BRACKETING METHODS: BACKGROUND [3-3]

— interval \([a, b]\) and \(f(x)\) are such that \(f(a) \cdot f(b) < 0\), i.e., \(f(a), f(b)\) have opposite signs.

— pick \(x_k \in [a, b]\) according to some “rule.”

\[\Rightarrow\] if \(f(x_k)\) has the same sign as \(f(a)\), then consider new, reduced interval to be \([x_k, b]\) and repeat iterative step, picking a new \(x_{k+1}\).

\[\Rightarrow\] if \(f(x_k)\) has the same sign as \(f(b)\), then consider new, reduced interval to be \([a, x_k]\) and repeat iterative step, picking a new \(x_{k+1}\).
We keep “halving” the interval, closing in on the root from both sides, until $f(x_k) \approx 0$. 
BRACKETING METHOD: ALGORITHM [3-4]

// initialize
// define function f here
    cin >> a >> b >> maxiter >> tol ;
    iter = 1;
// iterative loop
    do
        x = // special rule for particular method ;
        if ( f(x)*f(a) > 0 ) then
            a = x;
        else
            b = x;
        cout << iter << x << endln;
    until( (f(x) < tol) || (iter >= maxiter));

WAYS OF CHOOSING $x_k$ [3-4A]

Two major (easy) options: (1) middle of interval, or (2) root of linear approximation to curve.
BISECTION METHOD (BRACKETING STRATEGY) [3-5]  
(“BIS,” INTERNAL HALVING, BOLZANO’S)  

The special rule to choose $x \in [a, b]$ is “pick midpoint.”

Therefore, $x_k = x_{\text{midpoint}} = \frac{a+b}{2}$.

Original $a$ is replaced by $x_1$, then original $b$ is replaced by $x_2$, then new $a = x_1$ is replaced by $x_3$, etc., closing in on the root.
REGULA FALSI METHOD (BRACKETING STRATEGY) [3-6]
(FALSE POSITION)

Instead of choosing the new \( x_k \) as the midpoint of the interval, we try to get \( x_k \) closer to the actual root by approximating the curve by a straight line drawn from points on the curve corresponding to the endpoints of the current interval.

\[
(a, f(a)) \quad \text{root} \quad (b, f(b))
\]
The slope of the straight line can be computed two ways, from 
\((a, f(a))\) to \((x_k, 0)\) and from \((a, f(a))\) to \((b, f(b))\). Equating these two 
values we get:

\[
\text{slope of line } = m = \frac{f(b) - f(a)}{b - a} = \frac{0 - f(a)}{x_k - a}
\]

\[
\text{THUS, } x_k - a = \frac{-f(a)(b - a)}{f(b) - f(a)}
\]

or \[
x_k = a - \frac{f(a)(b - a)}{f(b) - f(a)} = \frac{af(b) - af(a) - bf(a) + af(a)}{f(b) - f(a)}
\]

\[
= \frac{af(b) - bf(a)}{f(b) - f(a)}
\]

Letting \(L = f(a)\) ("left") and \(R = f(b)\) ("right"), we get

\[
x_k = \frac{aR - bL}{R - L} = \frac{bL - aR}{L - R}
\]
ALTERNATIVE STRATEGY — SLOPE METHOD:
ALGORITHM [3-7]

// initialization
// define f
    cin >> x0 >> maxit >> tol;
    iter = 1;
    xprev = x0;
// iterative loop
    do
        Deltax = // special rule for method
        x = xprev + Deltax;
        cout << iter << x << Deltax;
        xprev = x;
        iter = iter + 1;
    until ((Deltax < tol) || (iter >= maxiter));
The slope of the line tangent to $f(x)$ at $x = x_k$, i.e., at point $A$ on the curve, has slope equal to

$$m = \frac{0 - f(x_k)}{x_{k+1} - x_k} = \frac{\Delta y}{\Delta x} = \frac{y_B - y_A}{x_B - x_A}$$

Since this line is TANGENT to $f(x)$, the slope $m$ must be equal to $f'(x_k)$. 

