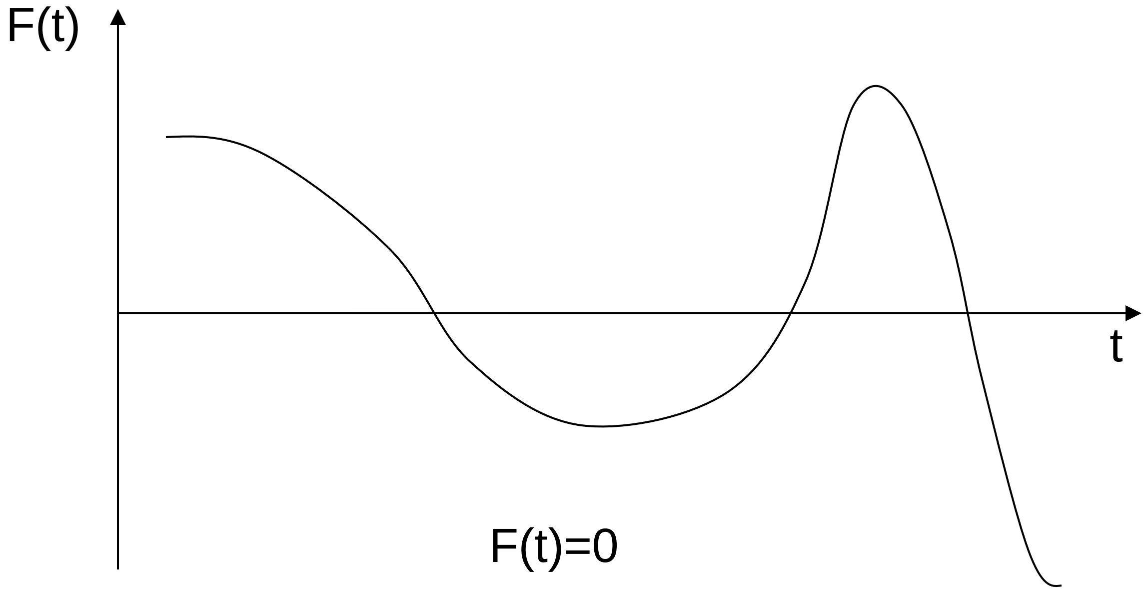
Numerical Analysis

- Root Finding I-

Hanyang University

Jong-Il Park

Root Finding



- When there is no analytic solution

  eg. $F(t) = e^{-t}(3\cos 2t + 4\sin 2t)t$

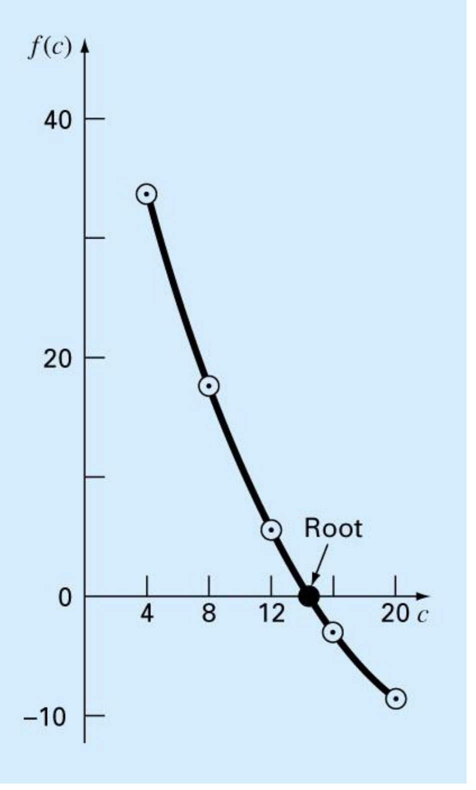
General procedure of root finding

Step 1. Searching rough initial solutions

- graphical method
- incremental search
- experience
- solution of a simplified model

Step 2. Finding an exact solution

- Bracketing method
- Open method

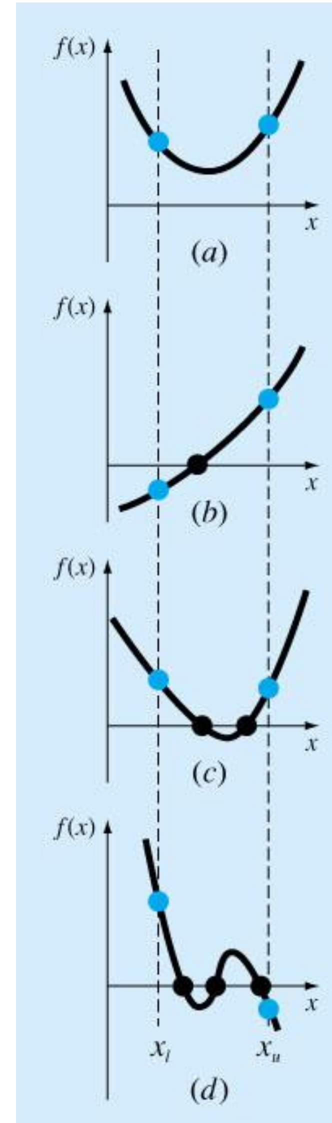


Graphical method

Bracketing method

- Assume: There must be a solution in a given interval [a,b]
- Key idea: Reducing the interval systematically
- Merit: Convergence to an exact solution

