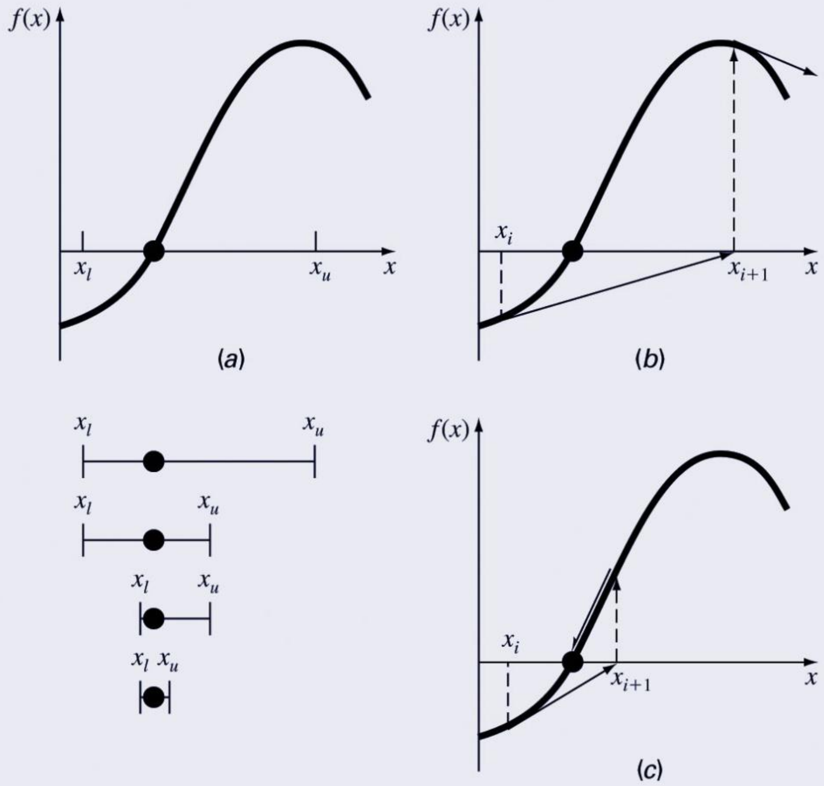
Chapter Objectives

- Recognizing the difference between bracketing and open methods for root location.
- Understanding the fixed-point iteration method and how you can evaluate its convergence characteristics.
- Knowing how to solve a roots problem with the Newton-Raphson method and appreciating the concept of quadratic convergence.
- Knowing how to implement both the secant and the modified secant methods.
- Knowing how to use MATLAB’s fzero function to estimate roots.
- Learning how to manipulate and determine the roots of polynomials with MATLAB.

Open Methods

- *Open methods* differ from bracketing methods, in that open methods require only a single starting value or two starting values that do not necessarily bracket a root.
- Open methods may diverge as the computation progresses, but when they do converge, they usually do so much faster than bracketing methods.

Graphical Comparison of Methods



a) Bracketing method
b) Diverging open method
c) Converging open method - note speed!

Simple Fixed-Point Iteration

- Rearrange the function $f(x)=0$ so that x is on the left-hand side of the equation: $x=g(x)$
- Use the new function g to predict a new value of x - that is, $x_{i+1}=g(x_i)$
- The approximate error is given by:

