

# Solution to Nonlinear Equations

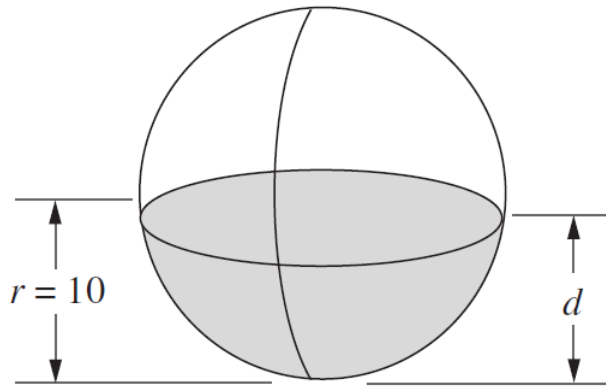
Peng Yu

Tel: 0755 8801 8911

Email: [yup6@sustech.edu.cn](mailto:yup6@sustech.edu.cn)

# Solution of nonlinear equations $f(x)=0$

An example



**Figure 2.1** The portion of a sphere of radius  $r$  that is to be submerged to a depth  $d$ .

$$M_w = \rho_{water} \int_0^d \pi(r^2 - (x - r)^2) dx = \rho_{water} \frac{\pi d^2(3r - d)}{3},$$

$$M_b = 4\pi r^3 \rho / 3.$$

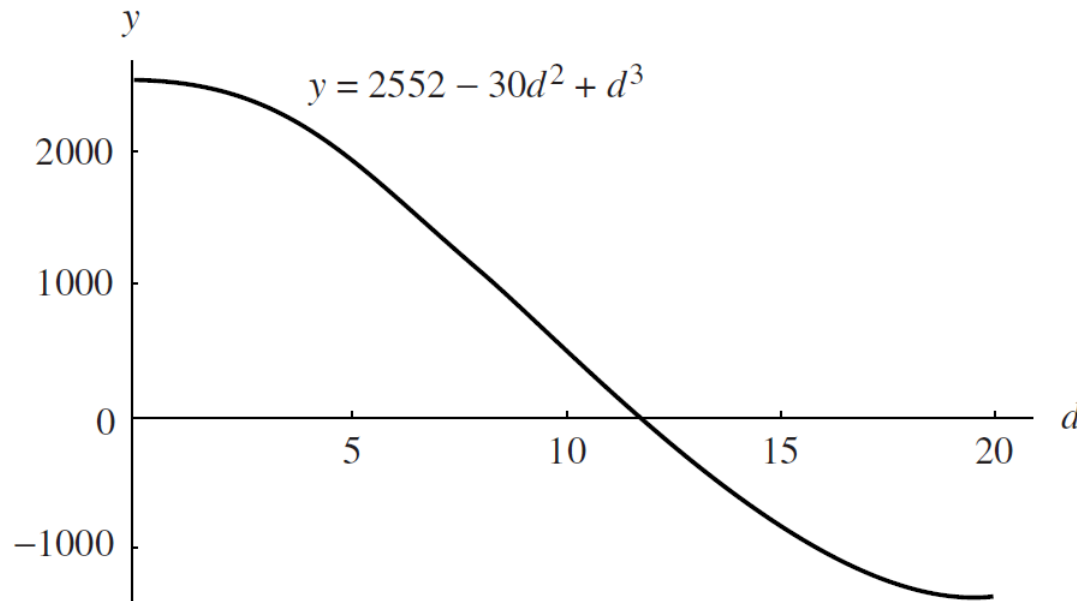
Applying Archimedes' law,  $M_w = M_b$ ,

$$\frac{\pi(d^3 \rho_{water} - 3d^2 r \rho_{water} + 4r^3 \rho)}{3} = 0.$$

An example:  $r = 10$ ,  $\rho = 0.638$ ,  $\rho_{\text{water}} = 1$

$$\frac{\pi (2552 - 30d^2 + d^3)}{3} = 0.$$

$$y = 2552 - 30d^2 + d^3$$



**Figure 2.2** The cubic  $y = 2552 - 30d^2 + d^3$ .

$$d_1 = -8.17607212, \quad d_2 = 11.86150151, \quad d_3 = 26.31457061$$

# Solution to Nonlinear Equations

- Iteration for Solving  $x = g(x)$
- Bracketing Methods for Locating a Root
- Initial Approximation and Convergence Criteria
- Newton-Raphson and Secant Methods
- Aitken's Process and Steffensen's and Muller's Methods

**Iteration for Solving  $x = g(x)$**

# Iteration

- A fundamental principle in computer science is iteration, i.e., a process is repeated until an answer is achieved.

$p_0$  (starting value)

$$p_1 = g(p_0)$$

$$p_2 = g(p_1)$$

$\vdots$

$$p_k = g(p_{k-1})$$

$$p_{k+1} = g(p_k)$$

$\vdots$

- A sequence of numbers
- If the numbers tend to a limit,  $p = g(p)$
- What if the numbers diverge or are periodic?

# Divergent iteration

**Example** The iterative rule  $p_0 = 1$  and  $p_{k+1} = 1.001p_k$  for  $k = 0, 1, \dots$  produces a divergent sequence.

The first 100 terms look as follows:

$$\begin{aligned} p_1 &= 1.001p_0 = (1.001)(1.000000) = 1.001000, \\ p_2 &= 1.001p_1 = (1.001)(1.001000) = 1.002001, \\ p_3 &= 1.001p_2 = (1.001)(1.002001) = 1.003003, \\ &\quad \vdots \qquad \qquad \qquad \vdots \qquad \qquad \qquad \vdots \\ p_{100} &= 1.001p_{99} = (1.001)(1.104012) = 1.105116. \end{aligned}$$

The process can be continued indefinitely, and  $\lim_{n \rightarrow \infty} p_n = +\infty$ .

The sequence  $\{p_k\}$  is a numerical solution to the differential equation  $y' = 0.001y$ . The solution is known to be  $y(x) = e^{0.001x}$ .

Indeed, if we compare the 100<sup>th</sup> term in the sequence with  $y(100)$ , we see that  $p_{100} = 1.105116 \approx 1.105171 = e^{0.1} = y(100)$ .

# Finding fixed points

- **Definition 2.1.** A **fixed point** of a function  $g(x)$  is a real number  $P$  such that  $P = g(P)$ .
- Geometrically, the fixed points of a function  $y = g(x)$  are the points of intersection of  $y = g(x)$  and  $y = x$ .
- **Definition 2.2.** The iteration  $p_{n+1} = g(p_n)$  for  $n = 0, 1, \dots$  is called **fixed-point iteration**.
- **Theorem 2.1.** Assume that  $g$  is a continuous function and that  $\{p_n\}$  is a sequence generated by fixed-point iteration. If  $\lim_{n \rightarrow \infty} p_n = P$ , then  $P$  is a fixed point of  $g(x)$ .