- Methods
  - bisection
  - linear interpolation



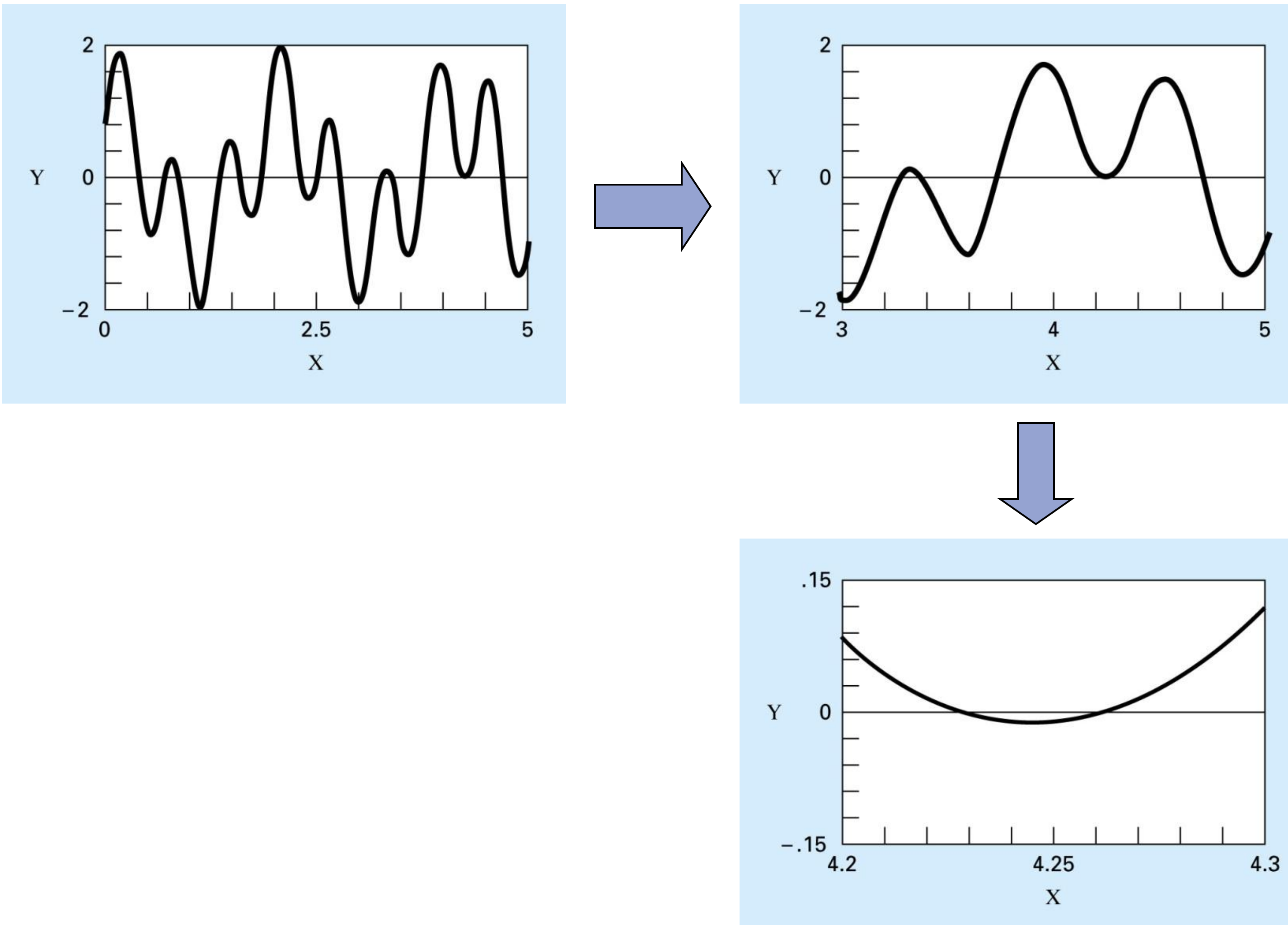
Open method

- Idea: Starting from an arbitrary point, find the root based on a regular iteration
- Risk: Divergence (depending on initial value)
- Merit: Faster convergence than bracketing methods

- Methods
  - fixed-point iteration
  - Newton-Raphson method
  - Secant method
  - Muller method

Initial step: Finding a rough approximation

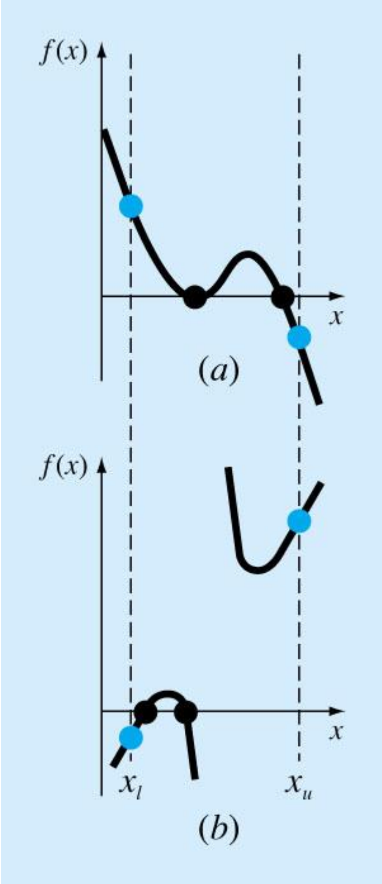
- Graphical method



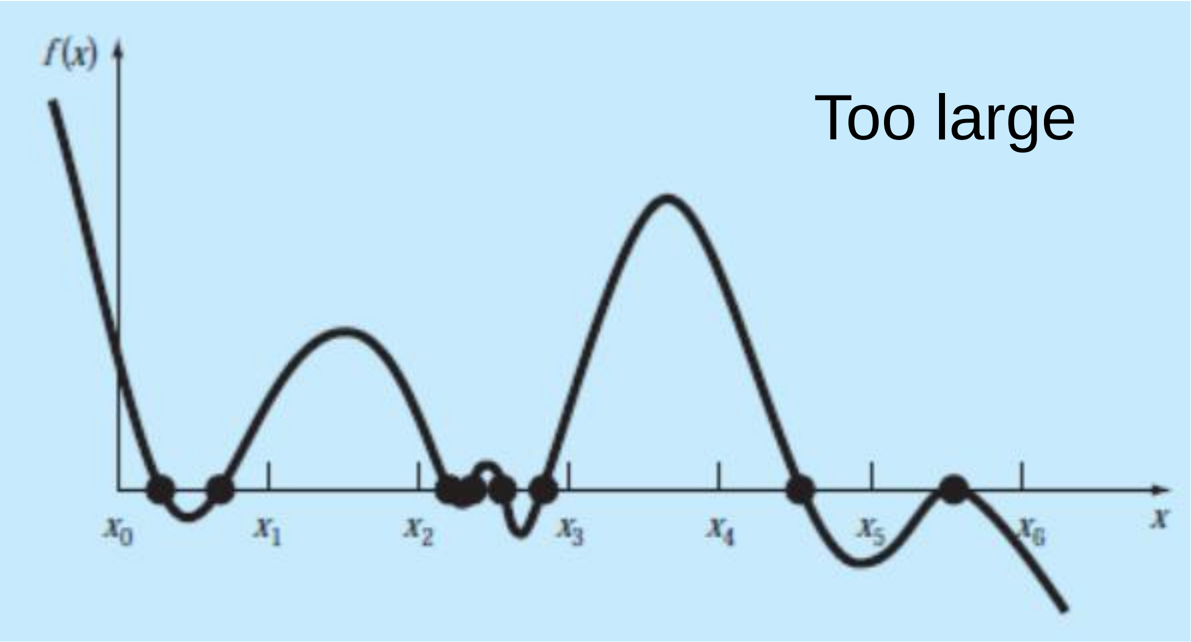
Initial step: Incremental search

Incremental search method

- Detecting a sign change in the interval [x, dx]
  - If change ➔ A solution exists
  - Difficulty:
    - Size of dx
      - Too small -> too long time
      - Too large -> missing solutions
    - Multiple roots
      - Possibility of missing a solution
        → Derivatives at both ends
        if different sign => maxima/minima in the interval
- This should be located at the initial step of finding a solution



