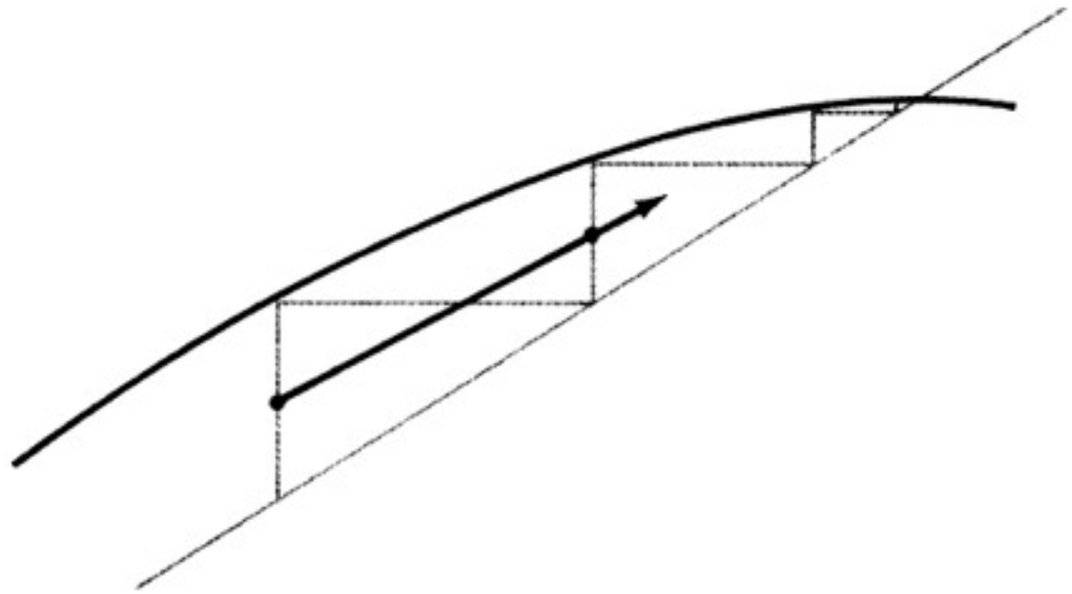
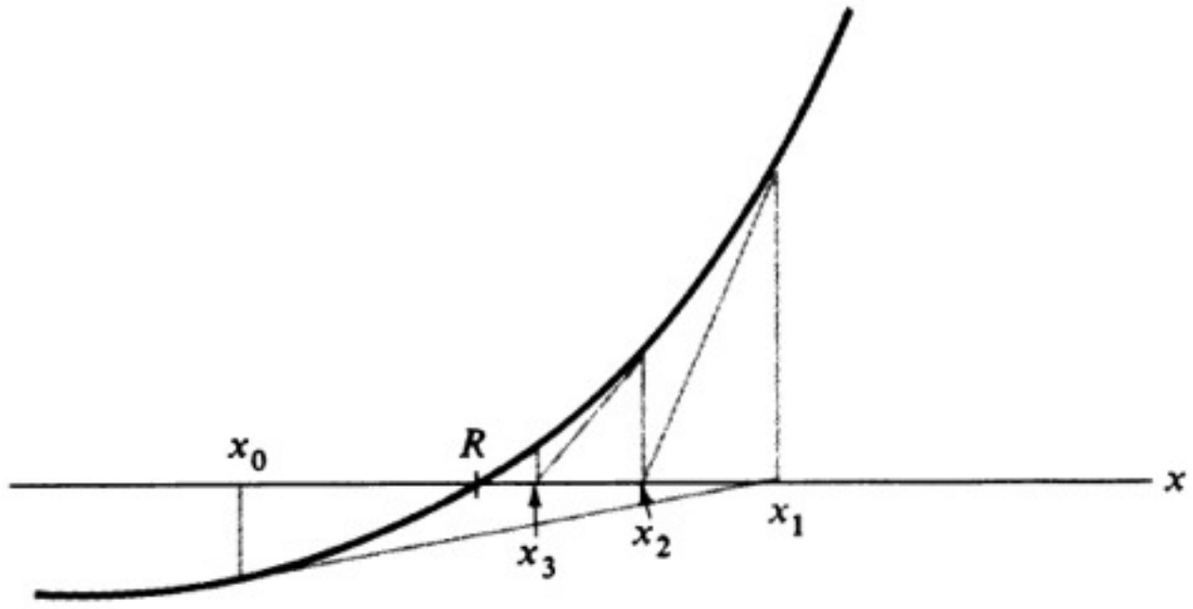
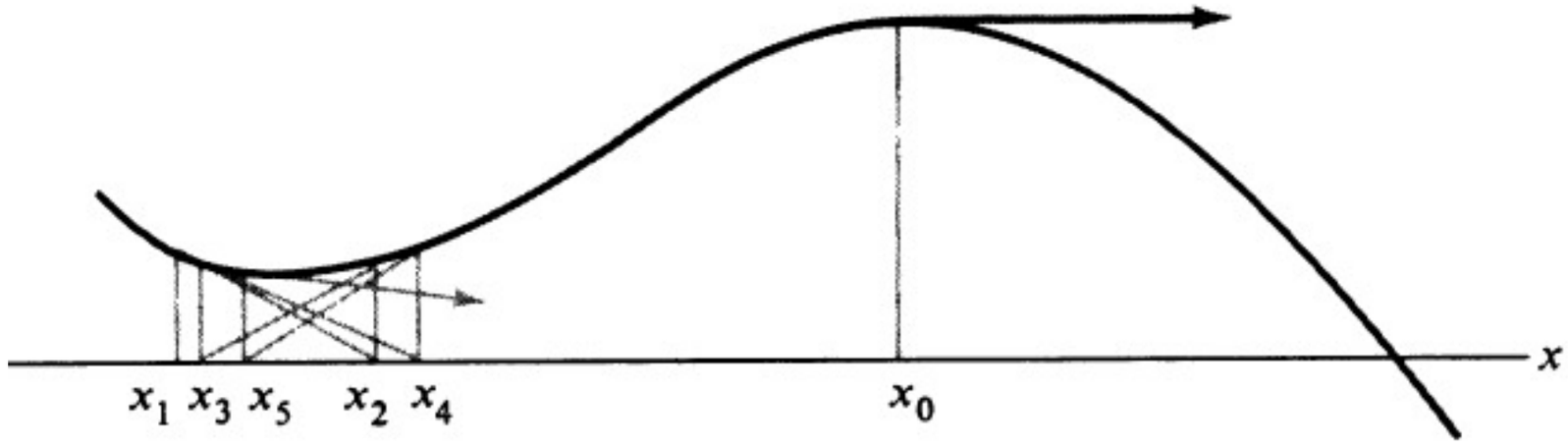