# An example

**Example** Consider the convergent iteration

$$p_0 = 0.5 \quad \text{and} \quad p_{k+1} = e^{-p_k} \quad \text{for } k = 0, 1, \dots$$

The first 10 terms are obtained by the calculations

$$\begin{aligned} p_1 &= e^{-0.500000} = 0.606531 \\ p_2 &= e^{-0.606531} = 0.545239 \\ p_3 &= e^{-0.545239} = 0.579703 \\ &\vdots \\ p_9 &= e^{-0.566409} = 0.567560 \\ p_{10} &= e^{-0.567560} = 0.566907 \end{aligned}$$

The sequence is converging, and further calculation reveal that

$$\lim_{n \rightarrow \infty} p_n = 0.567143 \dots$$

Thus, we found an approximation for the fixed point of the function  $y = e^{-x}$ .

# When does a fixed point exist

**Theorem 2.2.** Assume that  $g \in C[a, b]$ .

(a) If the range of the mapping  $y = g(x)$  satisfies  $y \in [a, b]$  for all  $x \in [a, b]$ , then  $g$  has a fixed point in  $[a, b]$ .

(b) Furthermore, suppose that  $g'(x)$  is defined over  $(a, b)$  and that a positive constant  $K < 1$  exists with  $|g'(x)| \leq K < 1$  for all  $x \in [a, b]$ ; then  $g$  has a unique fixed point  $P$  in  $[a, b]$ .

**Proof of (a).** If  $g(a) = a$  or  $g(b) = b$ , the assertion is true. Otherwise, the values of  $g(a)$  and  $g(b)$  must satisfy  $g(a) \in (a, b]$  and  $g(b) \in [a, b)$ . The function  $f(x) \equiv x - g(x)$  has a property that

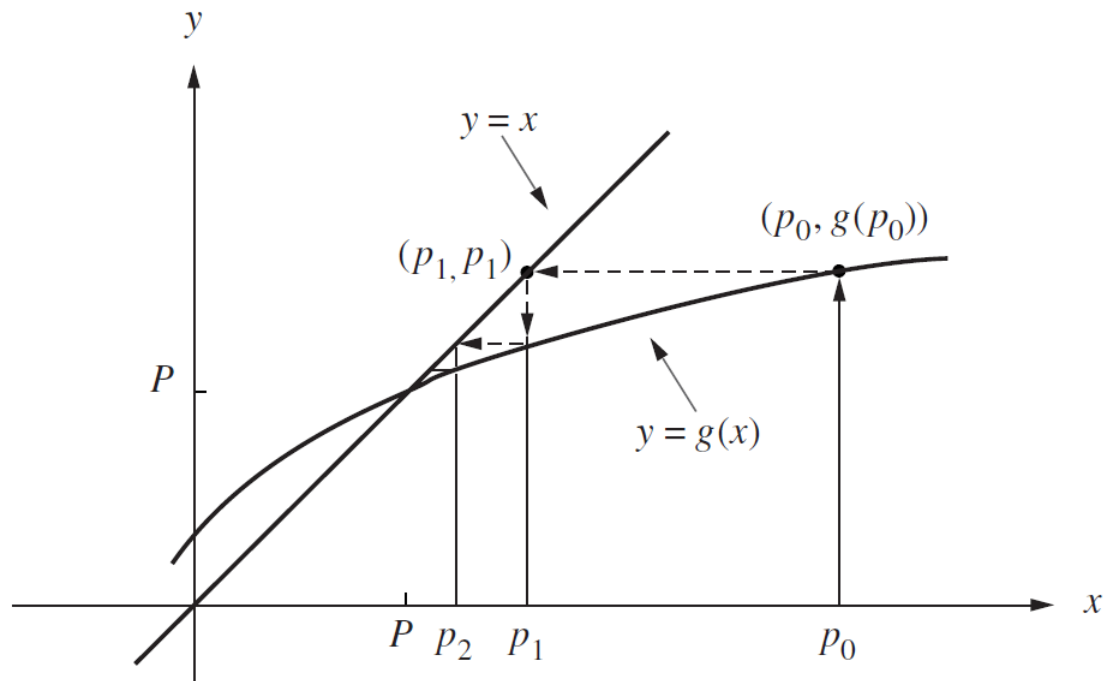
$$f(a) = a - g(a) < 0 \quad \text{and} \quad f(b) = b - g(b) > 0.$$

Now apply Theorem 1.2, **the intermediate value theorem**, to  $f(x)$ , with the constant  $L = 0$ , and conclude that there exists a number  $P$  with  $P \in (a, b)$  so that  $f(P) = 0$ . Therefore,  $P = g(P)$  and  $P$  is the desired fixed point of  $g(x)$ .

# Fixed-point theorem

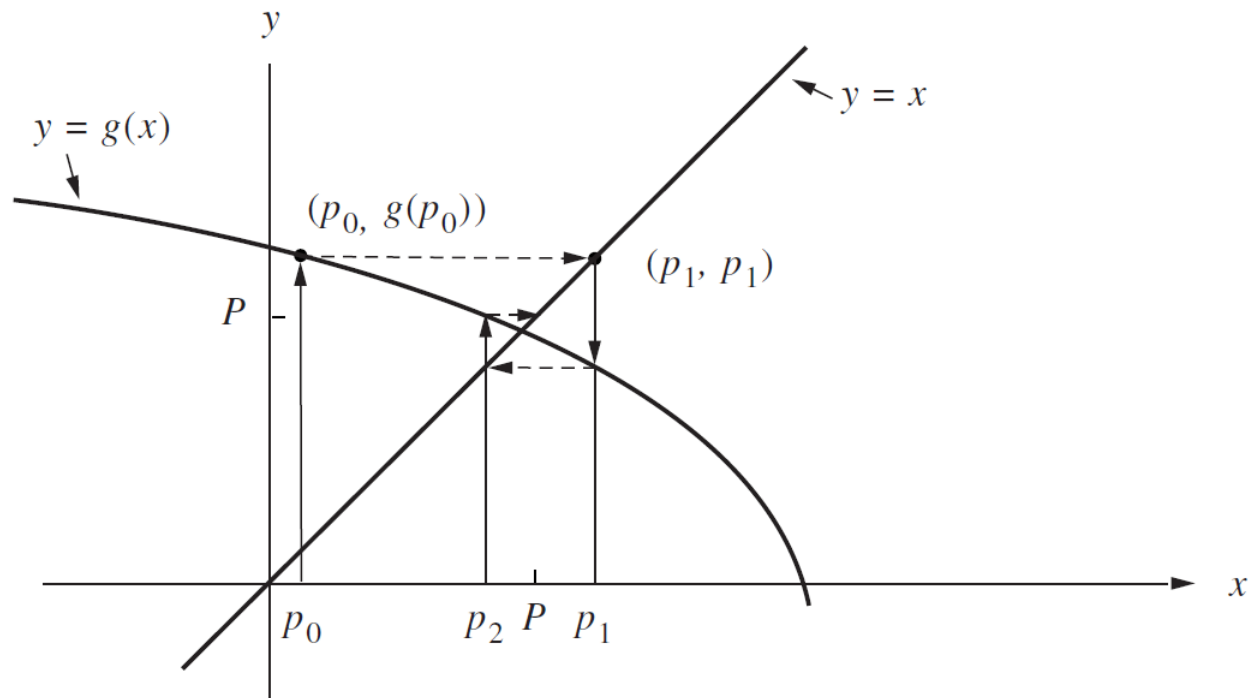
**Theorem 2.3 (Fixed-Point Theorem).** Assume that (i)  $g, g' \in C[a, b]$ , (ii)  $K$  is a positive constant, (iii)  $p_0 \in (a, b)$ , and (iv)  $g(x) \in [a, b]$  for all  $x \in [a, b]$ .