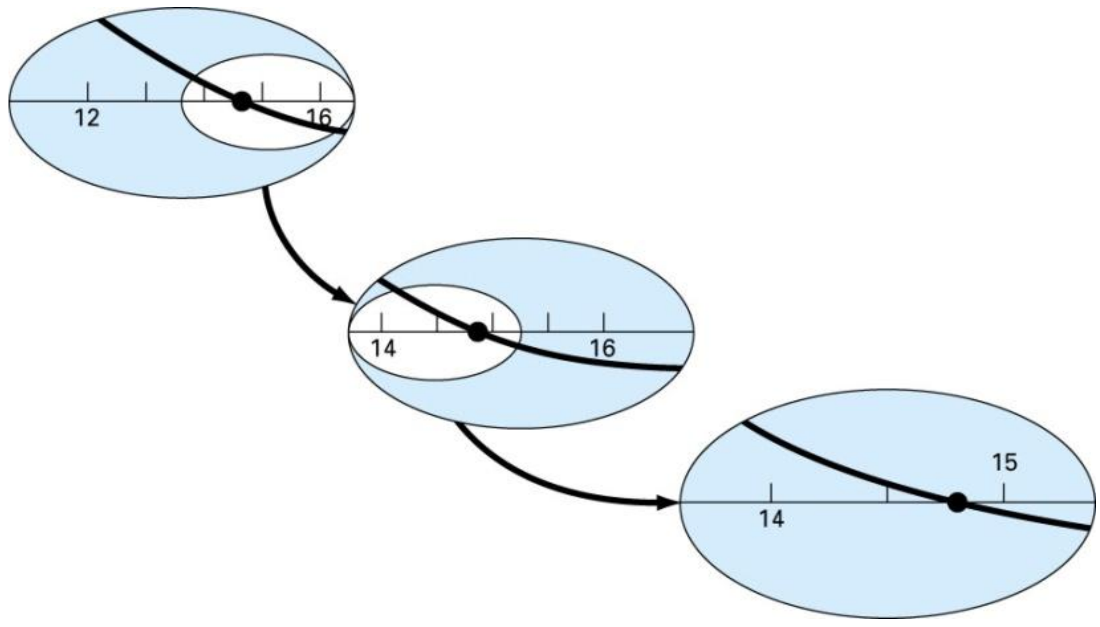
- Interval of incremental search



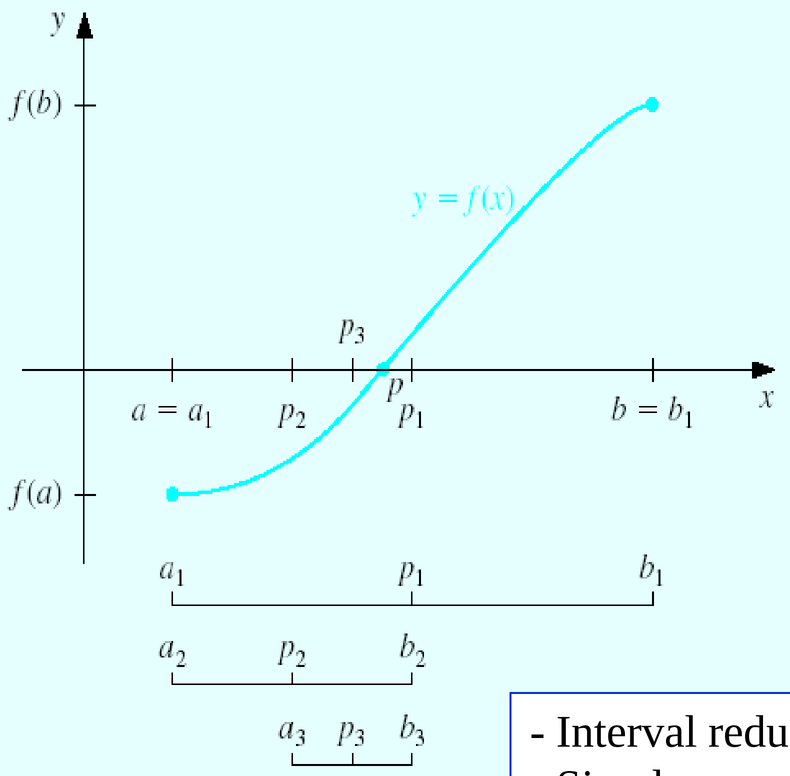
Bisection method(I)

=Half Interval method = Bolzano method

- Principle: A solution exists in [x1, x2] if f(x1)f(x2) < 0



Bisection method(II)

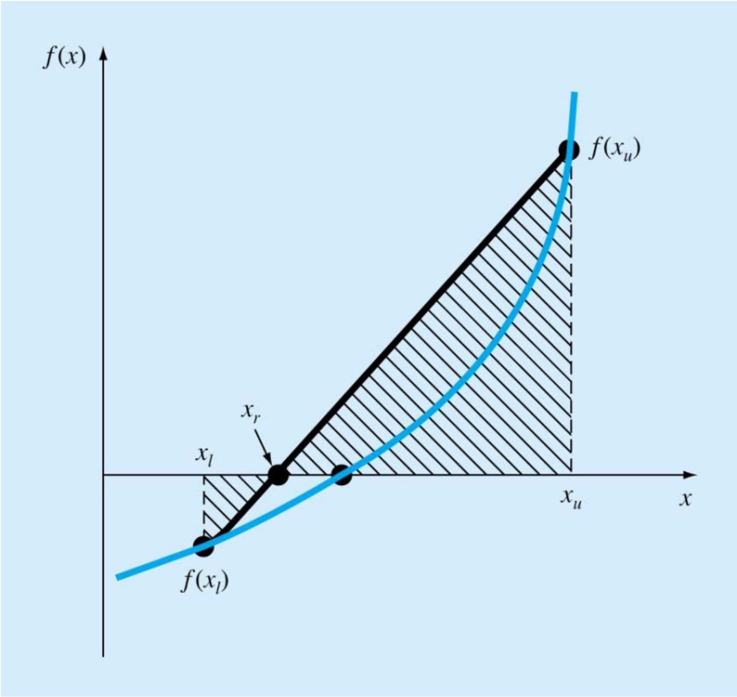


- Interval reduces by half at each iteration
- Simple
- At n-th iteration
  maximum error < (InitialInterval/2^n)
- Slow convergence
- Cannot cope with multiple roots

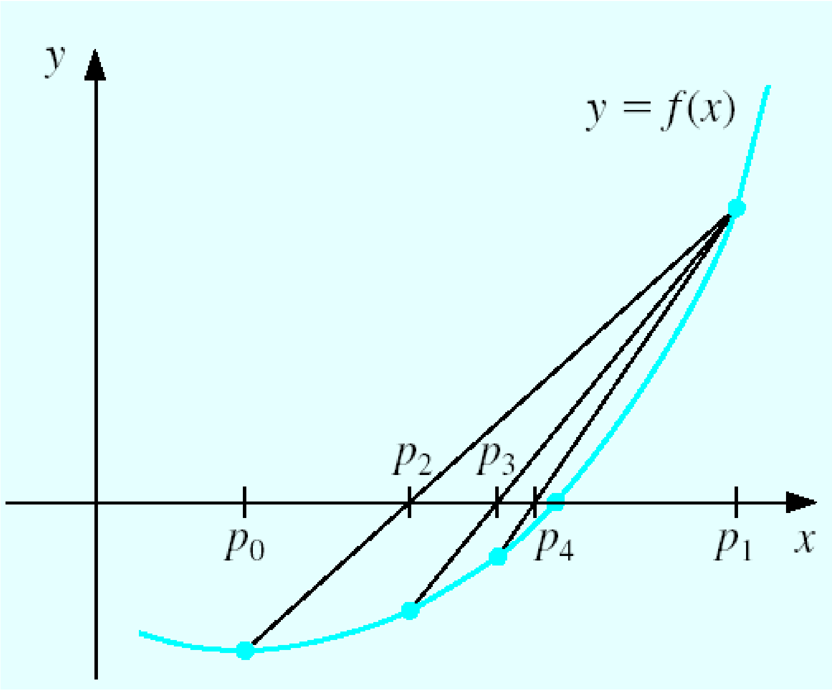
Linear interpolation method(I)

= False position method

- Principle



Algorithm



Linear interpolation method(II)

Algorithm

An interval $[a_{n+1}, b_{n+1}]$, for $n > 1$, containing an approximation to a root of $f(x) = 0$ is found from an interval $[a_n, b_n]$ containing the root by first computing

