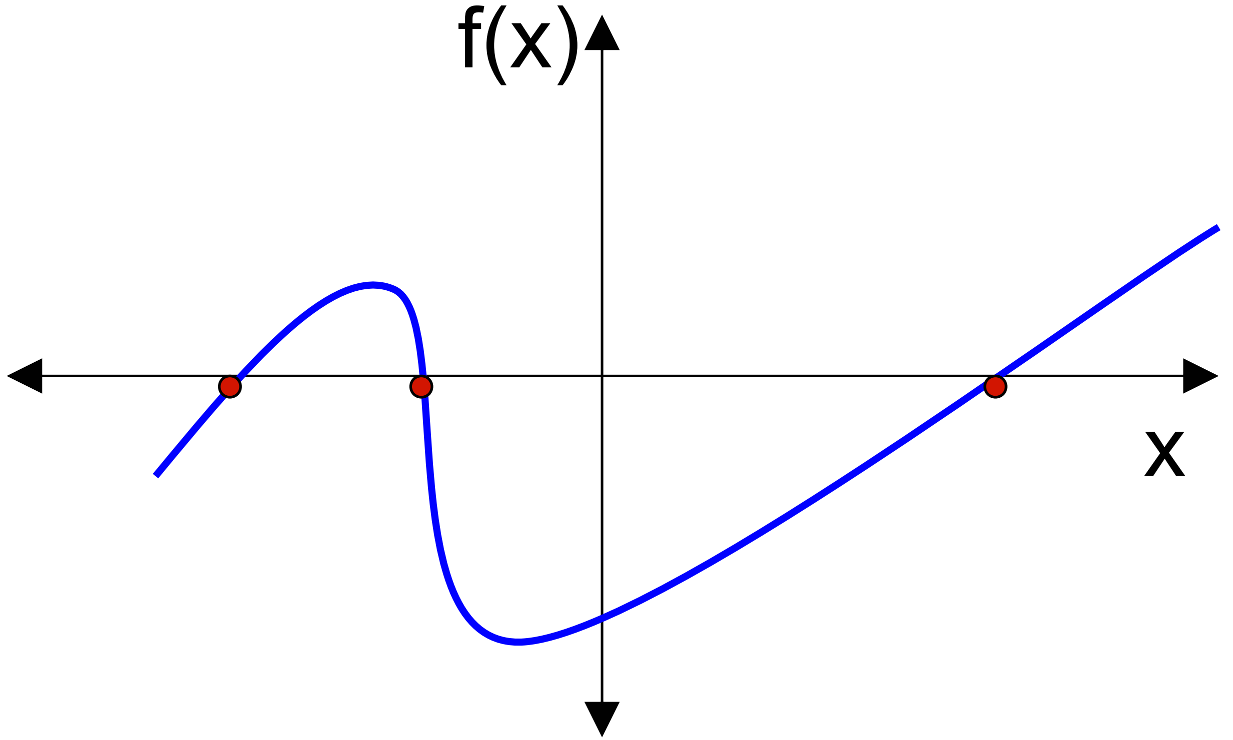
Numerical root finding

(Numerical Recipes in C++, Chapter 9)

OBJECTIVES:

- ❖ Follow the algorithms of root finding methods of solving a non-linear equation
- ❖ Use different methods to solve examples of finding roots of non-linear equation, and
- ❖ Enumerate the advantages and disadvantages of different root finding methods

- A basic problem in mathematics: solve an equation g(x) = y
- Convenient convention: rewrite this is f(x) = 0
- The solutions are called the roots of f



- We can also have a multidimensional problem in which coupled sets of equations must be solved

$f_1(x_1, x_2, \ldots) = 0$

$f_2(x_1, x_2, \ldots) = 0 \quad \rightarrow \quad \mathbf{f(x) = 0}$

……

(Much harder for the general multi-D non-linear case; but first let's look at the 1-D non-linear case, f(x) = 0)

Bracketing

- How do you know if a root exists?