If  $|g'(x)| \leq K < 1$  for all  $x \in [a, b]$ , then the iteration  $p_n = g(p_{n-1})$  will converge to the unique fixed point  $P \in [a, b]$ . In this case,  $P$  is said to be an attractive fixed point.



**Figure 2.4** (a) Monotone convergence when  $0 < g'(P) < 1$ .

# Fixed-point theorem

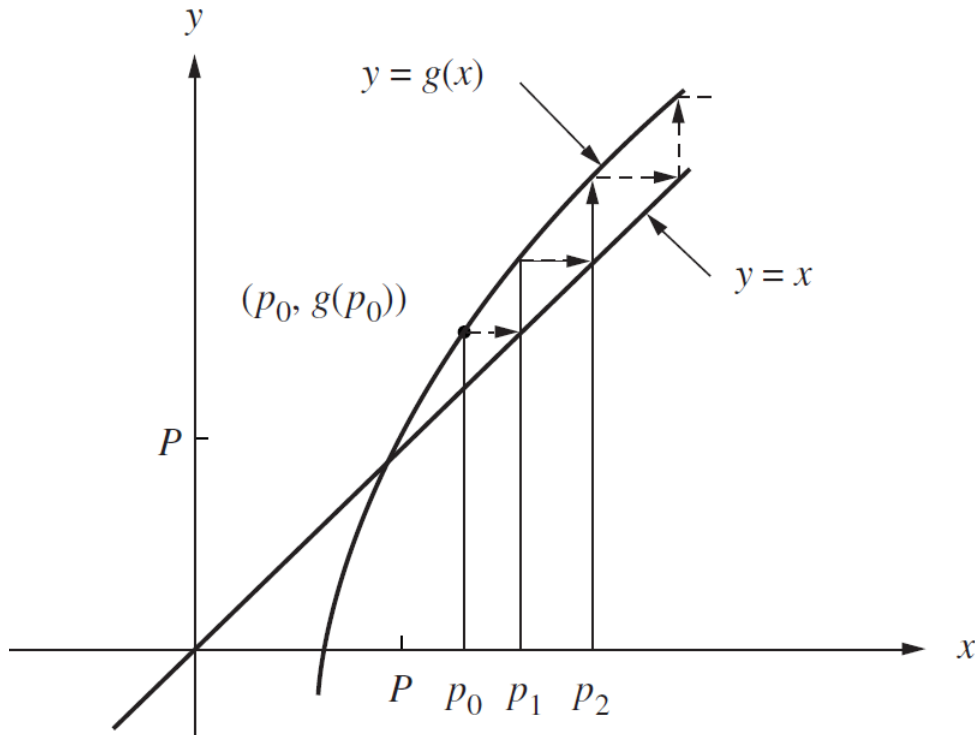


**Figure 2.4** (b) Oscillating convergence when  $-1 < g'(P) < 0$ .

# Fixed-point theorem

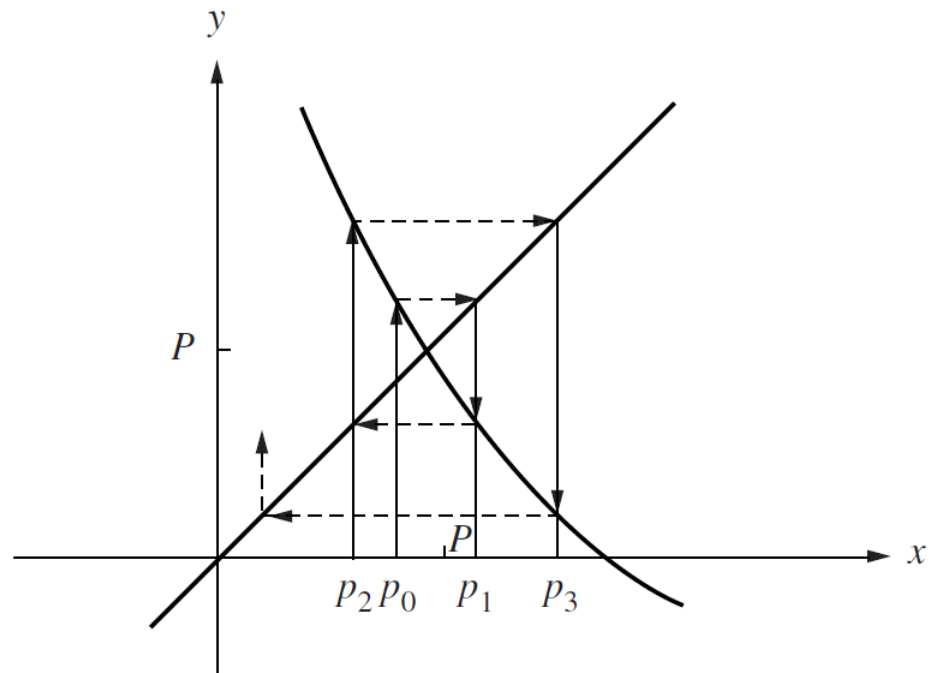
**Theorem 2.3 (Fixed-Point Theorem).** Assume that (i)  $g, g' \in C[a, b]$ , (ii)  $K$  is a positive constant, (iii)  $p_0 \in (a, b)$ , and (iv)  $g(x) \in [a, b]$  for all  $x \in [a, b]$ .

$|g'(x)| > 1$  for all  $x \in [a, b]$ , then the iteration  $p_n = g(p_{n-1})$  will not converge to  $P$ . In this case,  $P$  is said to be a repelling fixed point and the iteration exhibits local divergence.



**Figure 2.5** (a) Monotone divergence when  $1 < g'(P)$ .

# Fixed-point theorem



**Figure 2.5** (b) Divergent oscillation when  $g'(P) < -1$ .

# An example

Consider the iteration  $p_{n+1} = g(p_n)$  when the function  $g(x) = 1 + x - x^2/4$ . The fixed points can be found by solving the equation  $x = g(x)$ . The two solutions (fixed points of  $g$ ) are  $x = -2$  and  $x = 2$ . The derivative of the function is  $g'(x) = 1 - x/2$ , and there are only two cases to consider.