$$p_{n+1} = a_n - \frac{f(a_n)(b_n - a_n)}{f(b_n) - f(a_n)}.$$

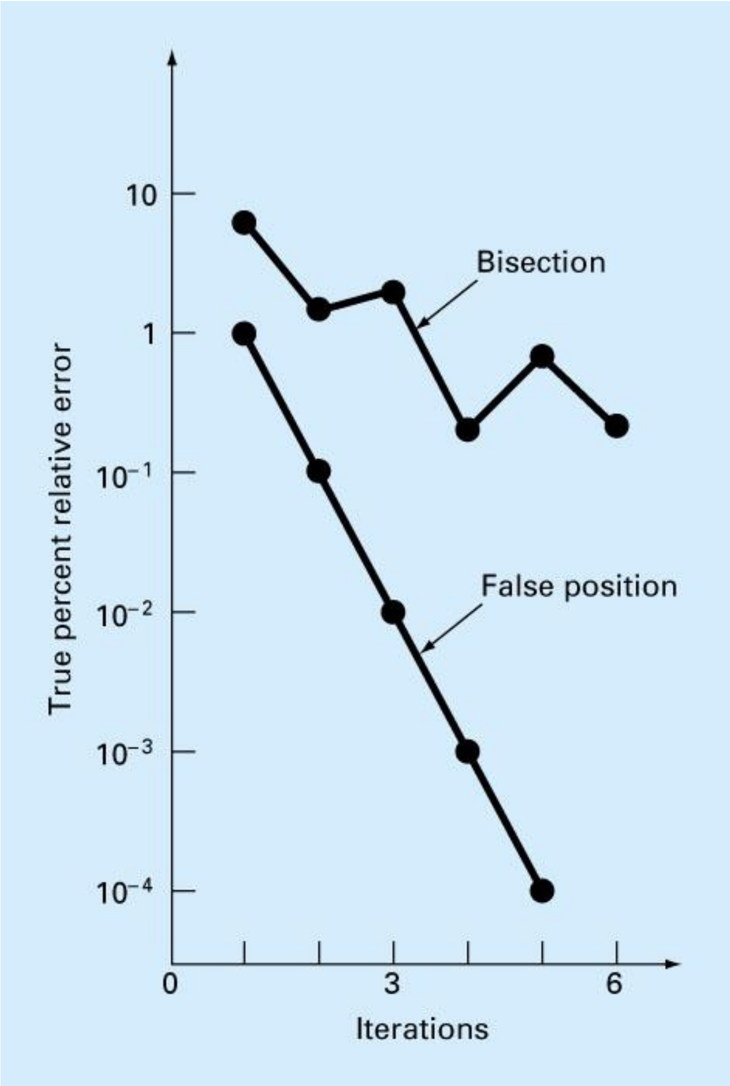
Then set

$$a_{n+1} = a_n \qquad \text{and} \qquad b_{n+1} = p_{n+1} \quad \text{if } f(a_n)f(p_{n+1}) < 0,$$

and

$$a_{n+1} = p_{n+1} \qquad \text{and} \qquad b_{n+1} = b_n \quad \text{otherwise.}$$

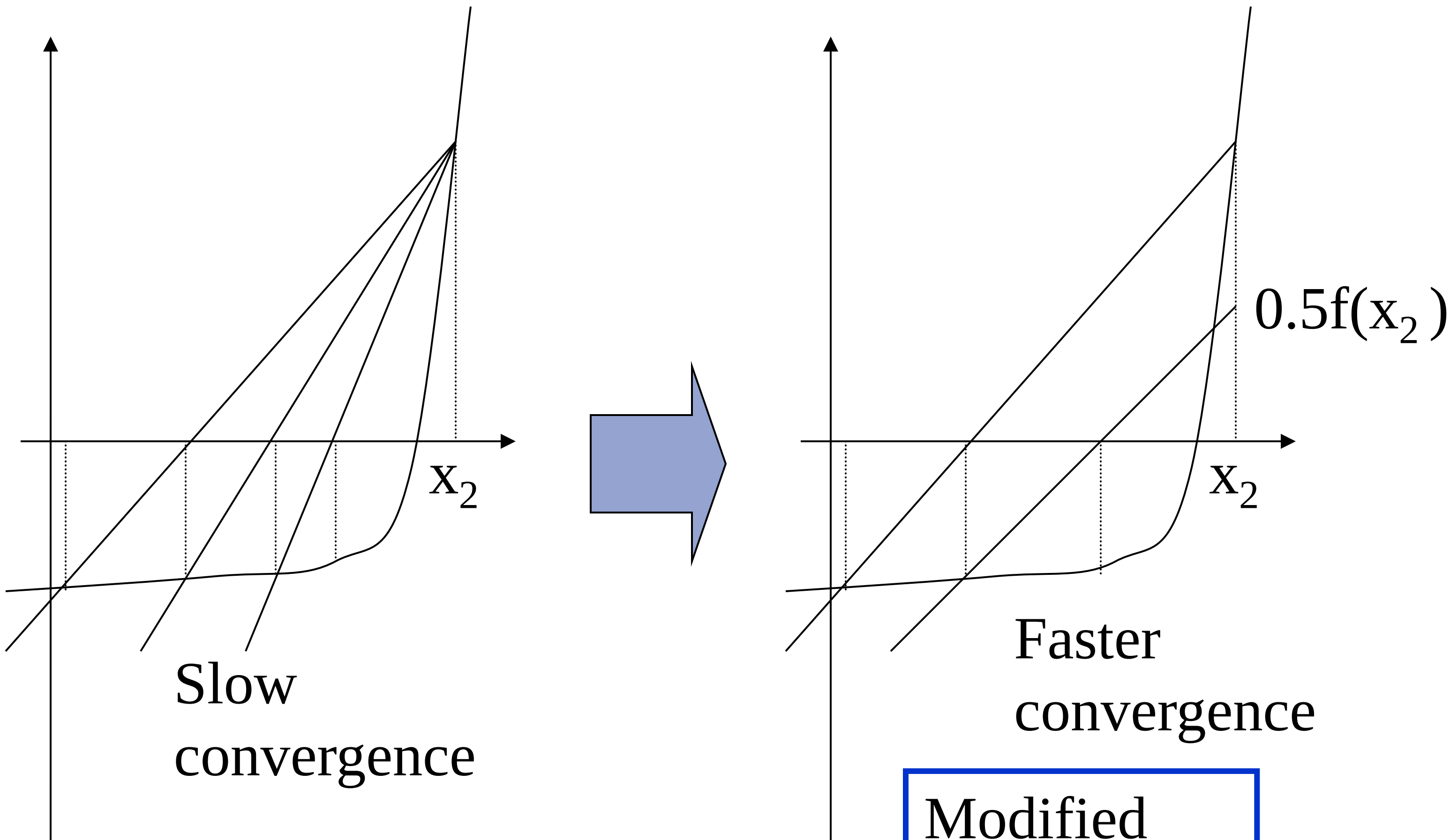
Convergence speed



- Faster than Bisection
- Convergence speed depends on curvature (Sometimes extremely slow)