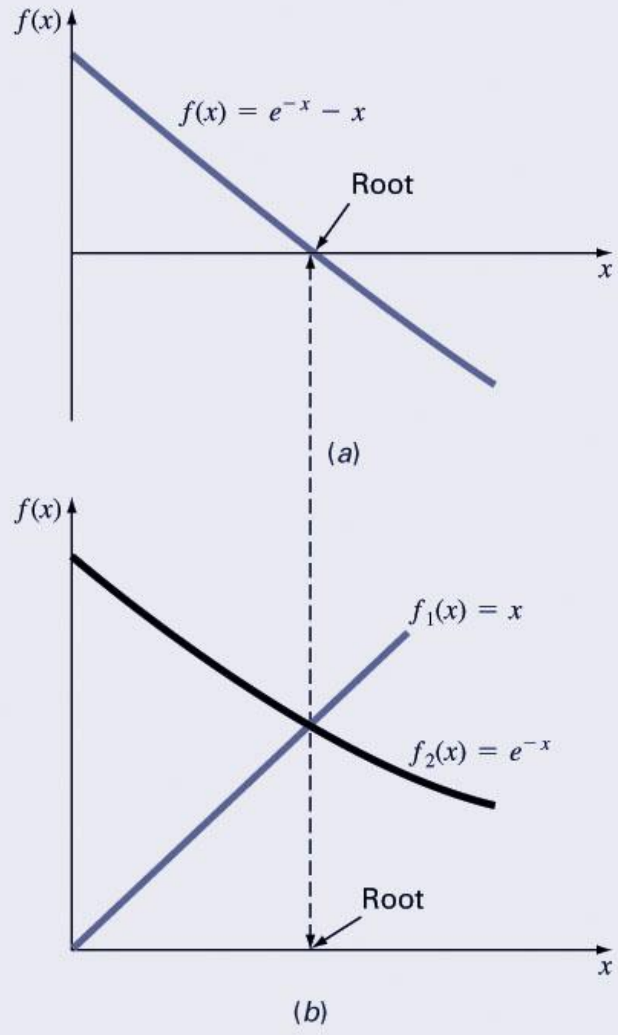
$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| 100\%$$

Example

- Solve $f(x)=e^{-x}-x$
- Re-write as $x=g(x)$ by isolating x (example: $x=e^{-x}$)
- Start with an initial guess (here, 0)

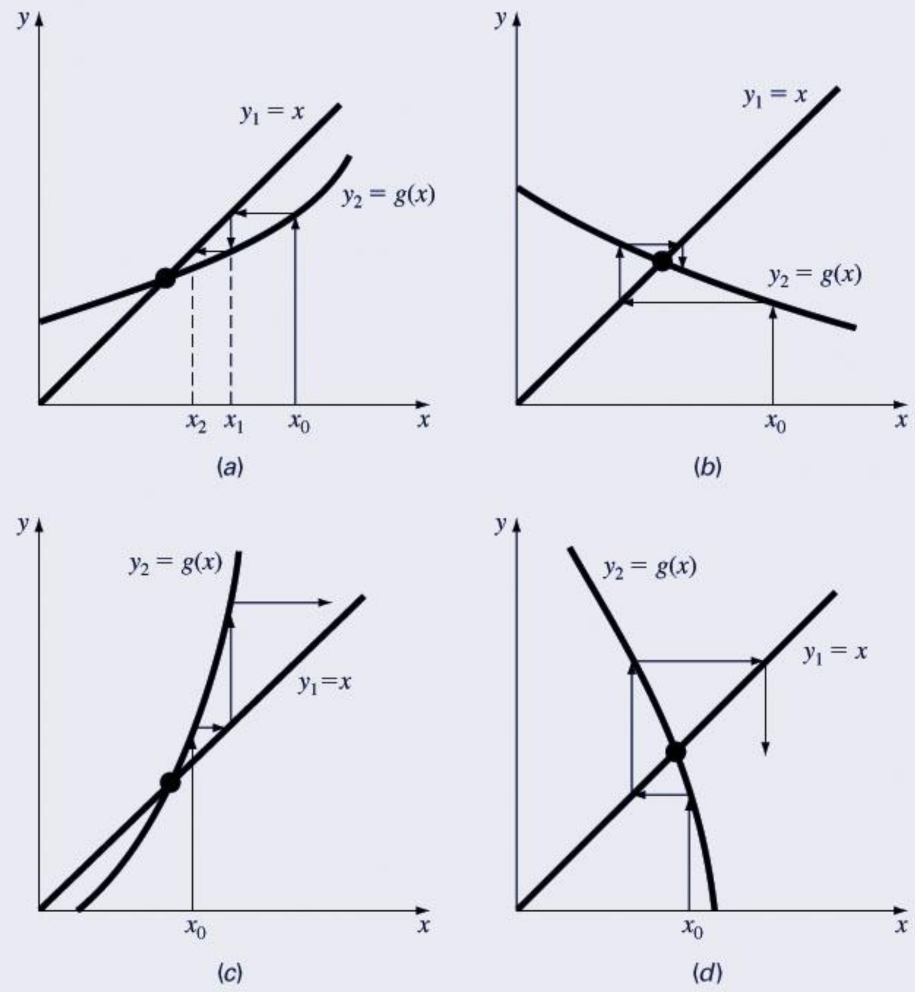
i	x_i	$\lvert\varepsilon_a\rvert$ %	$\lvert\varepsilon_t\rvert$ %	$\lvert\varepsilon_t\rvert_i/\lvert\varepsilon_t\rvert_{i-1}$
0	0.0000		100.000	
1	1.0000	100.000	76.322	0.763
2	0.3679	171.828	35.135	0.460
3	0.6922	46.854	22.050	0.628
4	0.5005	38.309	11.755	0.533