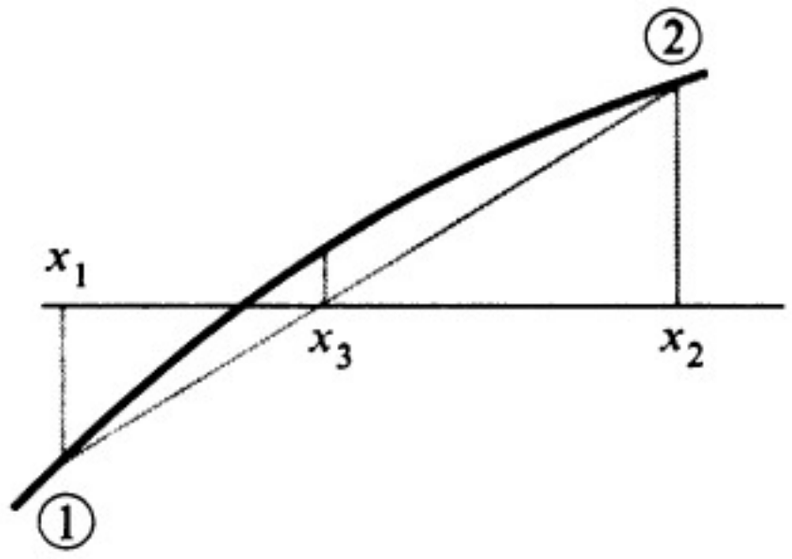
Convergence towards root in a more directed manner