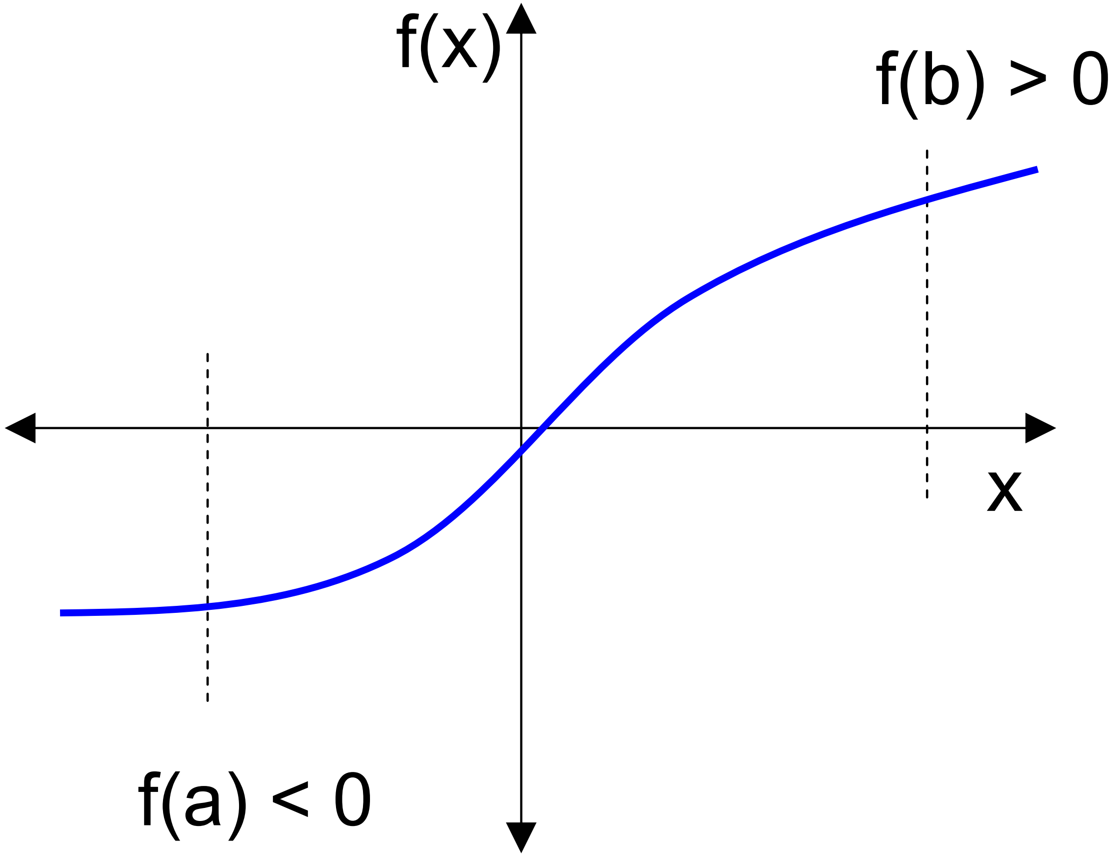
  If there a no singularities, then if you can find an interval [a,b] such that f(a) and f(b) have opposite signs, there must be at least one root between a and b

  Establishing such an interval is called “bracketing”