- Continue until some tolerance is reached



Convergence

- Convergence of the simple fixed-point iteration method requires that the derivative of g(x) near the root has a magnitude less than 1.
  a) Convergent, $0 \le g' < 1$
  b) Convergent, $-1 < g' \le 0$
  c) Divergent, $g' > 1$
  d) Divergent, $g' < -1$

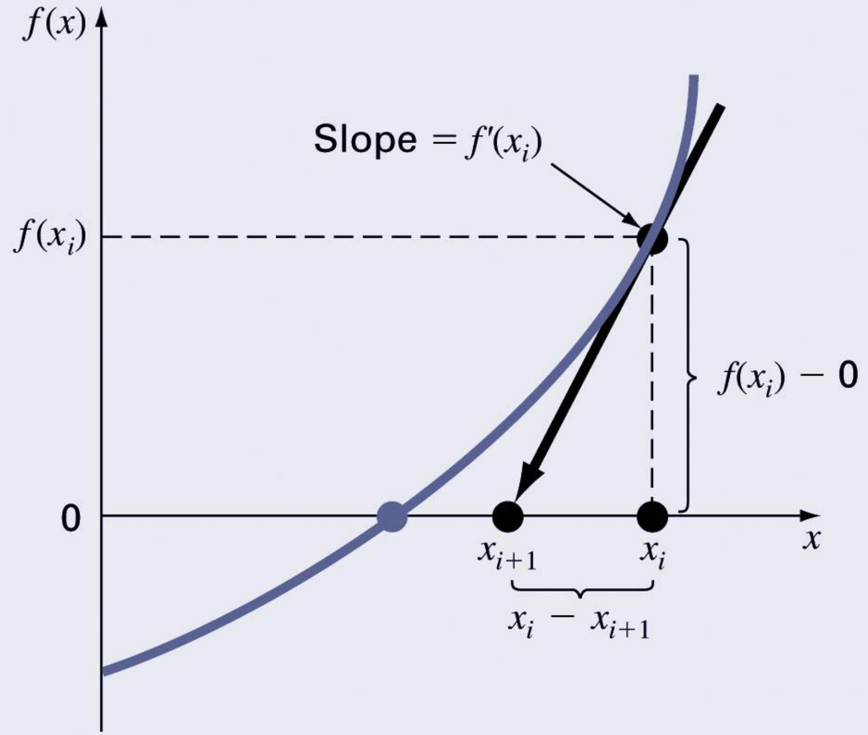


Newton-Raphson Method

- Based on forming the tangent line to the *f*(*x*) curve at some guess *x*, then following the tangent line to where it crosses the *x*-axis.

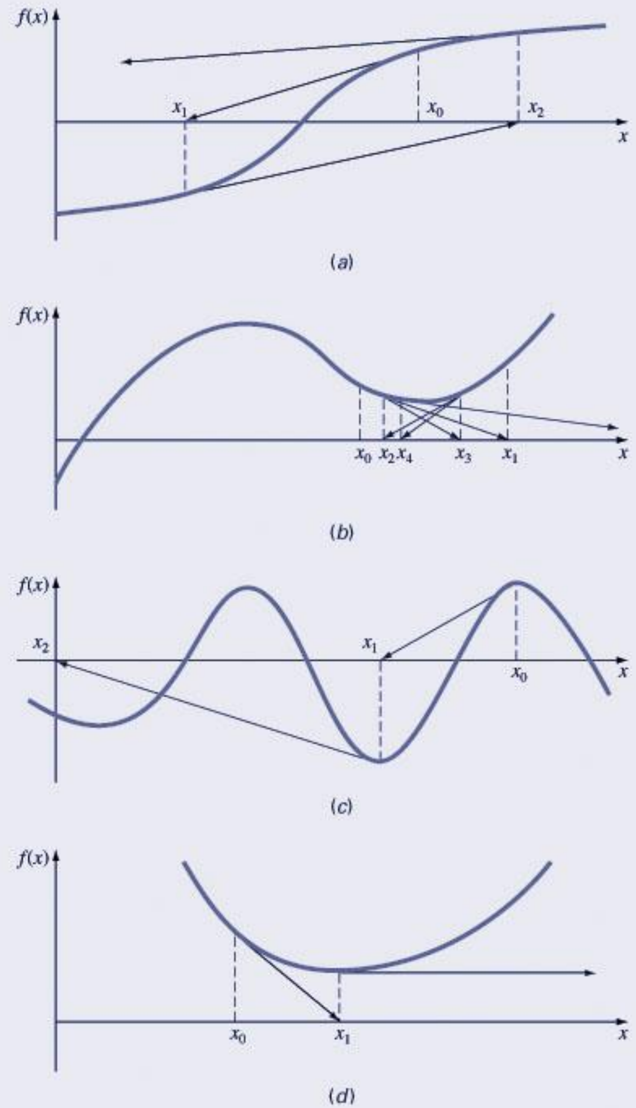
$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