Using tangent direction making convergence very fast

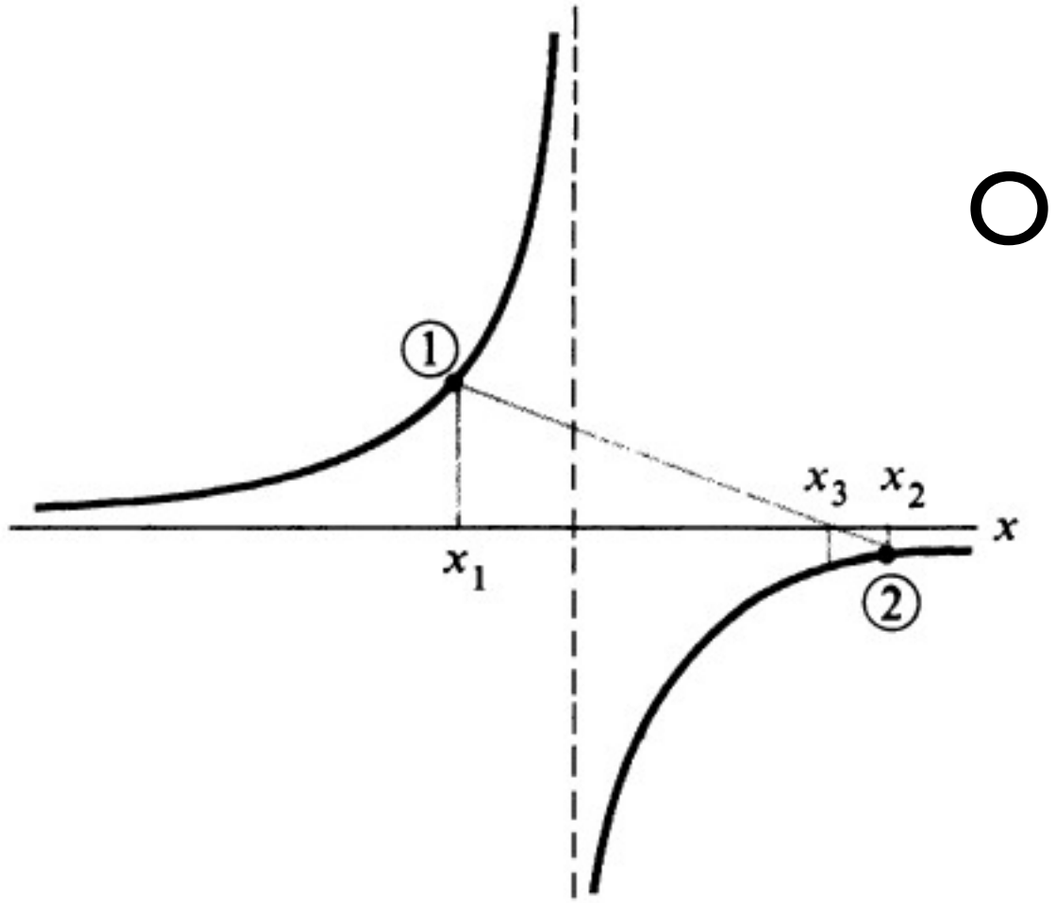


Dangers of Newton's method when slope ~ zero



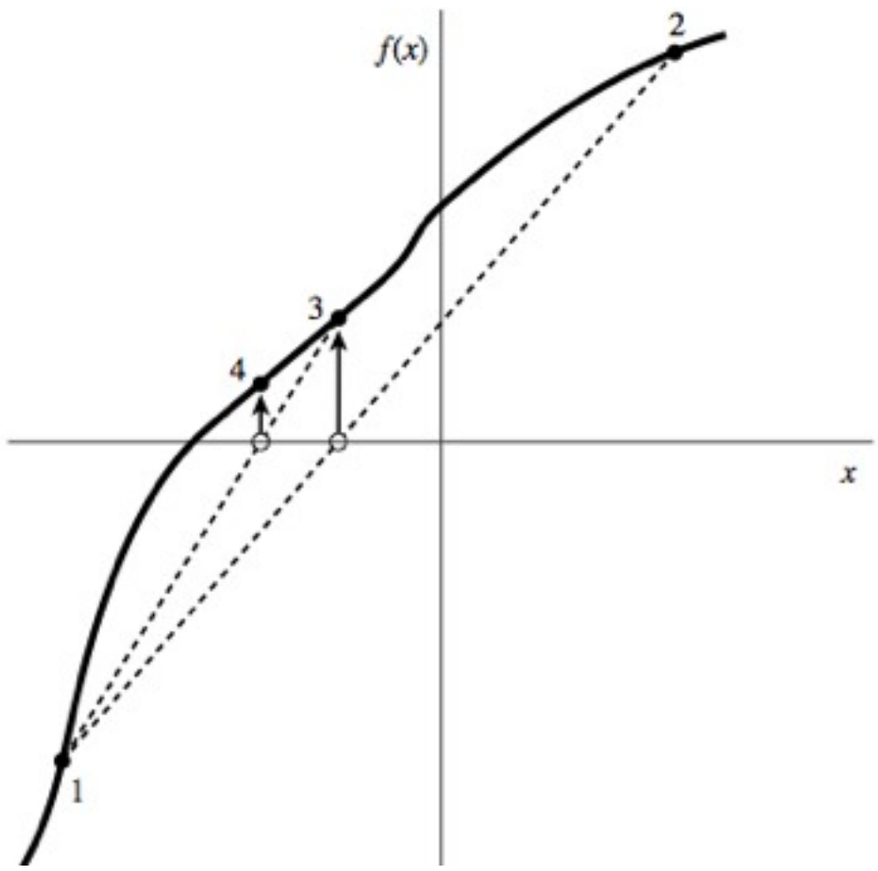
$$|\varepsilon_{i+1}| \to |\varepsilon_i|^{(1+\sqrt{5})/2} \cdot K$$

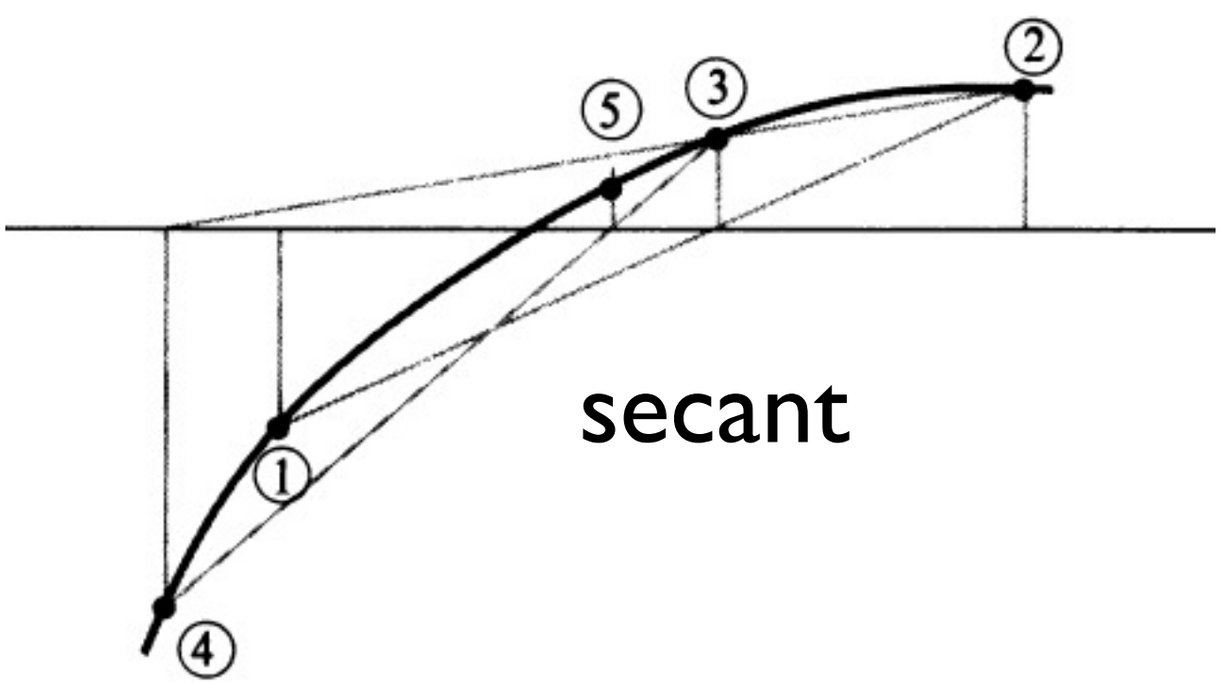
Use of secant instead of tangent. Slow but sure.