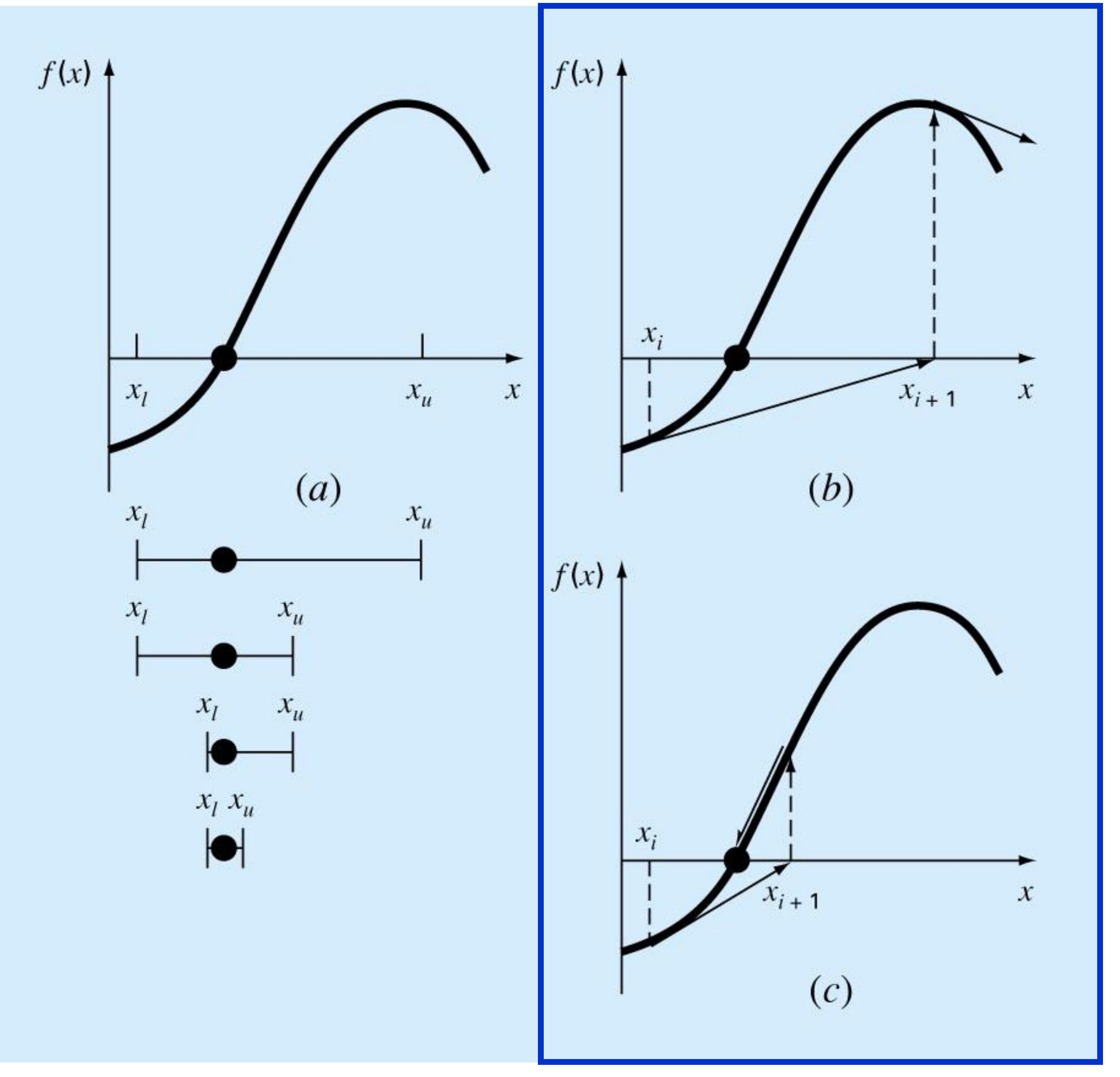
➔ Modified linear interpolation method

Modified linear interpolation



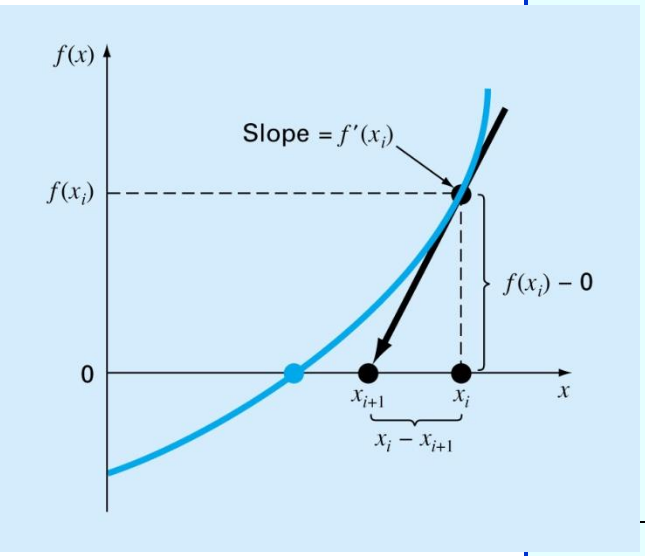
Modified Linear Interpolation

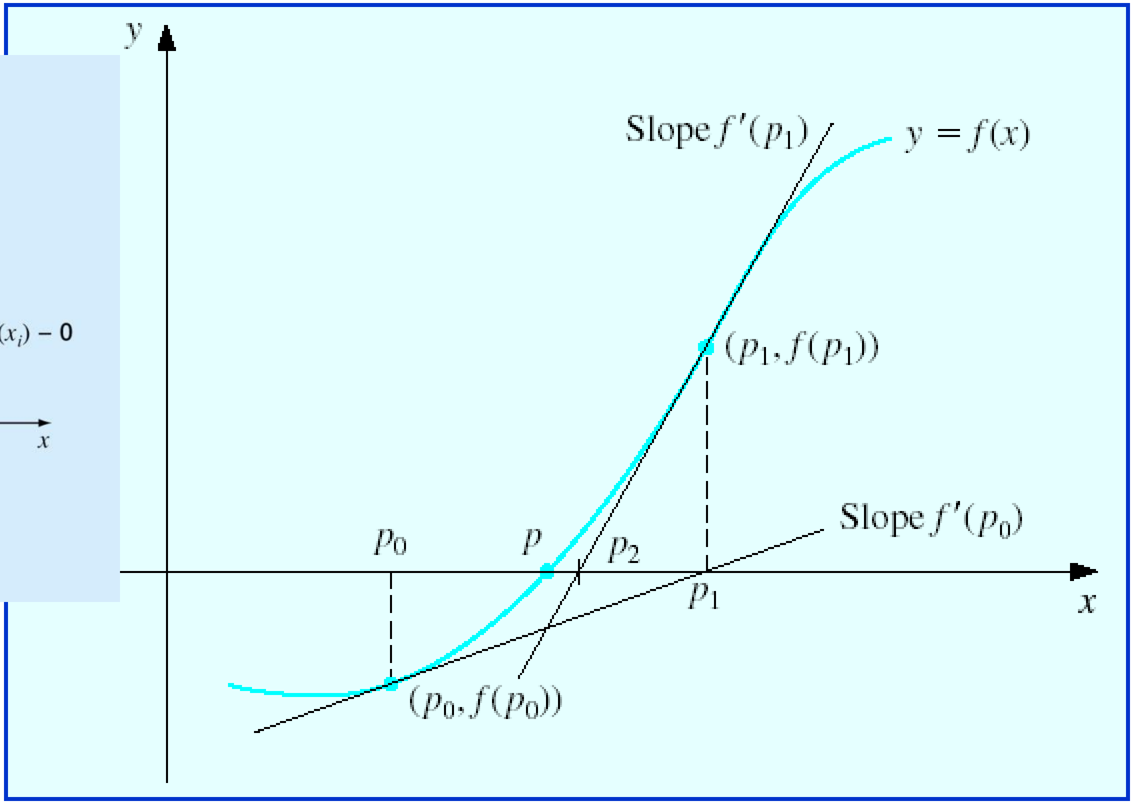
Open methods



Newton-Raphson method(I)

- Principle





The approximation p_{n+1} to a root of $f(x) = 0$ is computed from the approximation p_n using the equation

$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)}.$$

Newton-Raphson method(II)

- Fast convergence – quadratic convergence
- Inefficient if the derivative calculation is complex
- Initial guess matters
- Troubles
  - Cycling
  - Wandering
  - Overshooting

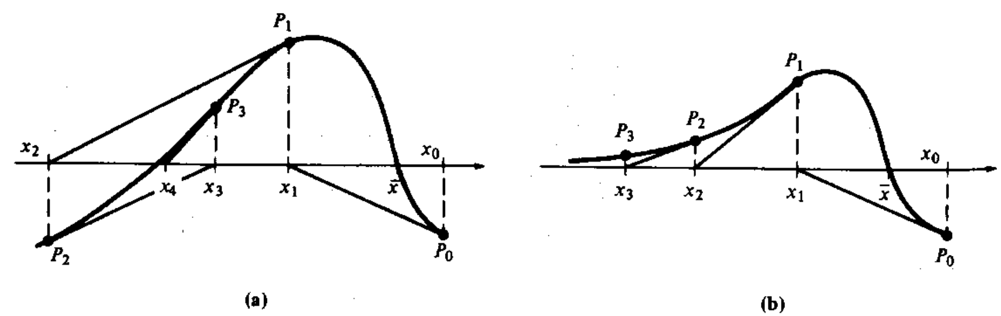


FIGURE 2.2-5 POSSIBLE CONSEQUENCES OF A POOR INITIAL GUESS ON NR.

