition). McGraw Hill, 2023. (Ch.6)