  Root finding without a bracket is tricky

Bracketing

- If there are no singularities, there must be a root between a and b



Root finding algorithms

- All root finding algorithms are iterative

  ➔ need a good guess (or small bracket) as a starting point

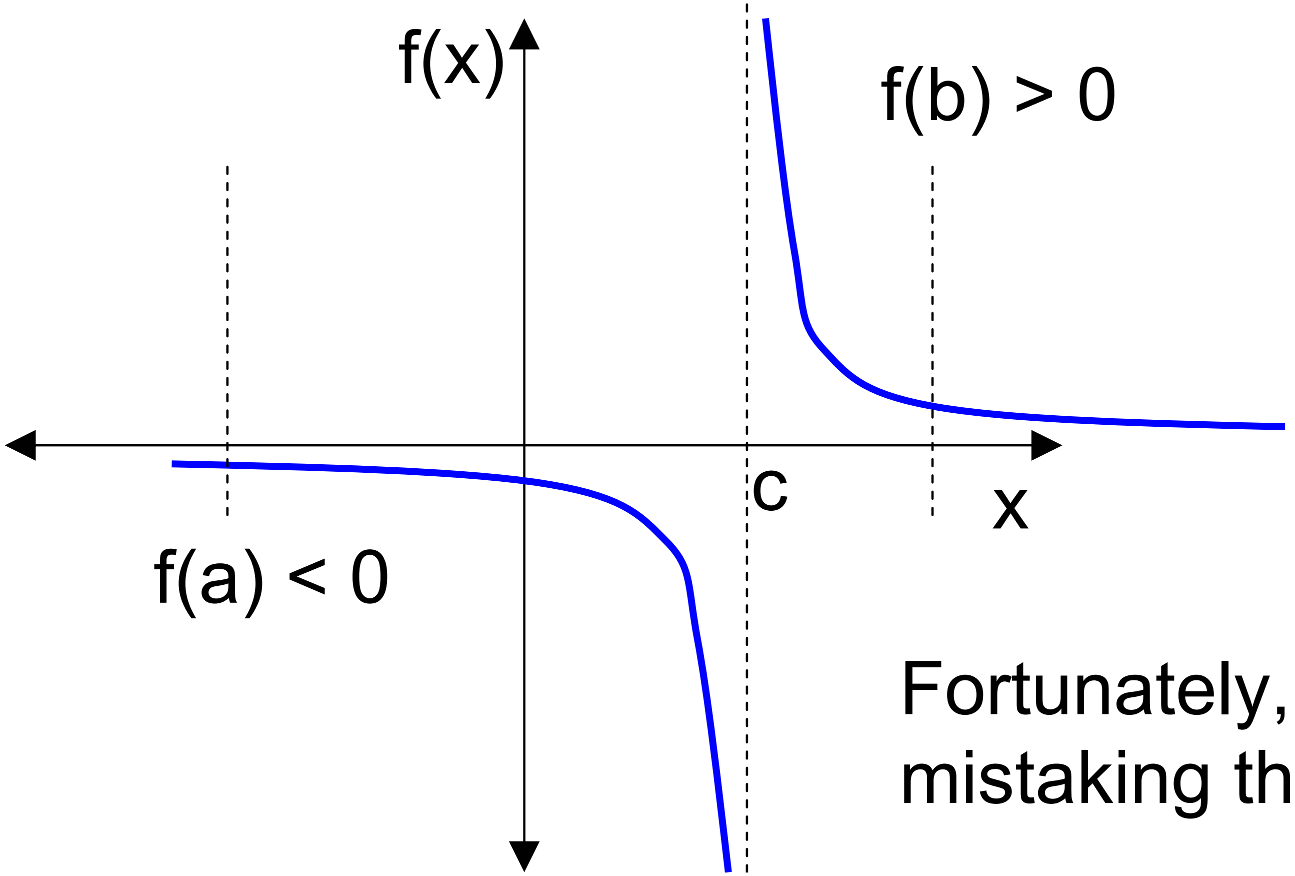
  Like numerical integration, we need to know something about our problem to create an optimal solution: making a graph never hurts

- In the 1-D case, there is a tradeoff between speed and robustness

- Slower methods maintain a bracket at all times (bisection, “false position”) – successive iterations simply contract the bracket
- Faster methods do not maintain a bracket and can shoot unstably away from the root (Newton-Raphson, “secant”)
- Hybrid methods seek to get the best of both worlds (Brent, Ridder, “safe Newton-Raphson”)