One of the dangers is getting discontinuity.

false-secant or regula-falsi





secant

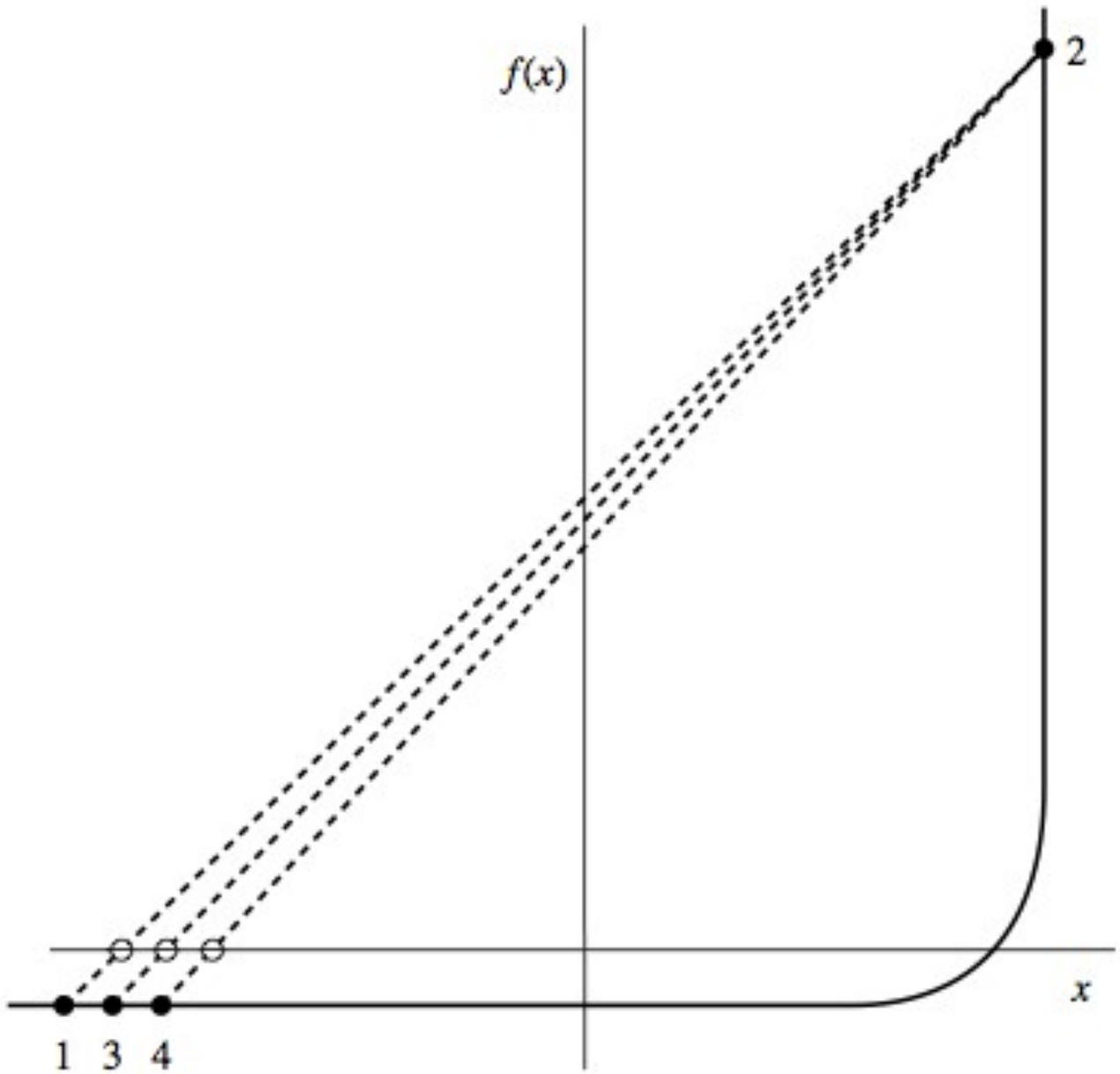
$$x_{i+1} = x_i \cdot \frac{y_{i-1}}{y_{i-1} - y_i} + x_{i-1} \cdot \frac{y_i}{y_i - y_{i-1}}$$

$$x_{i+1} = x_i \cdot \frac{y_{i-1}}{y_{i-1} - y_i} + x_{i-1} \cdot \frac{y_i}{y_i - y_{i-1}}$$

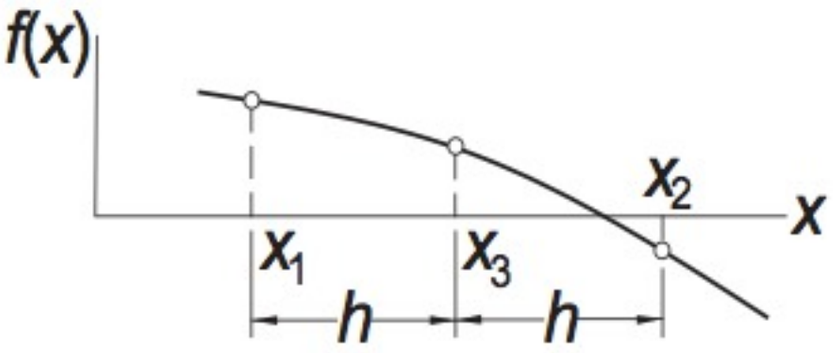
The false position method (also known as *regula falsi*) requires x_1 and x_2 to bracket the root. After the improved root is computed from Eq above either x_1 or x_2 is replaced by x_3. If f_3 has the same sign as f_1, we let $x_1 \leftarrow x_3$; otherwise we choose $x_2 \leftarrow x_3$. In this manner, the root is always bracketed in (x_1, x_2). The procedure is then repeated until convergence is obtained.