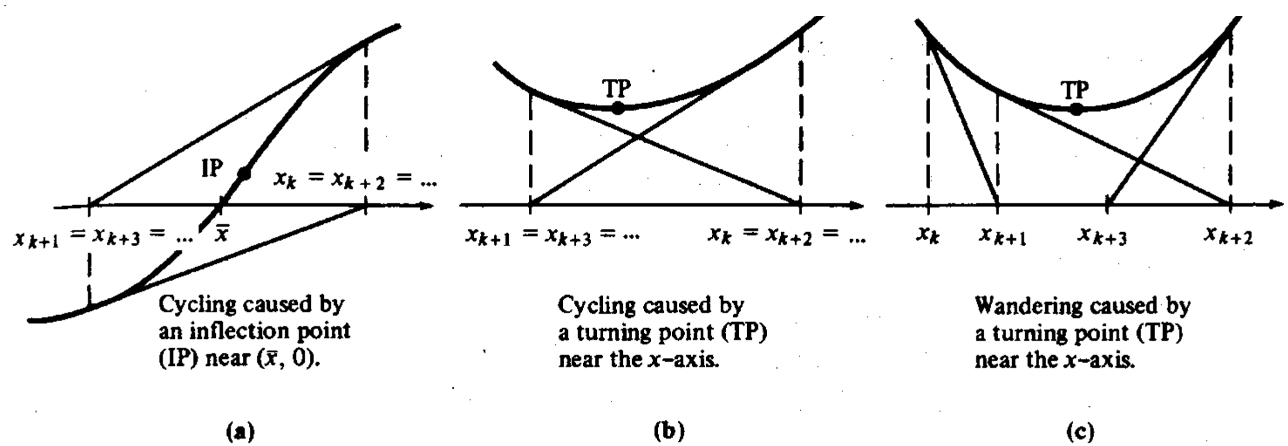
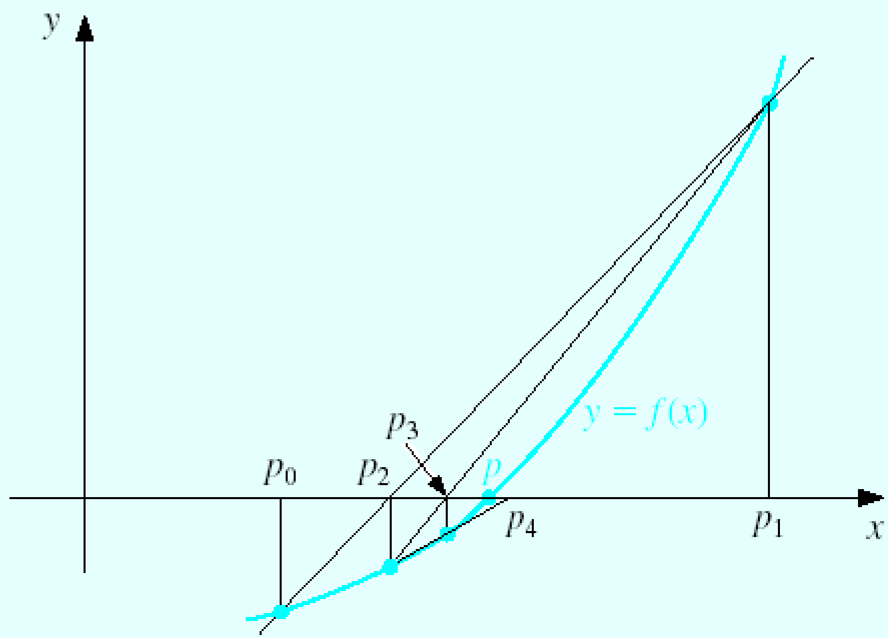


FIGURE 2.2-6 CYCLING AND WANDERING OF NR ITERATES.

Secant method(I)

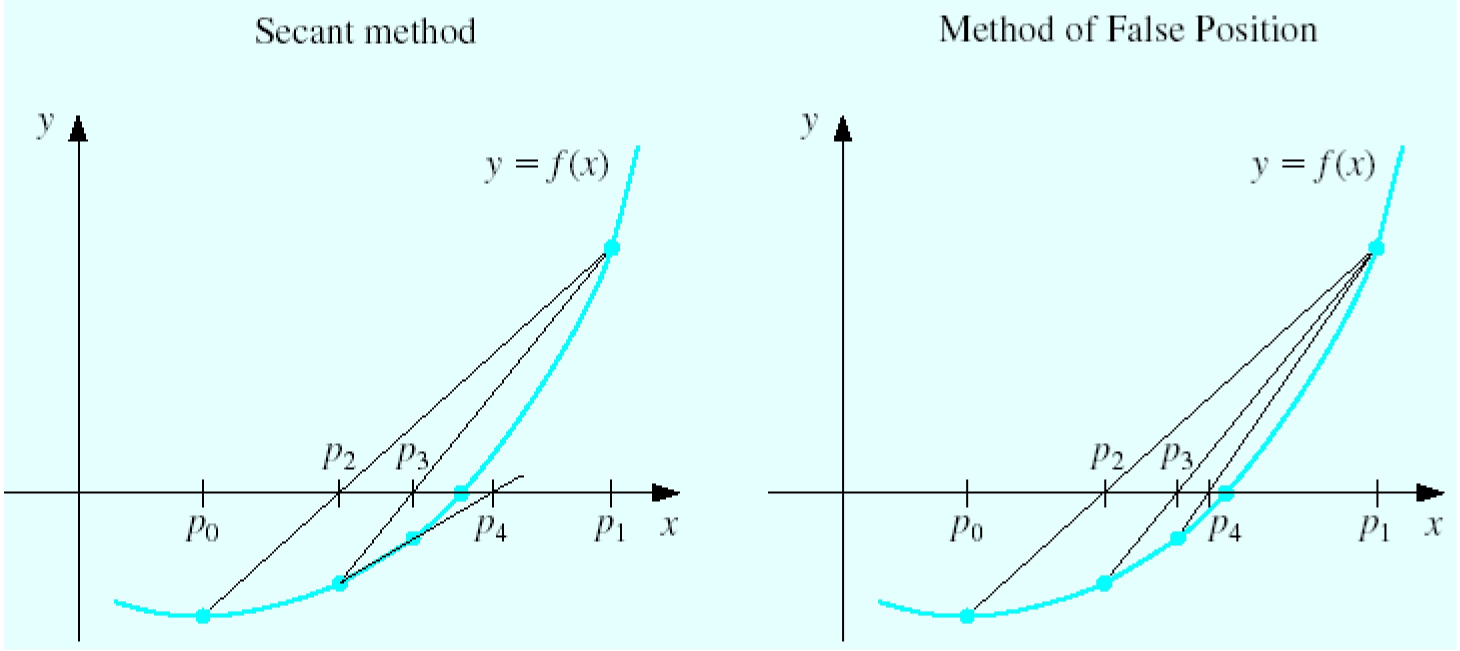


The approximation p_{n+1}, for $n > 1$, to a root of $f(x) = 0$ is computed from the approximations p_n and p_{n-1} using the equation