*Case(i):*  $P = -2$   
Start with  $p_0 = -2.05$   
then get  $p_1 = -2.100625$   
 $p_2 = -2.20378135$   
 $p_3 = -2.41794441$   
 $\vdots$   
 $\lim_{n \rightarrow \infty} p_n = -\infty.$

Since  $|g'(x)| > \frac{3}{2}$  on  $[-3, -1]$ , by Theorem 2.3, the sequence will not converge to  $P = -2$ .

*Case(ii):*  $P = 2$   
Start with  $p_0 = 1.6$   
then get  $p_1 = 1.96$   
 $p_2 = 1.9996$   
 $p_3 = 1.99999996$   
 $\vdots$   
 $\lim_{n \rightarrow \infty} p_n = 2.$

Since  $|g'(x)| < \frac{1}{2}$  on  $[1, 3]$ , by Theorem 2.3, the sequence will converge to  $P = 2$ .

What will happen when  $g'(P)=1$ ?

# Another example

Consider the iteration  $p_{n+1} = g(p_n)$  when the function  $g(x) = 2(x - 1)^{1/2}$  for  $x \geq 1$ . Only one fixed point  $P = 2$  exists. The derivative is  $g'(x) = 1/(x - 1)^{1/2}$  and  $g'(2) = 1$ , so Theorem 2.3 does not apply. There are two cases to consider when the starting value lies to the left or right of  $P = 2$ .

*Case(i):* Start with  $p_0 = 1.5$ ,  
then get  $p_1 = 1.41421356$   
 $p_2 = 1.28718851$   
 $p_3 = 1.07179943$   
 $p_4 = 0.53590832$   
 $\vdots$   
 $p_5 = 2(-0.46409168)^{1/2}.$

Since  $p_4$  lies outside the domain of  $g(x)$ , the term  $p_5$  cannot be computed.

*Case(ii):* Start with  $p_0 = 2.5$ ,  
then get  $p_1 = 2.44948974$   
 $p_2 = 2.40789513$   
 $p_3 = 2.37309514$   
 $p_4 = 2.34358284$   
 $\vdots$   
 $\lim_{n \rightarrow \infty} p_n = 2.$

This sequence is converging too slowly to the value  $P = 2$ , indeed,  $P_{1000} = 2.00398714$



# Absolute and relative error consideration

- In last example, case (ii), the sequence converges slowly, and after 1000 iterations

$$p_{1000} = 2.00398714 \quad p_{1001} = 2.00398317, \quad \text{and} \quad p_{1002} = 2.00397912$$

- Probably find convergence after a few thousand more iterations.
- What about a criterion for stopping the iteration?
- Difference between two consecutive terms:

$$|p_{1001} - p_{1002}| = |2.00398317 - 2.00397912| = 0.00000396.$$

- Absolute error:

$$|P - p_{1000}| = |2.00000000 - 2.00398714| = 0.00398714.$$

# Algorithm

**Program 2.1 (Fixed-Point Iteration).** To approximate a solution to the equation  $x = g(x)$  starting with the initial guess  $p_0$  and iterating  $p_{n+1} = g(p_n)$ .

```
function [k,p,err,P]=fixpt(g,p0,tol,max1)
% Input - g is the iteration function input as a string 'g'
%        - p0 is the initial guess for the fixed point
%        - tol is the tolerance
%        - max1 is the maximum number of iterations
%Output - k is the number of iterations that were carried out
%        - p is the approximation to the fixed point
%        - err is the error in the approximation
%        - P contains the sequence {pn}
P(1)= p0;
for k=2:max1
    P(k)=feval(g,P(k-1));
    err=abs(P(k)-P(k-1));
    relerr=err/(abs(P(k))+eps);
    p=P(k);
    if (err<tol) | (relerr<tol),break;end
end
if k == max1
    disp('maximum number of iterations exceeded')
end
P=P';
```

*Remark.* When using the user-defined function `fixpt`, it is necessary to input the M-file `g.m` as a string: `'g'` (see the Appendix).

# **Bracketing Methods for Locating a Root**

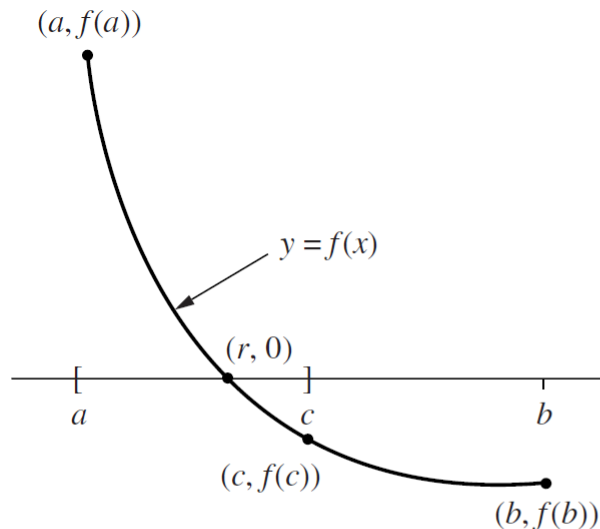
# Root of Equation

- **Definition 2.3.** Assume that  $f(x)$  is a continuous function. Any number  $r$  for which  $f(r) = 0$  is called a root of the equation  $f(x) = 0$ . Also, we say that  $r$  is *a zero of the function  $f(x)$* .
- **Example:**

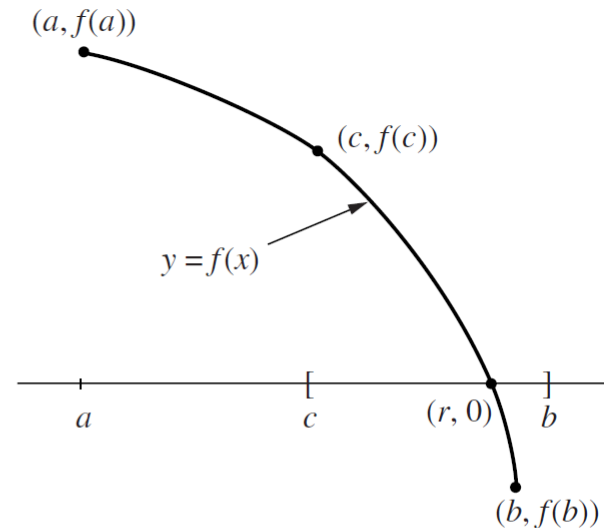
The equation  $2x^2 + 5x - 3 = 0$  has two real roots  $r_1 = 0.5$  and  $r_2 = -3$ , whereas the corresponding function  $f(x) = 2x^2 + 5x - 3 = (2x - 1)(x + 3)$  has two real zeros,  $r_1 = 0.5$  and  $r_2 = -3$ .

# Bisection method of Bolzano

- Consider a function  $f(x)$  which is continuous at interval  $[a, b]$ , and  $f(a)$  and  $f(b)$  have opposite signs
- The bisection method systematically moves the endpoints of the interval closer and closer together until we obtain an interval of arbitrarily small width that brackets the zero.