Bracketing

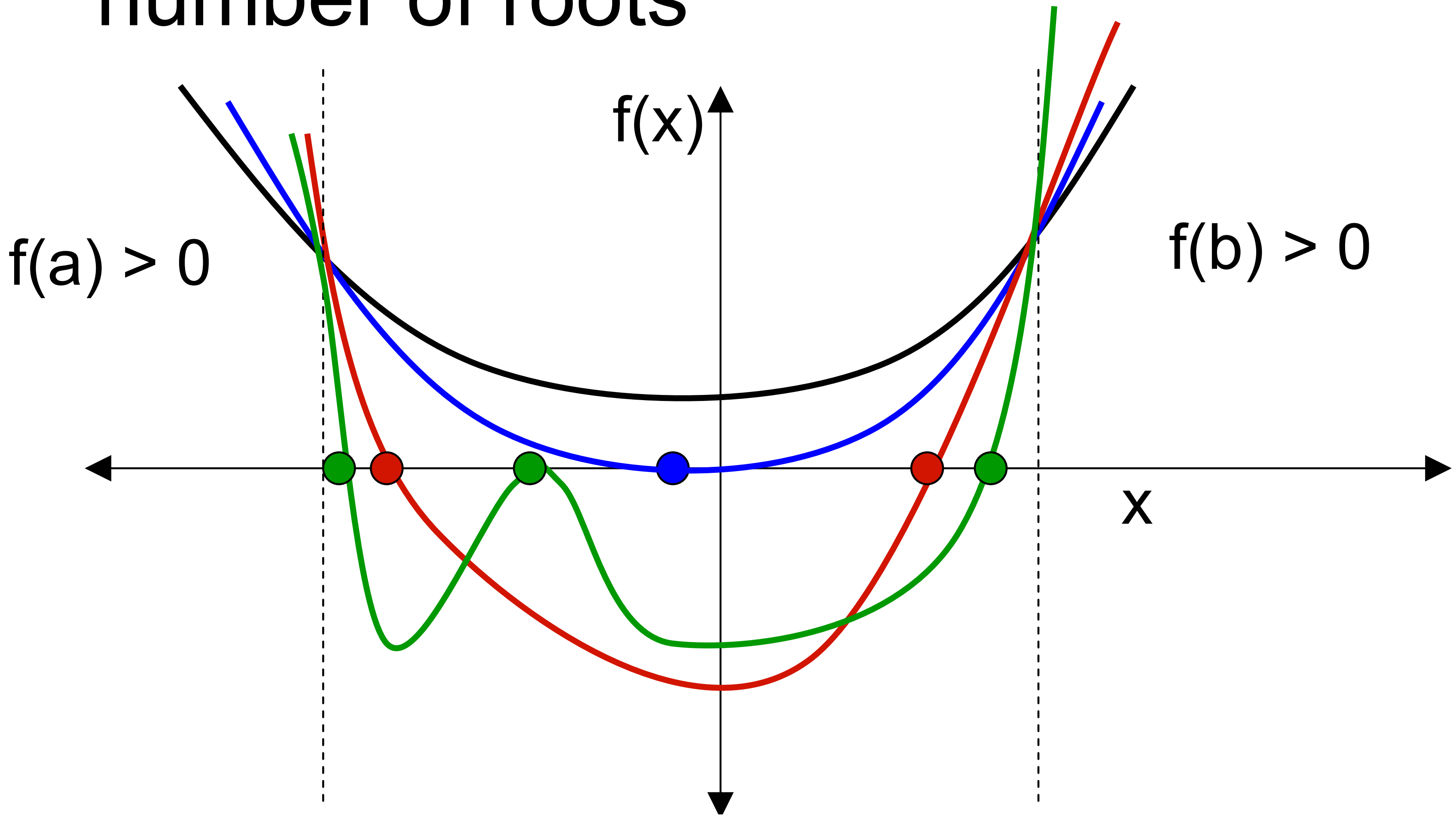
- A bracket is usually sufficient, although a singularity is still possible: e.g. $f(x) = (x - c)^{-1}$



Fortunately, no danger of mistaking this for a root!

Bracketing

- The absence of a bracket can imply any number of roots



Finding an initial bracket

- *Recipes* describes and provides two bracket-finding routines:

- ZBRAC: expands a seed interval looking for a bracket
- ZBRAK: subdivides a seed interval looking for brackets (good for multiple roots)

Pathologies

- Some functions have roots, but it is impossible to find brackets

  e.g. *Recipes* equation (3.0.1)