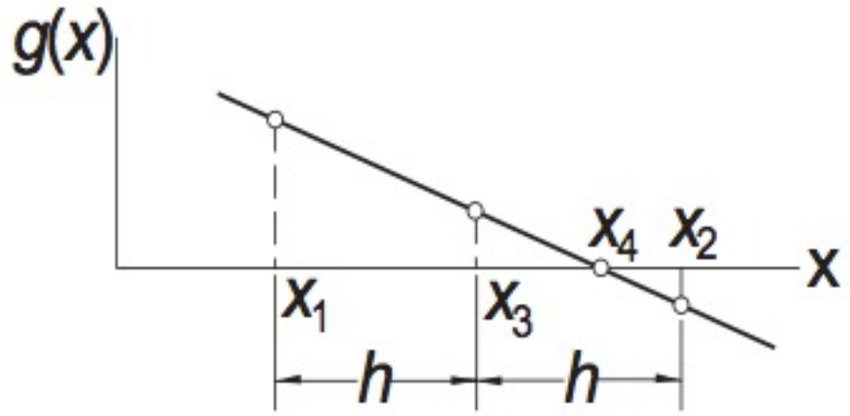
The secant method differs from the false position method in two details. (1) It does not require prior bracketing of the root; and (2) the oldest prior estimate of the root is discarded; that is, after x_3 is computed, we let $x_1 \leftarrow x_2$, $x_2 \leftarrow x_3$.

Failure of secant methods



Ridder's method





Ridder's Method

Ridder's method is a clever modification of the false position method. Assuming that the root is bracketed in (x_1, x_2), we first compute $f_3 = f(x_3)$, where x_3 is the midpoint of the bracket, as indicated in Fig. above Next, we the introduce the function

$$g(x) = f(x)e^{(x-x_1)Q}$$

where the constant Q is determined by requiring the points (x_1, g_1), (x_2, g_2), and (x_3, g_3) to lie on a straight line, as shown in Fig. above As before, the notation we use is $g_i = g(x_i)$. The improved value of the root is then obtained by linear interpolation of $g(x)$ rather than $f(x)$.

Let us now look at the details. From Eq. (a) we obtain

$$g_1 = f_1 \qquad g_2 = f_2 e^{2hQ} \qquad g_3 = f_3 e^{hQ}$$

where $h = (x_2 - x_1)/2$. The requirement that the three points in Fig. 4.3b lie on a straight line is $g_3 = (g_1 + g_2)/2$, or

$$f_3 e^{hQ} = \frac{1}{2}(f_1 + f_2 e^{2hQ})$$

which is a quadratic equation in e^{hQ}. The solution is

$$e^{hQ} = \frac{f_3 \pm \sqrt{f_3^2 - f_1 f_2}}{f_2} \qquad \text{(c)}$$

Linear interpolation based on points (x_1, g_1) and (x_3, g_3) now yields for the improved root

$$x_4 = x_3 - g_3 \frac{x_3 - x_1}{g_3 - g_1} = x_3 - f_3 e^{hQ} \frac{x_3 - x_1}{f_3 e^{hQ} - f_1}$$

where in the last step we utilized Eqs. (b). As the final step, we substitute e^{hQ} from Eq. (c), and obtain after some algebra

$$x_4 = x_3 \pm (x_3 - x_1) \frac{f_3}{\sqrt{f_3^2 - f_1 f_2}} \qquad (4.3)$$

It can be shown that the correct result is obtained by choosing the plus sign if $f_1 - f_2 > 0$, and the minus sign if $f_1 - f_2 < 0$. After the computation of x_4, new brackets are determined for the root and Eq. (4.3) is applied again. The procedure is repeated until the difference between two successive values of x_4 becomes negligible.

ridder method

```
function root = ridder(func,x1,x2,tol)
% Ridder's method for computing the root of f(x) = 0
% USAGE: root = ridder(func,a,b,tol)
% INPUT:
% func = handle of function that returns f(x).
% x1,x2  = limits of the interval containing the root.
% tol  = error tolerance (default is 1.0e6*eps).
% OUTPUT:
% root = zero of f(x) (root = NaN if failed to converge).

if nargin < 4; tol = 1.0e6*eps; end
f1 = func(x1);
if f1 == 0; root = x1; return; end
f2 = func(x2);
if f2 == 0; root = x2; return; end
if f1*f2 > 0
    error('Root is not bracketed in (a,b)')
end
```

```
for i = 0:30
    % Compute improved root from Ridder's formula
    x3 = 0.5*(x1 + x2); f3 = func(x3);
    if f3 == 0; root = x3; return; end
    s = sqrt(f3^2 - f1*f2);
    if s == 0; root = NaN; return; end
    dx = (x3 - x1)*f3/s;
    if (f1 - f2) < 0; dx = -dx; end
    x4 = x3 + dx; f4 = func(x4);
    % Test for convergence
    if i > 0;
        if abs(x4 - xOld) < tol*max(abs(x4),1.0)
            root = x4; return
        end
    end
    xOld = x4;
    % Re-bracket the root
    if f3*f4 > 0
        if f1*f4 < 0; x2 = x4; f2 = f4;
        else          x1 = x4; f1 = f4;
        end
    else
            x1 = x3; x2 = x4; f1 = f3; f2 = f4;
    end
end
root = NaN;
```

Example

Determine the root of $f(x) = x^3 - 10x^2 + 5 = 0$ that lies in $(0.6, 0.8)$ with Ridder's method.