(a) If  $f(a)$  and  $f(c)$  have opposite signs, then squeeze from the right.



(b) If  $f(c)$  and  $f(b)$  have opposite signs, then squeeze from the left.

**Figure 2.6** The decision process for the bisection process.

# Bisection theorem

$$(8) \quad a_0 \leq a_1 \leq \cdots \leq a_n \leq \cdots \leq r \leq \cdots \leq b_n \leq \cdots \leq b_1 \leq b_0,$$

where  $c_n = (a_n + b_n)/2$ , and if  $f(a_{n+1})f(b_{n+1}) < 0$ , then

$$(9) \quad [a_{n+1}, b_{n+1}] = [a_n, c_n] \text{ or } [a_{n+1}, b_{n+1}] = [c_n, b_n] \text{ for all } n.$$

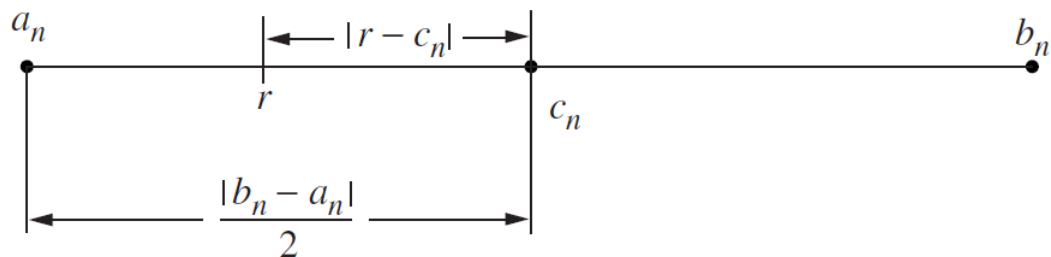
**Theorem 2.4 (Bisection Theorem).** Assume that  $f \in C[a, b]$  and that there exists a number  $r \in [a, b]$  such that  $f(r) = 0$ . If  $f(a)$  and  $f(b)$  have opposite signs, and  $\{c_n\}_{n=0}^{\infty}$  represents the sequence of midpoints generated by the bisection process of (8) and (9), then

$$(10) \quad |r - c_n| \leq \frac{b - a}{2^{n+1}} \quad \text{for } n = 0, 1, \dots,$$

and therefore the sequence  $\{c_n\}_{n=0}^{\infty}$  converges to the zero  $x = r$ ; that is,

$$(11) \quad \lim_{n \rightarrow \infty} c_n = r.$$

# Bisection theorem



**Figure 2.7** The root  $r$  and midpoint  $c_n$  of  $[a_n, b_n]$  for the bisection method.

$$|r - c_n| \leq \frac{b_n - a_n}{2} \quad \text{for all } n.$$

# An example

**Table 2.1** Bisection Method Solution of  $x \sin(x) - 1 = 0$

$k$	Left endpoint, $a_k$	Midpoint, $c_k$	Right endpoint, $b_k$	Function value, $f(c_k)$
0	0	1.	2.	−0.158529
1	1.0	1.5	2.0	0.496242
2	1.00	1.25	1.50	0.186231
3	1.000	1.125	1.250	0.015051
4	1.0000	1.0625	1.1250	−0.071827
5	1.06250	1.09375	1.12500	−0.028362
6	1.093750	1.109375	1.125000	−0.006643
7	1.1093750	1.1171875	1.1250000	0.004208
8	1.10937500	1.11328125	1.11718750	−0.001216
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Converges to  $r = 1.114157141\dots$



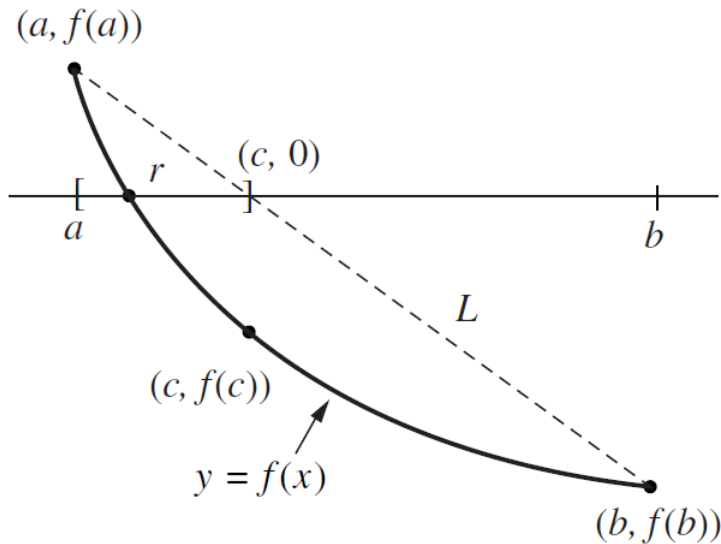
# Algorithm

**Program 2.2 (Bisection Method).** To approximate a root of the equation  $f(x) = 0$  in the interval  $[a, b]$ . Proceed with the method only if  $f(x)$  is continuous and  $f(a)$  and  $f(b)$  have opposite signs.

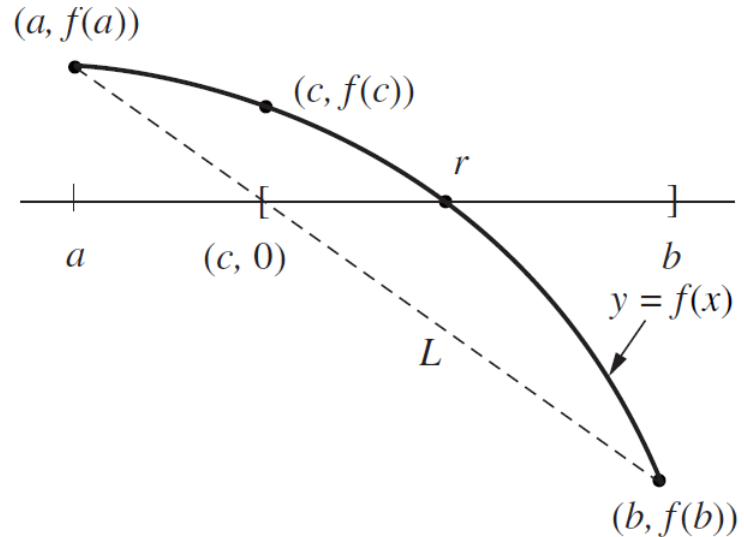
```
function [c,err,yc]=bisect(f,a,b,delta)
%Input  - f is the function input as a string 'f'
%        - a and b are the left and right endpoints
%        - delta is the tolerance
%Output - c is the zero
%        - yc=f(c)
%        - err is the error estimate for c
ya=feval(f,a);
yb=feval(f,b);
if ya*yb>0,break,end
max1=1+round((log(b-a)-log(delta))/log(2));
for k=1:max1
    c=(a+b)/2;
    yc=feval(f,c);
    if yc==0
        a=c;
        b=c;
    elseif yb*yc>0
        b=c;
        yb=yc;
    else
        a=c;
        ya=yc;
    end
    if b-a < delta, break,end
end
c=(a+b)/2;
err=abs(b-a);
yc=feval(f,c);
```

# Method of false position

- Converges much faster



(a) If  $f(a)$  and  $f(c)$  have opposite signs, then squeeze from the right.