$$p_{n+1} = p_n - \frac{f(p_n)(p_n - p_{n-1})}{f(p_n) - f(p_{n-1})}.$$

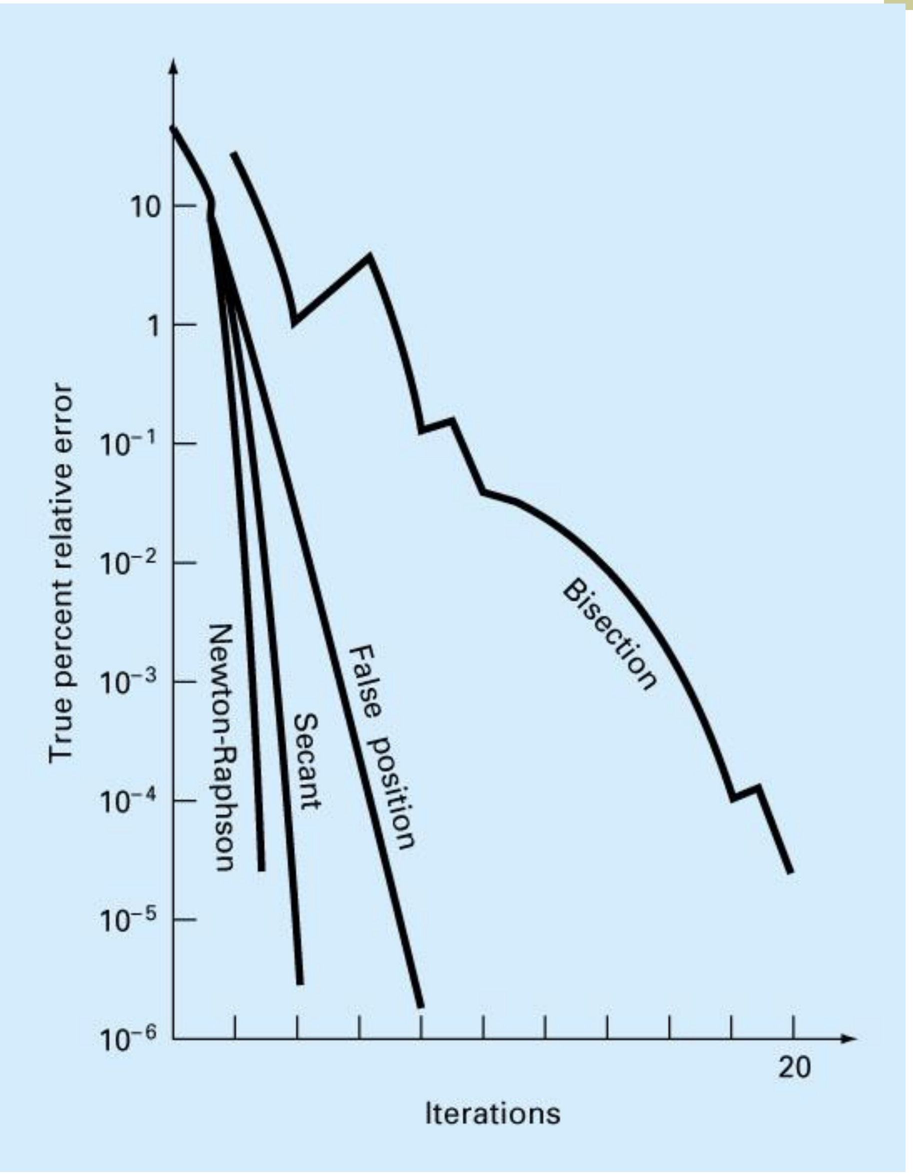
Secant method(II)

- Similar complexity to linear interpolation
  - Only update rule is different



- Faster convergence than linear interpolation
- More stable than Newton-Raphson method

Comparison: Convergence speed



Fixed point iteration(I)

- Principle

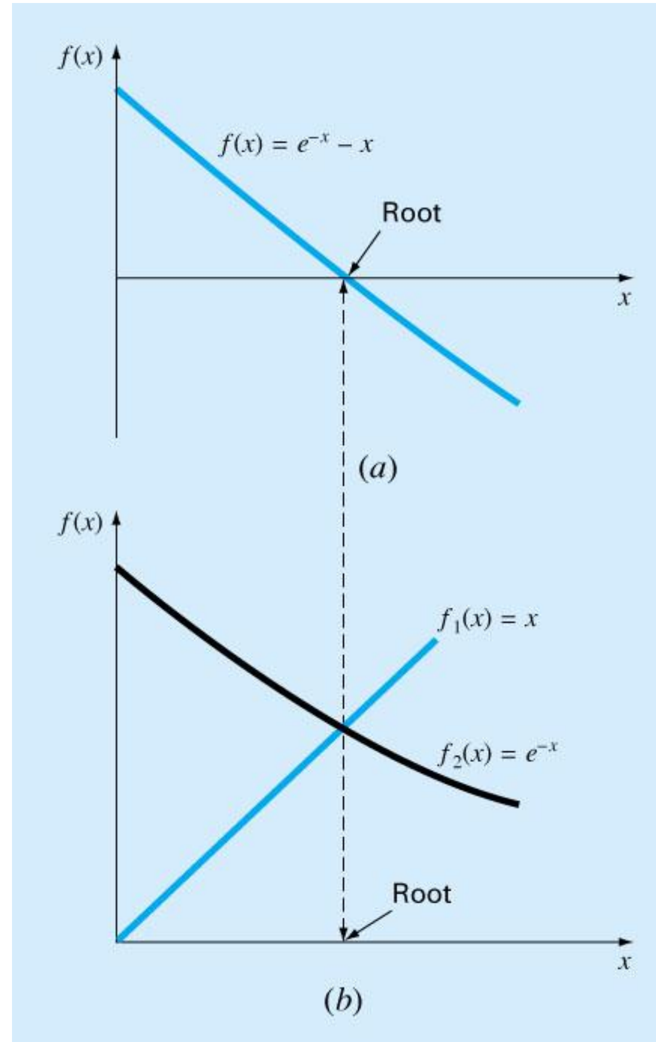
  f(x)=0 ➔ x=g(x)