Solution The starting points are

$$x_1 = 0.6 \qquad f_1 = 0.6^3 - 10(0.6)^2 + 5 = 1.6160$$

$$x_2 = 0.8 \qquad f_2 = 0.8^3 - 10(0.8)^2 + 5 = -0.8880$$

First iteration Bisection yields the point

$$x_3 = 0.7 \qquad f_3 = 0.7^3 - 10(0.7)^2 + 5 = 0.4430$$

The improved estimate of the root can now be computed with Ridder's formula:

$$s = \sqrt{f_3^2 - f_1 f_2} = \sqrt{0.4330^2 - 1.6160(-0.8880)} = 1.2738$$

$$x_4 = x_3 \pm (x_3 - x_1)\frac{f_3}{s}$$

Because $f_1 > f_2$ we must use the plus sign. Therefore,

$$x_4 = 0.7 + (0.7 - 0.6)\frac{0.4430}{1.2738} = 0.7348$$

$$f_4 = 0.7348^3 - 10(0.7348)^2 + 5 = -0.0026$$

As the root clearly lies in the interval (x_3, x_4), we let

$$x_1 \leftarrow x_3 = 0.7 \qquad f_1 \leftarrow f_3 = 0.4430$$

$$x_2 \leftarrow x_4 = 0.7348 \qquad f_2 \leftarrow f_4 = -0.0026$$

which are the starting points for the next iteration.

Second iteration

$$x_3 = 0.5(x_1 + x_2) = 0.5(0.7 + 0.7348) = 0.7174$$

$$f_3 = 0.7174^3 - 10(0.7174)^2 + 5 = 0.2226$$

$$s = \sqrt{f_3^2 - f_1 f_2} = \sqrt{0.2226^2 - 0.4430(-0.0026)} = 0.2252$$

$$x_4 = x_3 \pm (x_3 - x_1)\frac{f_3}{s}$$

Since $f_1 > f_2$ we again use the plus sign, so that

$$x_4 = 0.7174 + (0.7174 - 0.7)\frac{0.2226}{0.2252} = 0.7346$$

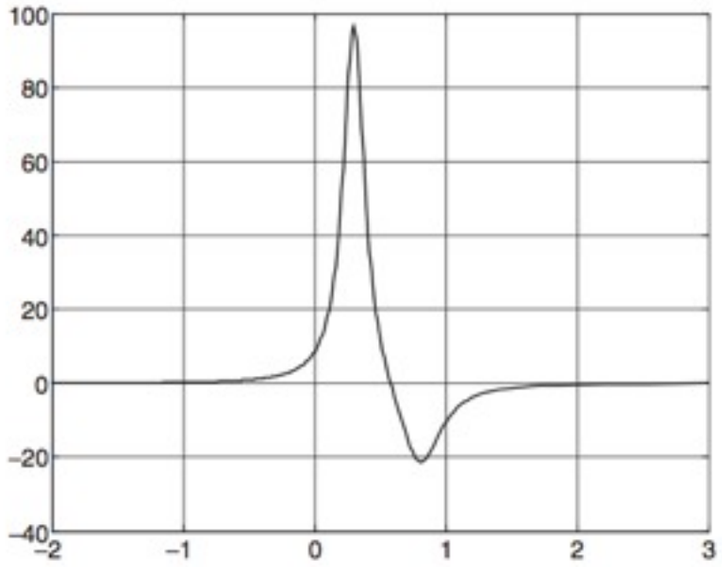
$$f_4 = 0.7346^3 - 10(0.7346)^2 + 5 = 0.0000$$

Thus the root is $x = 0.7346$, accurate to at least four decimal places.

Example

Compute the zero of the function

$$f(x) = \frac{1}{(x-0.3)^2 + 0.01} - \frac{1}{(x-0.8)^2 + 0.04}$$



Solution The M-file for the function is

```
function y = fex4_5(x)
% Function used in Example 4.5
y = 1/((x - 0.3)^2 + 0.01)...
  - 1/((x - 0.8)^2 + 0.04);
```

We obtain the approximate location of the root by plotting the function. The following commands produce the plot shown:

```
>> fplot(@fex4_5,[-2,3])
>> grid on
```

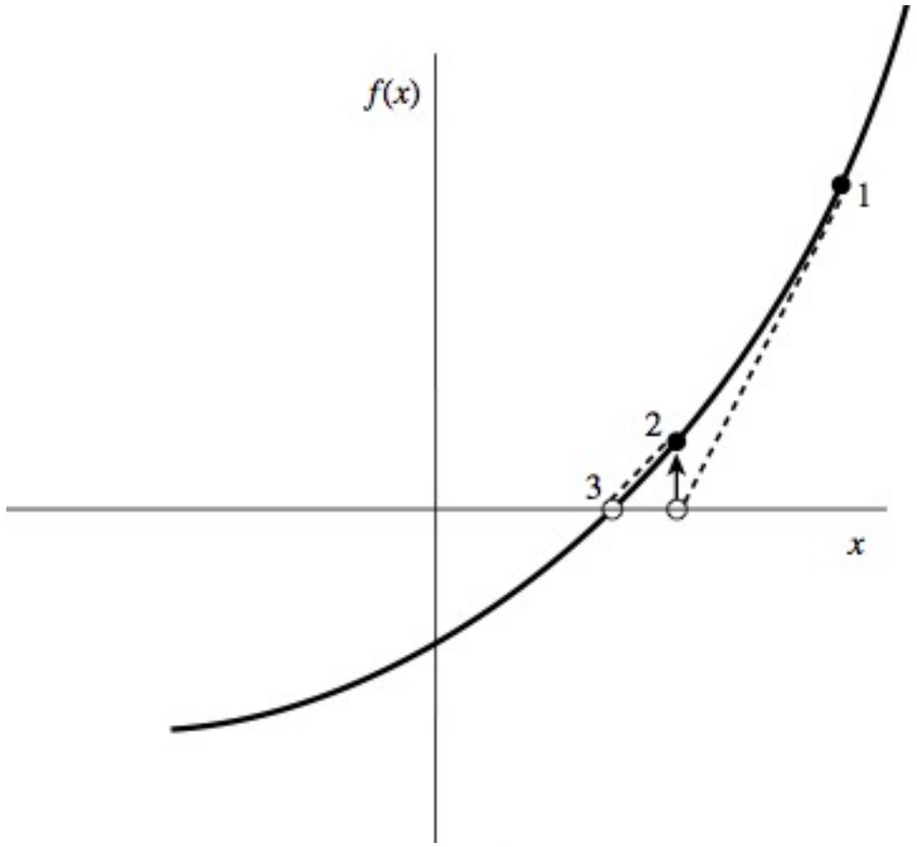
It is evident that the root of $f(x) = 0$ lies between $x = 0.5$ and 0.7. We can extract this root with the command

```
>> ridder(@fex4_5,0.5,0.7)
```