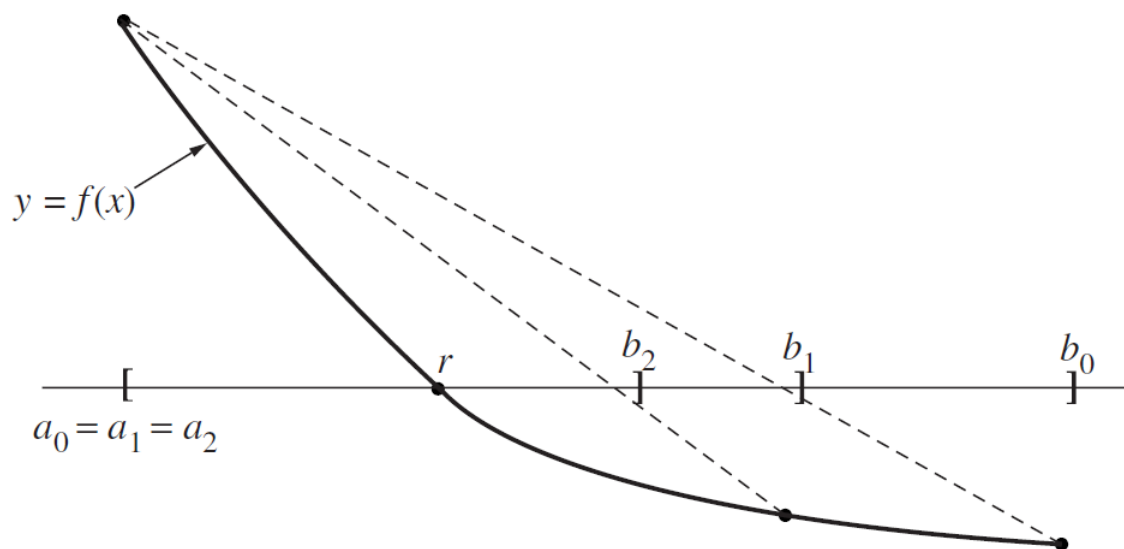
(b) If  $f(c)$  and  $f(b)$  have opposite signs, then squeeze from the left.

**Figure 2.8** The decision process for the false position method.

# Convergence of the False Position Method

$$c_n = b_n - \frac{f(b_n)(b_n - a_n)}{f(b_n) - f(a_n)}.$$

- Sequence  $c_n$  will converge to  $r$ .
- But beware; although the interval width  $b_n - a_n$  is getting smaller, it is possible that it may not go to zero.



**Figure 2.9** The stationary endpoint for the false position method.

# An example

**Table 2.2** False Position Method Solution of  $x \sin(x) - 1 = 0$

$k$	Left endpoint, $a_k$	Midpoint, $c_k$	Right endpoint, $b_k$	Function value, $f(c_k)$
0	0.00000000	1.09975017	2.00000000	-0.02001921
1	1.09975017	1.12124074	2.00000000	0.00983461
2	1.09975017	1.11416120	1.12124074	0.00000563
3	1.09975017	1.11415714	1.11416120	0.00000000

Converges to  $r = 1.11415714$  much faster

The termination criterion is different

# Algorithm

**Program 2.3 (False Position or Regula Falsi Method).** To approximate a root of the equation  $f(x) = 0$  in the interval  $[a, b]$ . Proceed with the method only if  $f(x)$  is continuous and  $f(a)$  and  $f(b)$  have opposite signs.

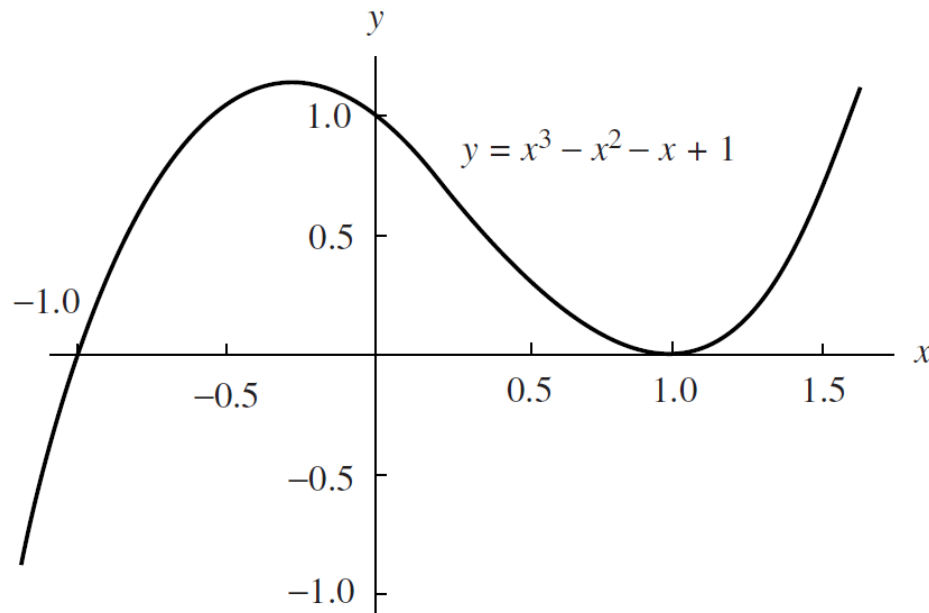
```
function [c,err,yc]=regula(f,a,b,delta,epsilon,max1)
%Input  - f is the function input as a string 'f'
%        - a and b are the left and right endpoints
%        - delta is the tolerance for the zero
%        - epsilon is the tolerance for the value of f at the zero
%        - max1 is the maximum number of iterations
%Output - c is the zero
%        - yc=f(c)
%        - err is the error estimate for c

ya=feval(f,a);
yb=feval(f,b);
if ya*yb>0
    disp('Note: f(a)*f(b)>0'),
    break,
end
for k=1:max1
    dx=yb*(b-a)/(yb-ya);
    c=b-dx;
    ac=c-a;
    yc=feval(f,c);
    if yc==0,break;
    elseif yb*yc>0
        b=c;
        yb=yc;
    else
        a=c;
        ya=yc;
    end
    dx=min(abs(dx),ac);
    if abs(dx)<delta,break,end
    if abs(yc)<epsilon,break,end
end
c;
err=abs(b-a)/2;
yc=feval(f,c);
```