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



Pros and Cons

- Pro: The error of the i+1th iteration is roughly proportional to the square of the error of the ith iteration - this is called *quadratic convergence*
- Con: Some functions show slow or poor convergence



Secant Methods

- A potential problem in implementing the Newton-Raphson method is the evaluation of the derivative - there are certain functions whose derivatives may be difficult or inconvenient to evaluate.
- For these cases, the derivative can be approximated by a backward finite divided difference:

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

Secant Methods (cont)

- Substitution of this approximation for the derivative to the Newton-Raphson method equation gives:

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

- Note - this method requires *two* initial estimates of *x* but does *not* require an analytical expression of the derivative.

MATLAB's fzero Function

- MATLAB's fzero provides the best qualities of both bracketing methods and open methods.
  - Using an initial guess:
    x = fzero(*function*, *x0*)
    [*x*, *fx*] = fzero(*function*, *x0*)
    - *function* is a function handle to the function being evaluated
    - *x0* is the initial guess
    - *x* is the location of the root
    - *fx* is the function evaluated at that root
  - Using an initial bracket:
    x = fzero(*function*, [*x0 x1*])
    [*x*, *fx*] = fzero(*function*, [*x0 x1*])
    - As above, except *x0* and *x1* are guesses that *must* bracket a sign change

fzero Options

- Options may be passed to fzero as a third input argument - the options are a data structure created by the optimset command
- *options* = optimset('par_1', val_1, 'par_2', val_2,…)
  - par_n is the name of the parameter to be set
  - val_n is the value to which to set that parameter
  - The parameters commonly used with fzero are:
    - display: when set to 'iter' displays a detailed record of all the iterations
    - tolx: A positive scalar that sets a termination tolerance on x.

fzero Example

- options = optimset(‘display’, ‘iter’);
  - Sets options to display each iteration of root finding process
- [x, fx] = fzero(@(x) x^10-1, 0.5, options)
  - Uses fzero to find roots of $f(x)=x^{10}-1$ starting with an initial guess of x=0.5.
- MATLAB reports x=1, fx=0 after 35 function counts

Polynomials

- MATLAB has a built in program called roots to determine all the roots of a polynomial - including imaginary and complex ones.
- x = roots(c)
  - x is a column vector containing the roots
  - c is a row vector containing the polynomial coefficients
- Example:
  - Find the roots of $f(x)=x^5-3.5x^4+2.75x^3+2.125x^2-3.875x+1.25$
  - x = roots([1 -3.5 2.75 2.125 -3.875 1.25])

Polynomials (cont)

- MATLAB's poly function can be used to determine polynomial coefficients if roots are given:
  - b = poly([0.5 -1])
    - Finds $f(x)$ where $f(x) = 0$ for $x=0.5$ and $x=-1$
    - MATLAB reports b = [1.000 0.5000 -0.5000]
    - This corresponds to $f(x)=x^2+0.5x-0.5$
- MATLAB's polyval function can evaluate a polynomial at one or more points:
  - a = [1 -3.5 2.75 2.125 -3.875 1.25];
    - If used as coefficients of a polynomial, this corresponds to $f(x)=x^5-3.5x^4+2.75x^3+2.125x^2-3.875x+1.25$
  - polyval(a, 1)
    - This calculates $f(1)$, which MATLAB reports as -0.2500