The result, which required four iterations, is

```
ans =
 0.5800
```

Newton's Method

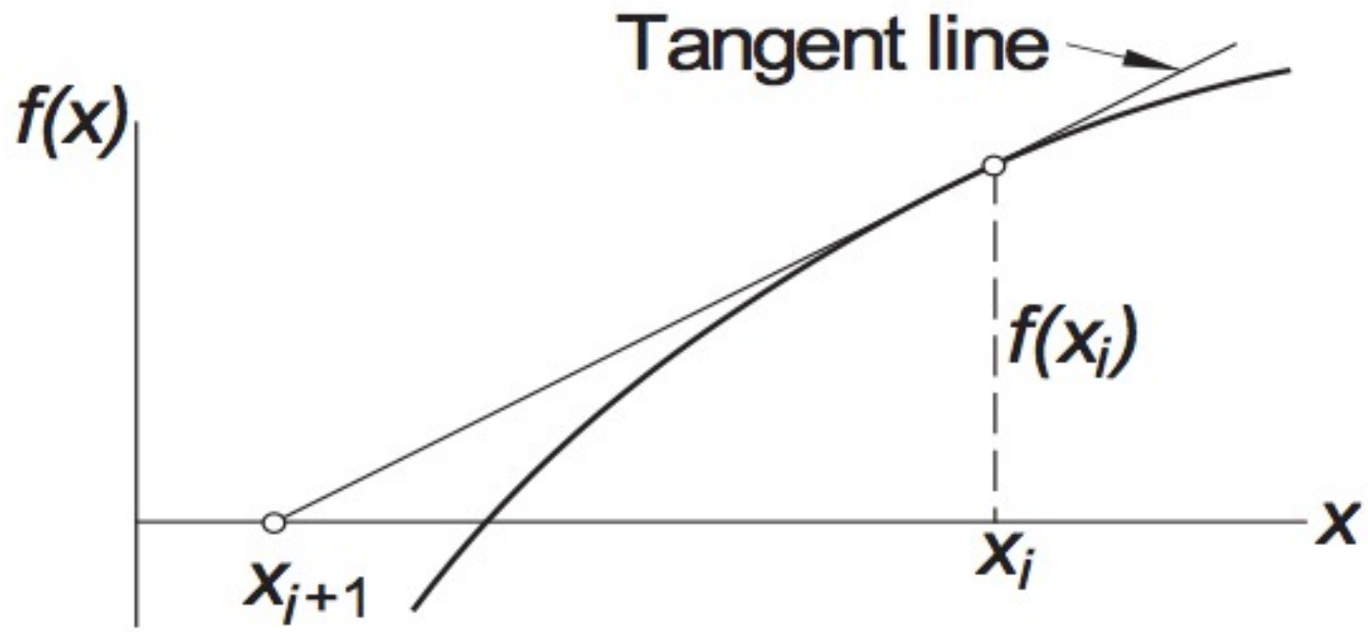


$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2$$

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

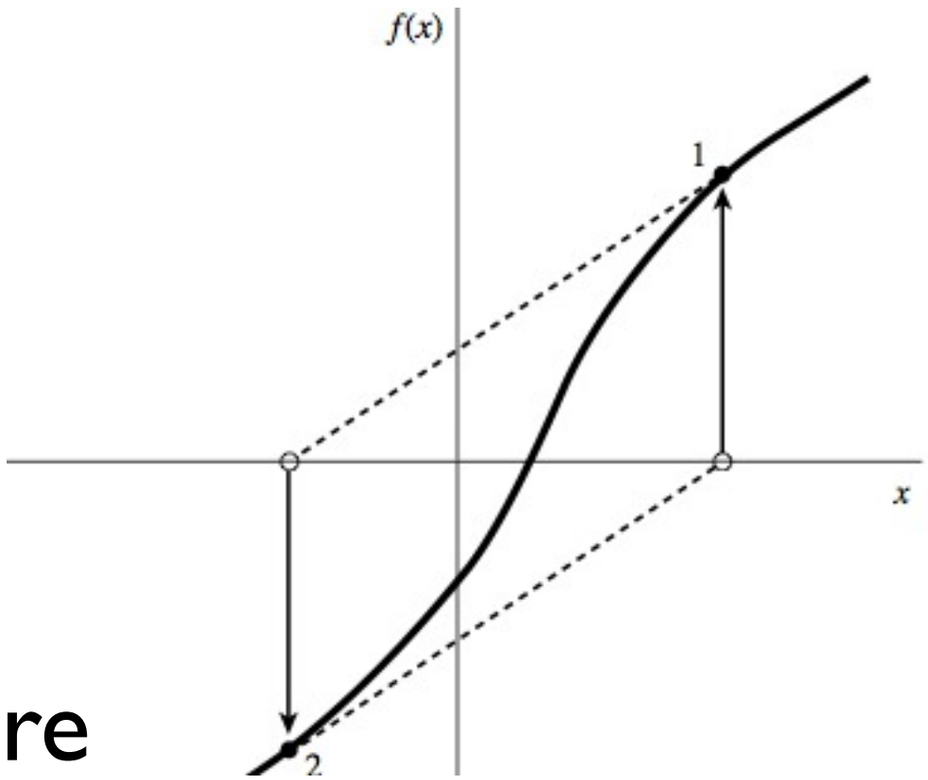
$$E_{i+1} = -\frac{f''(x_i)}{2f'(x_i)} E_i^2$$