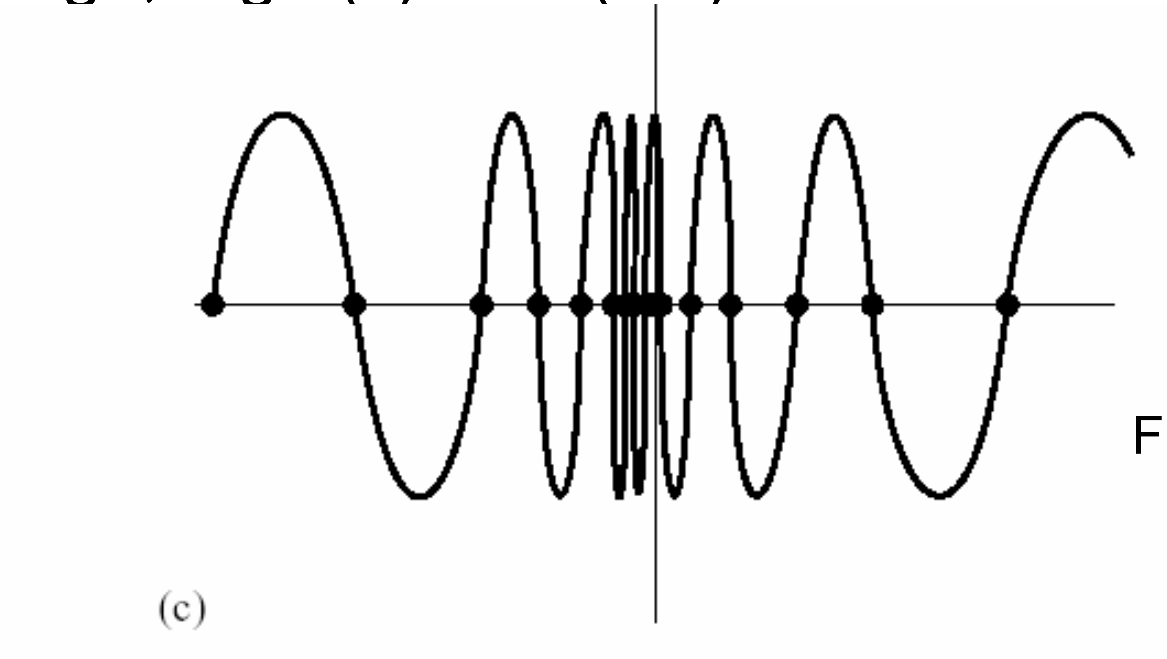
  $0 = f(x) \equiv 3x^2 + (1/\pi^4) \ln [(x - \pi)^2] + 1$

  Has a weak singularity at $x = \pi$, two roots very near $x = \pi$, but is negative only in the interval $x = \pi \pm 10^{-667}$

  You will probably never find a bracket even in quad precision

Pathologies

- Some functions have infinitely many roots over a finite range, e.g. f(x) = sin(1/x)



From *Recipes*, Fig 9.1.1

- MORAL: understand your function before jumping into a numerical solution

Root finding algorithms: bisection

- Once you have found a bracket, it can always be narrowed using the bisection method:
  - Divide [a,b] into [a , ½(a+b)] and [½(a+b), b]
  - Keep the valid bracket and repeat until the interval reaches a preset size
  - Totally foolproof – will also find singularities if they exist

Performance of bisection

- In bisection, the size of the bracket after n iterations is $\varepsilon_n = \frac{1}{2}\,\varepsilon_{n-1}$
- This is called linear convergence, because $\varepsilon_n \propto (\varepsilon_{n-1})^m$ with m = 1
- Faster methods have m > 1 and have "supralinear" convergence

Performance of bisection

- Note that linear convergence isn't at all bad: we still have $\varepsilon_n = 2^{-n}\,\varepsilon_0$
- Number of interations needed to achieve precision ε is $\log_2(\varepsilon_0/\varepsilon)$
- Need to quit anyway when ε approaches machine accuracy (for which n might typically be ~ 40 in double precision)

False– Position Method of Solving a Nonlinear Equation

- Bisection only makes use of information about the sign of the function. We can do better if we consider also the magnitude

- The secant and false position methods both use linear interpolation to decide how to narrow the bracket
  - False position always maintains a valid bracket
  - Secant doesn't necessarily, and is therefore less robust (but faster)