BF-9 §2.3, p 68 Alg 2.3
Thus, we get

\[ f'(x_k) = \frac{-f(x_k)}{x_{k+1} - x_k} \]

or

\[ x_{k+1} - x_k = \frac{-f(x_k)}{f'(x_k)} \]

or

\[ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \]

Thus the \( \Delta x \) for the Newton-Raphson method is

\[ \Delta x = \frac{-f(x_k)}{f'(x_k)} \]

\[ \Rightarrow \text{Problem: need to know } f'(x). \]

*Matlab Demo – newtondriver*
SECANT METHOD (SLOPE STRATEGY) [3-9]

BACKGROUND

⇒ Substitute secant line in place of tangent.

Computing the slope of the secant line through $A$ and $B$, from $A$ to $B$ and from $B$ to $C$, we get

$$m_{AB} = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

$$m_{BC} = \frac{0 - f(x_k)}{x_{k+1} - x_k}$$
Equating $m_{AB}$ with $m_{BC}$, we get

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

Thus

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

and

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

Thus, for the Secant Method, $\Delta x = -\frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{x-1})}$
COMPARISON [3-10]

I WORK

**NEWTON:** requires $f(x)$ and $f'(x)$. But $f'(x)$ may be difficult (impossible) to obtain!

**SECANT:** requires $x_k, x_{k-1}$ at every step, even initially. Normally, you pick one initial guess and make a small perturbation for the second value.

II CONVERGENCE

**NEWTON:** **QUADRATIC** for simple roots; **LINEAR** for multiple roots.

**SECANT:** **ALMOST-QUADRATIC** ("super-linear," 1.618) for simple roots; **LINEAR** for multiple roots.
COMMENTS [3-11]

1. Although FP seems faster, one can find intervals and functions for which FP is slower than BIS.

2. Bracketing methods are linear in convergence. Thus, they are always slower than SEC or NR.

3. FP and BIS (bracketing strategies) will converge! SEC and NR (slope strategies) may not!

4. Some problems may need a combination of strategies and methods. I.e., run BIS for a few iterations (guaranteed convergence) to get “close” to a root, and then switch to SEC or NR for speed and accuracy.

5. NR can converge rapidly. Old assignment for CS/Math 61 involved finding $\sqrt{5}$ to at least 100 places. It took only 7 iterations!
BASIC ROOT-FINDING: SUMMARY [3-12]

1. All methods find only 1 root (at a time).
   We do **not** find ALL roots simultaneously.

2. All methods need to know (more or less) where the
   root-to-be-found lies (for initial guess).


4. RE: robustness – BIS, FP always converge. NR, SEC may not!

5. RE: ease of coding – if \( f(x) \) is complicated or not analytically
   known (e.g., derived only as input signal), \( f'(x) \) may be
   difficult/impossible to obtain and/or code. THUS, use SEC.

6. RE: ease of use – if writing code for others, SEC needs only to
   have access to \( f(x) \).
MULTIPLE ROOTS [3-13]  BF-9 §2.4, p 82

Def. A root $\alpha$ of $f(x)$ is said to be of MULTIPLICITY $p$ iff

$$f(x) = (x - \alpha)^p q(x).$$

Some methods (e.g., NR, SEC) slow down for multiple roots.

I – If we know a given root $\alpha$ is of multiplicity $p$, we can modify NR to:

$$x_{k+1} = x_k - p \frac{f(x_k)}{f'(x_k)}$$

to speed it up.
Another approach is to differentiate, repeatedly, the given function. **WHY?**

\[ f(x) = (x - \alpha)^p g(x) \]

Then root \( \alpha \) is of multiplicity \( p \).

Then \( f'(x) = (x - \alpha)^p g'(x) + g(x)p(x - \alpha)^{p-1} \) (via product rule)

\[ = (x - \alpha)^{p-1}[(x - \alpha)g'(x) + p \cdot g(x)] \]

\[ = (x - \alpha)^{p-1}h(x) \] (renaming the bracketed term)

**NOTE:** root \( \alpha \) is of multi. \( p - 1 \) for function \( f'(x) \).

Continuing the differentiation, we can show that \( \alpha \) is a simple root of \( f^{(p-1)}(x) \).

**THUS,** we can first differentiate \( f \) and then apply a root-finding scheme to the derived function.

. BF-9 alt p 84 top — define \( \mu(x) = \frac{f(x)}{f'(x)} \). If \( p \) is a root of \( f \) of multip. \( m \), then \( p \) is a simple root of \( \mu \).
III – Beware of inherent (machine) error (i.e., “fuzziness”)

a) A simple root may look like 3 roots!

b) A root of multiplicity two may look like two roots or no roots.