  - Iteration rule

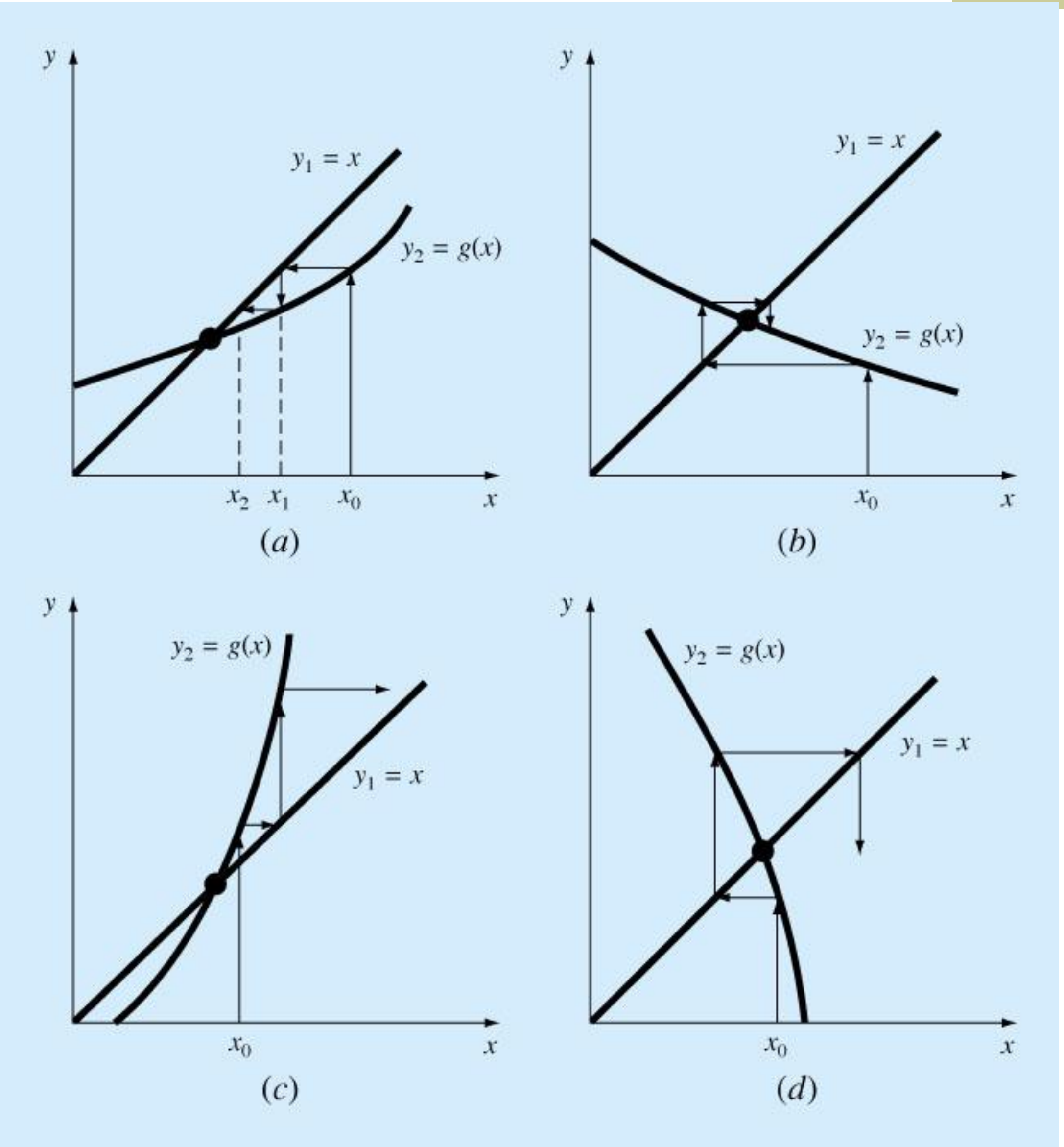
    $x_{k+1}=g(x_k)$

- Convergent

  if contraction mapping

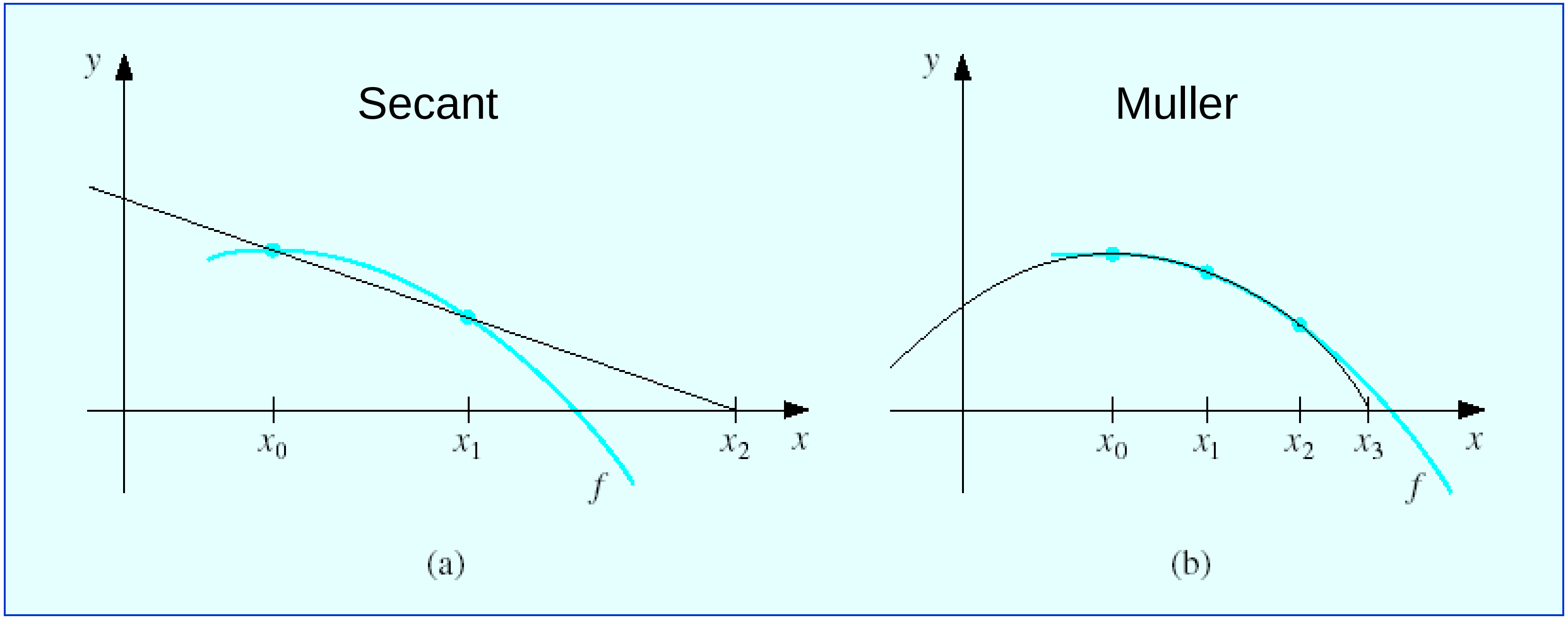


Fixed point iteration(II)



Muller method(I)

- Generalization of the Secant method
- Fast convergence



Muller method(II)

- Algorithm

Given initial approximations p_0, p_1, and p_2, generate

$$p_3 = p_2 - \frac{2c}{b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}},$$

where

$$c = f(p_2),$$

$$b = \frac{(p_0 - p_2)^2[f(p_1) - f(p_2)] - (p_1 - p_2)^2[f(p_0) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)},$$

and

$$a = \frac{(p_1 - p_2)[f(p_0) - f(p_2)] - (p_0 - p_2)[f(p_1) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)}.$$

Then continue the iteration, with p_1, p_2, and p_3 replacing p_0, p_1, and p_2.

Error analysis

- Linear convergence

$$|p - p_{n+1}| \le M|p - p_n|$$

- Quadratic convergence

$$|p - p_{n+1}| \le M|p - p_n|^2$$

Example 1 Suppose that $\{p_n\}$ converges linearly to $p = 0$, $\{\hat{p}_n\}$ converges quadratically to $p = 0$, and the constant $M = 0.5$ is the same in each case. Then

Table 2.7

n	Linear Convergence Sequence Bound $p_n = (0.5)^n$	Quadratic Convergence Sequence Bound $\hat{p}_n = (0.5)^{2^n-1}$
1	5.0000×10^{-1}	5.0000×10^{-1}
2	2.5000×10^{-1}	1.2500×10^{-1}
3	1.2500×10^{-1}	7.8125×10^{-3}
4	6.2500×10^{-2}	3.0518×10^{-5}
5	3.1250×10^{-2}	4.6566×10^{-10}
6	1.5625×10^{-2}	1.0842×10^{-19}
7	7.8125×10^{-3}	5.8775×10^{-39}

Accelerating convergence

- Aitken's Δ^2 method

If $\{p_n\}_{n=0}^{\infty}$ is a sequence that converges linearly to p, and if

$$q_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n},$$

then $\{q_n\}_{n=0}^{\infty}$ also converges to p, and, in general, more rapidly.

Example 2 The sequence $\{p_n\}_{n=1}^{\infty}$, where $p_n = \cos(1/n)$, converges linearly to $p = 1$. The first few terms of the sequences $\{p_n\}_{n=1}^{\infty}$ and $\{q_n\}_{n=1}^{\infty}$ are given in Table 2.8. It certainly appears that $\{q_n\}_{n=1}^{\infty}$ converges more rapidly to $p = 1$ than does $\{p_n\}_{n=1}^{\infty}$.

n	p_n	q_n
1	0.54030	0.96178
2	0.87758	0.98213
3	0.94496	0.98979
4	0.96891	0.99342
5	0.98007	0.99541
6	0.98614	
7	0.98981	

Rapid convergence!

Fail-safe methods

- Combination of Newton and Bisection
  - if(out of bound || small update by Newton)
    update by Bisection method
    ➔ rtsafe.c in NR in C

- Ridder's method (good performance)
  - Evaluating the function at the midpoint
  - Balancing efficiency and reliability
    ➔ zriddr.c in NR in C

Homework #2 (I)

[Due: 9/26]

- Programming: Find the roots of the Bessel function J_o in the interval [1.0, 10.0] using the following methods:
  - a) Bisection (rtbis.c)
  - b) Linear interpolation (rtflsp.c)
  - c) Secant (rtsec.c)
  - d) Newton-Raphson (rtnewt.c)
  - e) Newton with bracketing (rtsafe.c)
  - Hint
    - First, use bracketing routine (zbrak.c).
    - Then, call proper routines in NR in C.
    - Set xacc=10^{-6} x
    - **Use "pointer to function" to write succinct source codes**

Bessel functions can be found in Ch.6 of NR in C (bessj0.c, bessj1.c).
For Newton methods, use the following property of the Bessel function:

$$\frac{dJ_0(x)}{dx} = -J_1(x)$$

(Cont')

Homework #2 (II)

- Write source codes for Muller method (muller.c) and do the same job as above.
- Discuss the convergence speed of the methods.
- Solve *one interesting nonlinear equation you want to solve* using the routine of rtsafe.c in NR in C