Introduction

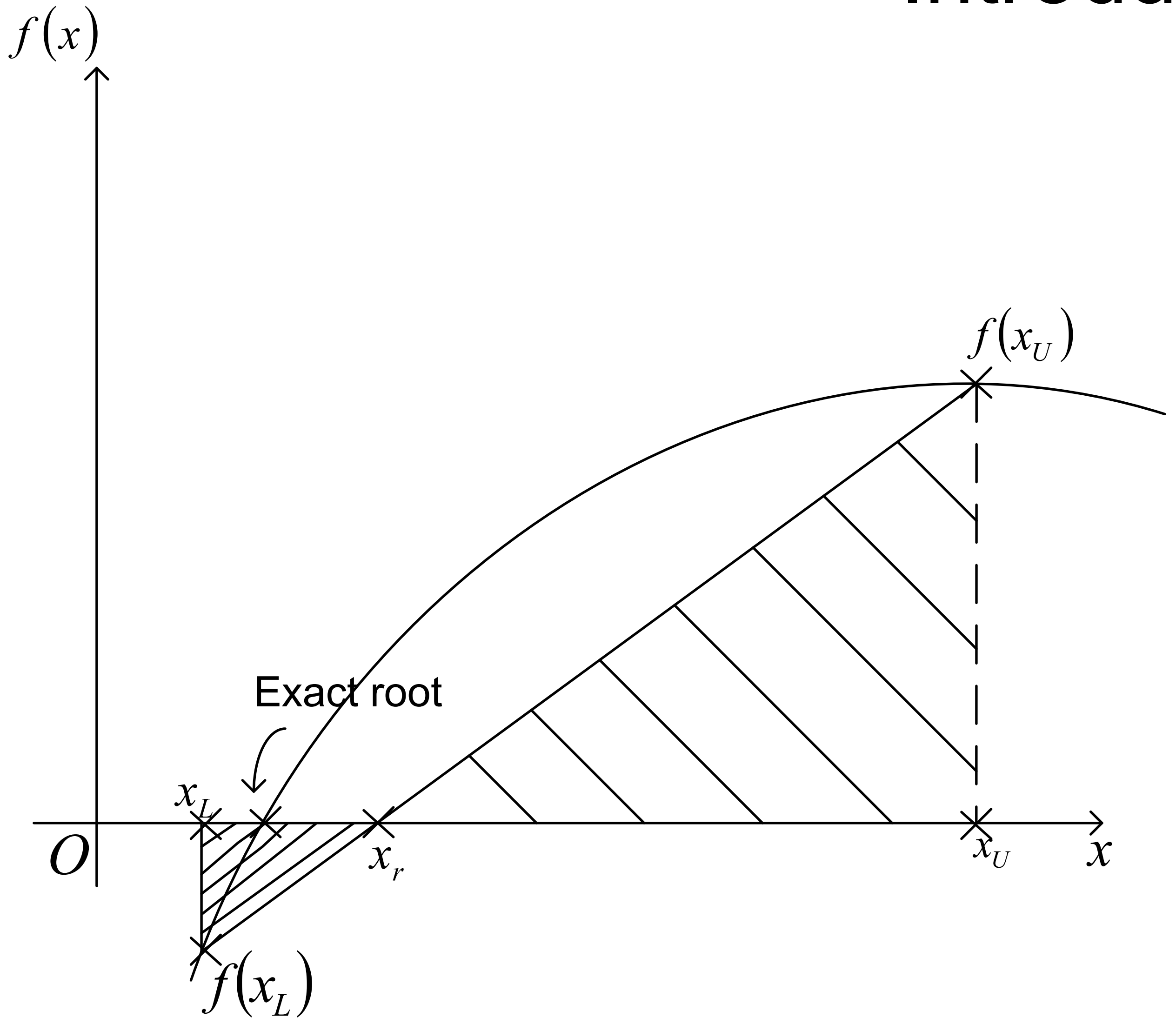


Figure 6 False-Position Method

$$f(x) = 0 \qquad (1)$$

In the Bisection method

$$f(x_L) * f(x_U) < 0 \qquad (2)$$

$$x_r = \frac{x_L + x_U}{2} \qquad (3)$$

Based on two similar triangles, shown in Figure 6, one gets:

$$\frac{f(x_L)}{x_r - x_L} = \frac{f(x_U)}{x_r - x_U} \tag{4}$$

The signs for both sides of Eq. 4 is consistent, since:

$$f(x_L) < 0;\ x_r - x_L > 0$$
$$f(x_U) > 0;\ x_r - x_U < 0$$

From Eq. (4), one obtains

$$(x_r - x_L)f(x_U) = (x_r - x_U)f(x_L)$$

$$x_U f(x_L) - x_L f(x_U) = x_r \{f(x_L) - f(x_U)\}$$

The above equation can be solved to obtain the next predicted root x_r as

$$x_r = \frac{x_U f(x_L) - x_L f(x_U)}{f(x_L) - f(x_U)} \tag{5}$$

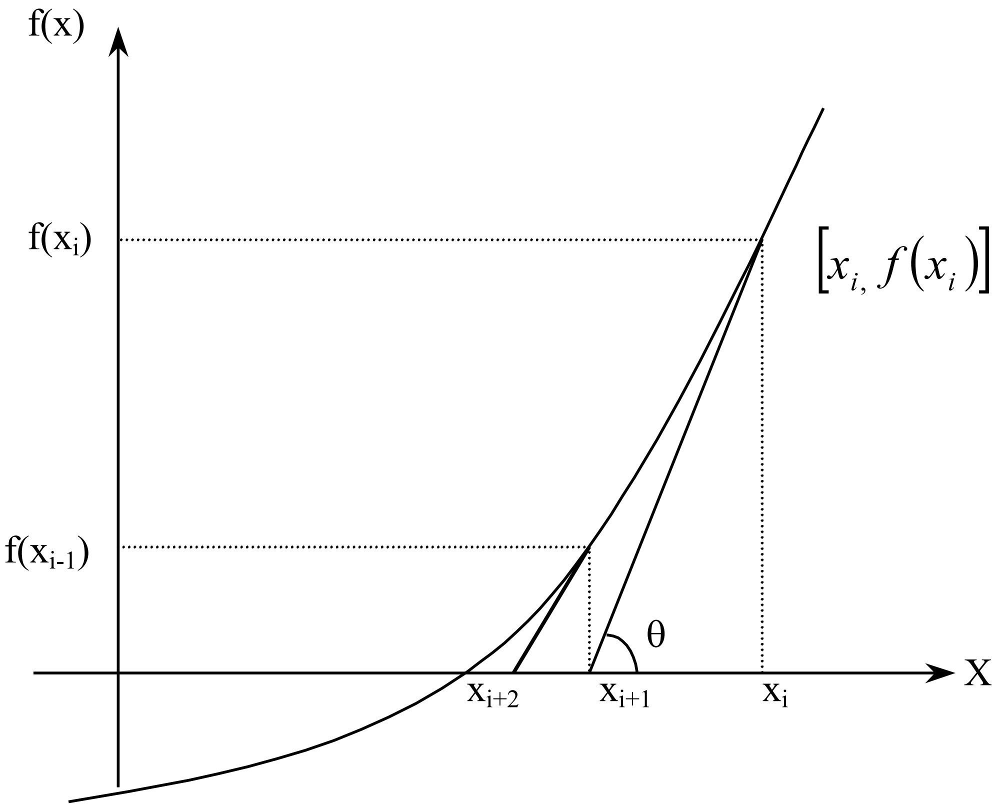
The above equation,

$$x_r = x_U - \frac{f(x_U)\{x_L - x_U\}}{f(x_L) - f(x_U)} \tag{6}$$

Or

$$x_r = x_L - \frac{f(x_L)}{\left\{\dfrac{f(x_U) - f(x_L)}{x_U - x_L}\right\}} \tag{7}$$