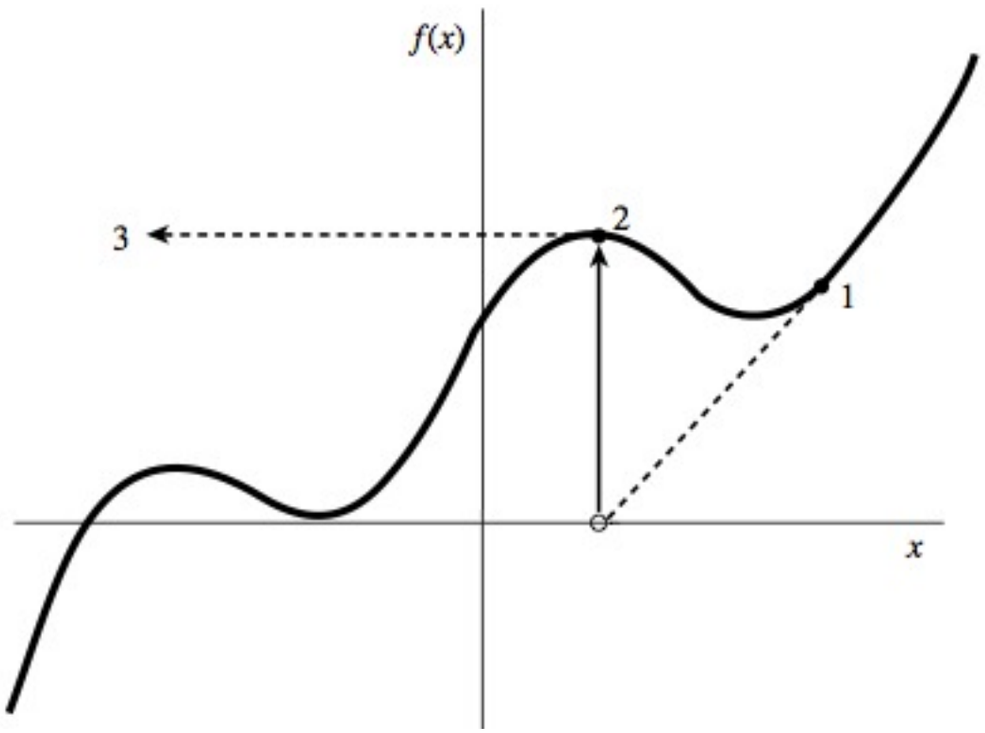
Newton's Method Algorithm



1. Let x be a guess for the root of $f(x) = 0$.
2. Compute $\Delta x = -f(x)/f'(x)$.
3. Let $x \leftarrow x + \Delta x$ and repeat steps 2–3 until $|\Delta x| < \varepsilon$.

Failure





■ newtonRaphson

The following *safe version* of the Newton–Raphson method assumes that the root to be computed is initially bracketed in `(a,b)`. The midpoint of the bracket is used as the initial guess of the root. The brackets are updated after each iteration. If a Newton–Raphson iteration does not stay within the brackets, it is disregarded and replaced with bisection. Since `newtonRaphson` uses the function `f(x)` as well as its derivative, function routines for both (denoted by `func` and `dfunc` in the listing) must be provided by the user.

```
% finding a root of f(x) = 0.
% USAGE: root = newtonRaphson(func,dfunc,a,b,tol)
% INPUT:
% func  = handle of function that returns f(x).
% dfunc = handle of function that returns f'(x).
% a,b   = brackets (limits) of the root.
% tol   = error tolerance (default is 1.0e6*eps).
% OUTPUT:
% root = zero of f(x) (root = NaN if no convergence).
```

```
if nargin < 5; tol = 1.0e6*eps; end
fa = feval(func,a); fb = feval(func,b);
if fa == 0; root = a; return; end
if fb == 0; root = b; return; end
if fa*fb > 0.0
    error('Root is not bracketed in (a,b)')
end
x = (a + b)/2.0;
for i = 1:30
    fx = feval(func,x);
    if abs(fx) < tol; root = x; return; end
    % Tighten brackets on the root
    if fa*fx < 0.0; b = x;
    else; a = x;
    end
    % Try Newton-Raphson step
    dfx = feval(dfunc,x);
    if abs(dfx) == 0; dx = b - a;
    else; dx = -fx/dfx;
    end
    x = x + dx;
    % If x not in bracket, use bisection
    if (b - x)*(x - a) < 0.0
        dx = (b - a)/2.0;
        x = a + dx;
    end
    % Check for convergence
    if abs(dx) < tol*max(b,1.0)
        root = x; return
    end
end
root = NaN
```