# **Initial Approximation and Convergence Criteria**

# Initial Approximation and Convergence Criteria

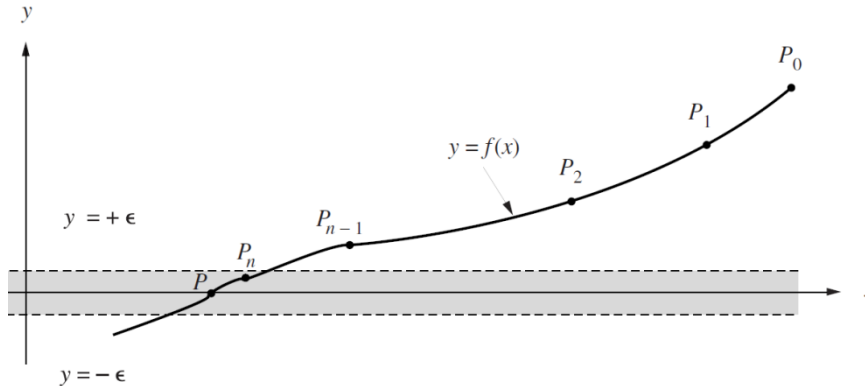
- **Bracketing methods: globally convergent.** Once the interval has been found, the iterations will proceed until a root is found
- **Newton-Raphson method or secant method: locally convergent.** Require a close approximation to the root; converge more rapidly.
- View the graph  $y = f(x)$  and make decisions based on what it looks like.



**Figure 2.10** The graph of the cubic polynomial  $y = x^3 - x^2 - x + 1$ .

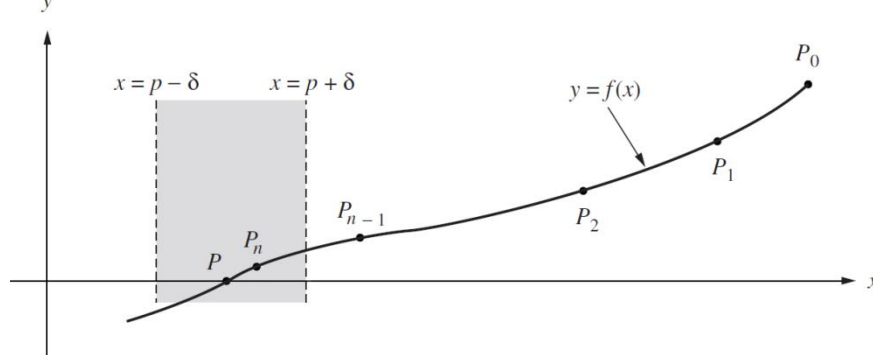
# Checking for convergence

- An algorithm must be used to compute a value  $p_n$  that is an acceptable computer solution.
- To solve  $f(x) = 0$ , the final value  $p_n$  should have the property that  $|f(p_n)| < \epsilon$ .



(a) The horizontal convergence band for locating a solution to  $f(x) = 0$ .

- We can also consider if  $p_n$  lies in the small interval  $x = p + \delta$  and  $x = p - \delta$ .

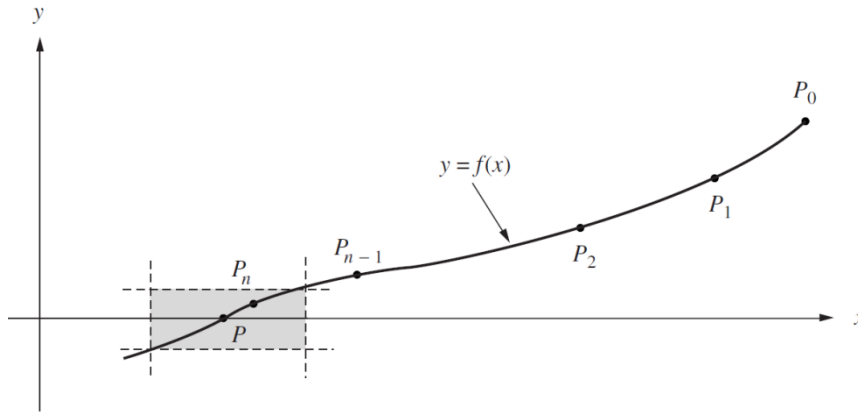


(b) The vertical convergence band for locating a solution to  $f(x) = 0$



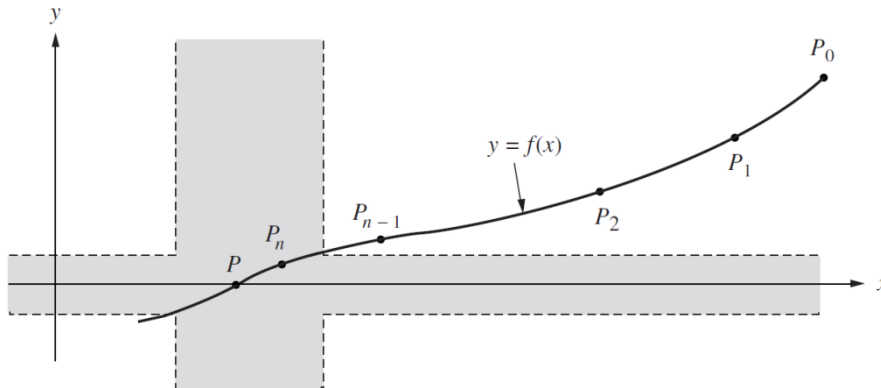
# Checking for convergence

- The second criterion is often desired, but it is difficult to implement because it involves the unknown solution  $p$ . Instead  $p_n \approx p_{n-1}$  can be used.



The size of the tolerances  $\delta$  and  $\epsilon$  are crucial, should be about  $100 \times 10^{-M}$ ,  $M$  is the number of decimal digits in the computer's floating-point numbers.

- (a) The rectangular region defined by  $|x - p| < \delta$  **AND**  $|y| < \epsilon$ .



estimate for the absolute error

$$|p_n - p_{n-1}| < \delta$$

or

$$\frac{2|p_n - p_{n-1}|}{|p_n| + |p_{n-1}|} < \delta$$

estimate for the relative error

- (b) The unbounded region defined by  $|x - p| < \delta$  **OR**  $|y| < \epsilon$ .

# Algorithm

**Program 2.4 (Approximate Location of Roots).** To roughly estimate the locations of the roots of the equation  $f(x) = 0$  over the interval  $[a, b]$ , by using the equally spaced sample points  $(x_k, f(x_k))$  and the following criteria:

(i)  $(y_{k-1})(y_k) < 0$ , or

(ii)  $|y_k| < \epsilon$  and  $(y_k - y_{k-1})(y_{k+1} - y_k) < 0$ .