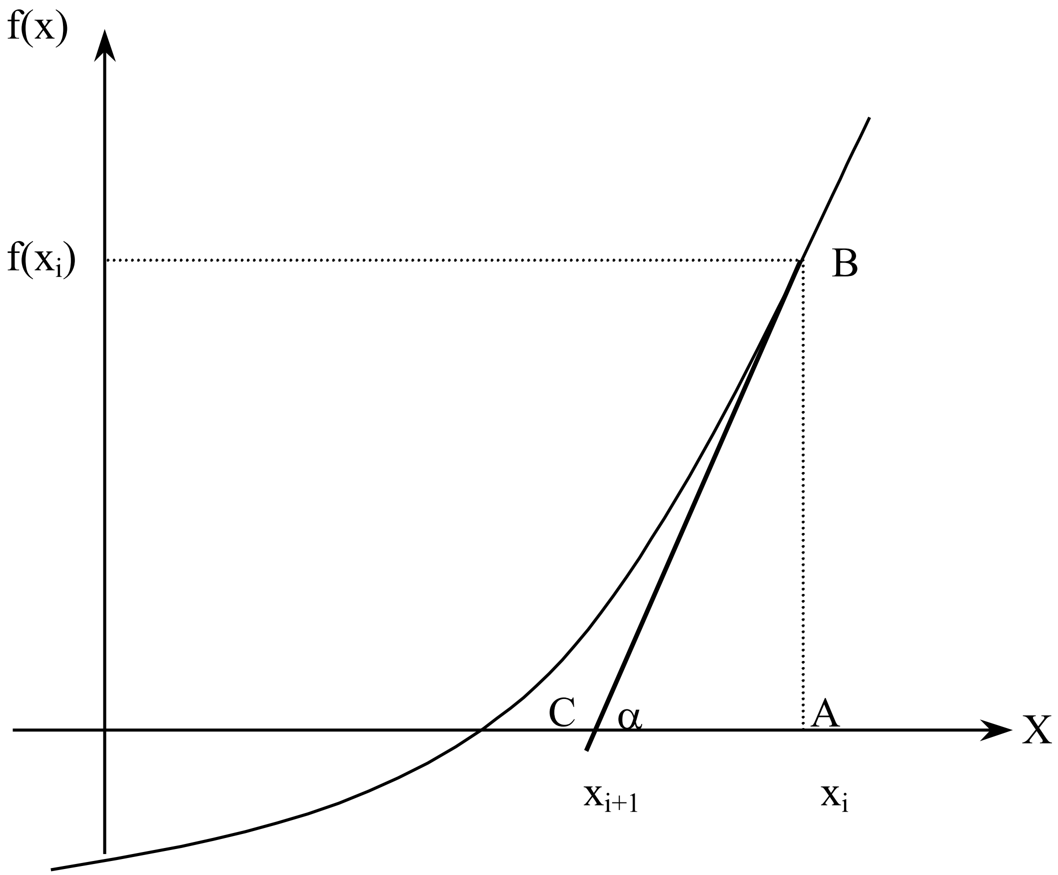
Newton-Raphson Method of Solving a Nonlinear Equation



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

Figure 7 Geometrical illustration of the Newton-Raphson method.

Derivation



$$\tan(\alpha) = \frac{AB}{AC}$$

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

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

Figure 8 Derivation of the Newton-Raphson method.

Newton-Raphson

- When the derivative of f(x) is known, and when f(x) is well behaved, the celebrated (and ancient) Newton-Raphson method gives the fastest convergence of all ("quadratic", i.e. m = 2, such that $\varepsilon_n = \varepsilon_{n-1}^2$)

- Relies on the Taylor expansion

  $f(x + \delta) = f(x) + \delta f'(x) + ½ \delta^2 f''(x) + \ldots..$

  If *ith* iteration, x_i, is close to the root, then for the next iteration, try $x_{i+1} = x_i + \delta$ with $\delta = - f(x_i) / f'(x_i)$

- Convergence is rapid, and the method is very useful for "polishing" a root (i.e. refining an estimate that is nearly correct)

Clearly, a few iterations usually yields an accurate result in the limit of small δ, because terms of order $\frac{1}{2}\,\delta^2 f''(x)$ or higher are much smaller than $\delta\, f(x)$

$$f(x + \delta) = f(x) + \delta\, f'(x) + \tfrac{1}{2}\,\delta^2 f''(x) + \ldots..$$

Exception: when $f'(x)$ is very small (or zero)