That is, either  $f(x_{k-1})$  and  $f(x_k)$  have opposite signs or  $|f(x_k)|$  is small and the slope of the curve  $y = f(x)$  changes sign near  $(x_k, f(x_k))$ .

```
function R = approot (X,epsilon)

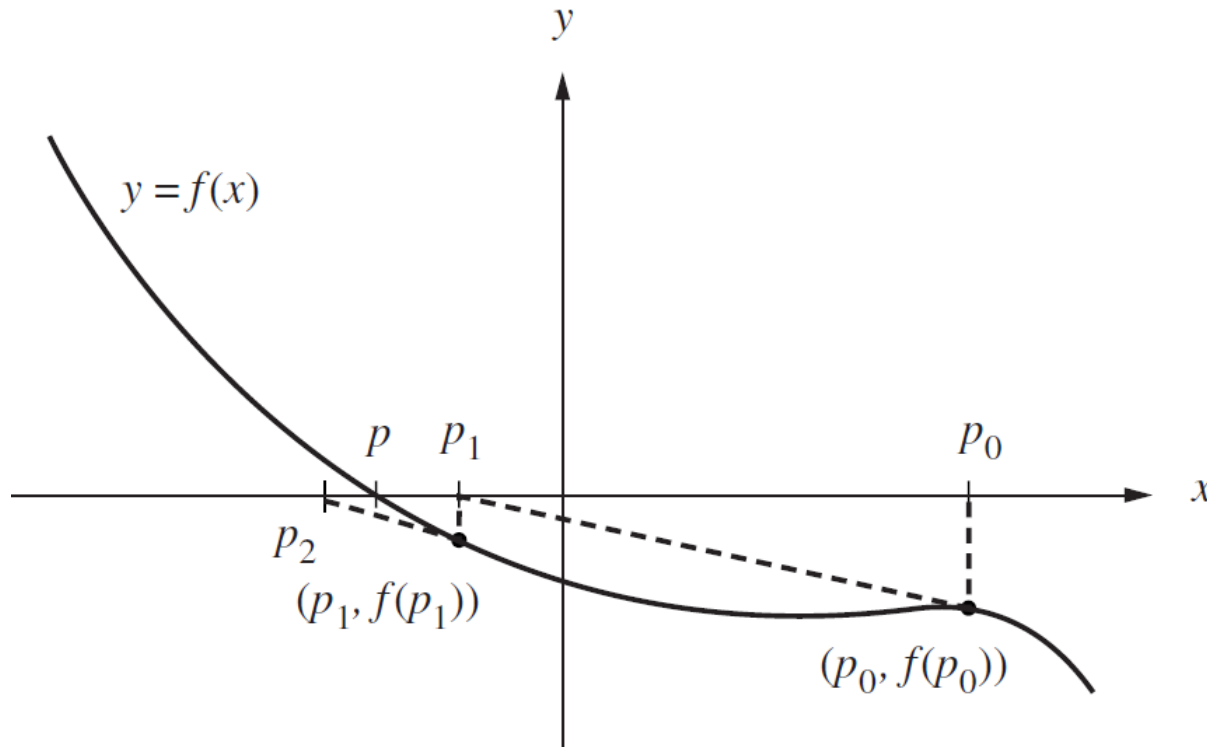
% Input  - f is the object function saved as an M-file named f.m
%         - X is the vector of abscissas
%         - epsilon is the tolerance
% Output - R is the vector of approximate roots

Y=f(X);
yrange = max(Y)-min(Y);
epsilon2 = yrange*epsilon;
n=length(X);
m=0;
X(n+1)=X(n);
Y(n+1)=Y(n);
for k=2:n,
    if Y(k-1)*Y(k)<=0,
        m=m+1;
        R(m)=(X(k-1)+X(k))/2;
    end
    s=(Y(k)-Y(k-1))*(Y(k+1)-Y(k));
    if (abs(Y(k)) < epsilon2) & (s<=0),
        m=m+1;
        R(m)=X(k);
    end
end
end
```

# **Newton-Raphson and Secant Methods**

# Slope Methods for Finding Roots

- If  $f(x)$ ,  $f'(x)$ , and  $f''(x)$  are continuous near a root  $p$ , then this extra information regarding the nature of  $f(x)$  can be used to develop algorithms that will produce sequences  $\{p_k\}$  that converge faster to  $p$  than either the bisection or false position method.



$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}.$$

**Figure 2.13** The geometric construction of  $p_1$  and  $p_2$  for the Newton-Raphson method.

# Newton-Raphson Theorem

**Theorem 2.5 (Newton-Raphson Theorem).** Assume that  $f \in C^2[a, b]$  and there exists a number  $p \in [a, b]$ , where  $f(p) = 0$ . If  $f'(p) \neq 0$ , then there exists a  $\delta > 0$  such that the sequence  $\{p_k\}_{k=0}^{\infty}$  defined by the iteration

$$(4) \quad p_k = g(p_{k-1}) = p_{k-1} - \frac{f(p_{k-1})}{f'(p_{k-1})} \quad \text{for } k = 1, 2, \dots$$

will converge to  $p$  for any initial approximation  $p_0 \in [p - \delta, p + \delta]$ .

*Remark.* The function  $g(x)$  defined by the formula

$$(5) \quad g(x) = x - \frac{f(x)}{f'(x)}$$

is called the ***Newton-Raphson iteration function***. Since  $f(p) = 0$ , it is easy to see that  $g(p) = p$ . Thus the Newton-Raphson iteration for finding the root of the equation  $f(x) = 0$  is accomplished by finding a fixed point of the function  $g(x)$ .

# Newton's Iteration for Finding Square Roots

Assume that  $A > 0$  is a real number and let  $p_0 > 0$  be an initial approximation to  $\sqrt{A}$ . Define the sequence  $\{p_k\}_{k=0}^{\infty}$  using the recursive rule

$$(11) \quad p_k = \frac{p_{k-1} + \frac{A}{p_{k-1}}}{2} \quad \text{for } k = 1, 2, \dots$$

Then the sequence  $\{p_k\}_{k=0}^{\infty}$  converges to  $\sqrt{A}$ ; that is,  $\lim_{n \rightarrow \infty} p_k = \sqrt{A}$ .

*Outline of Proof.* Start with the function  $f(x) = x^2 - A$ , and the roots of the equation  $x^2 - A = 0$  are  $\pm\sqrt{A}$ . Now use  $f(x)$  and the derivative  $f'(x)$  in formula (5) and write down the Newton-Raphson iteration formula

$$(12) \quad g(x) = x - \frac{f(x)}{f'(x)} = x - \frac{x^2 - A}{2x}.$$

This formula can be simplified to obtain

$$(13) \quad g(x) = \frac{x + \frac{A}{x}}{2}.$$

When  $g(x)$  in (13) is used to define the recursive iteration in (4), the result is formula (11). It can be proved that the sequence that is generated in (11) will converge for any starting value  $p_0 > 0$ . The details are left for the exercises.

# An example

Use Newton's square-root algorithm to find  $\sqrt{5}$ .

Starting with  $p_0 = 2$  and using formula (11), we compute

$$\begin{aligned} p_1 &= \frac{2 + 5/2}{2} = 2.25 \\ p_2 &= \frac{2.25 + 5/2.25}{2} = 2.236111111 \\ p_3 &= \frac{2.236111111 + 5/2.236111111}{2} = 2.236067978 \\ p_4 &= \frac{2.236067978 + 5/2.236067978}{2} = 2.236067978. \end{aligned}$$

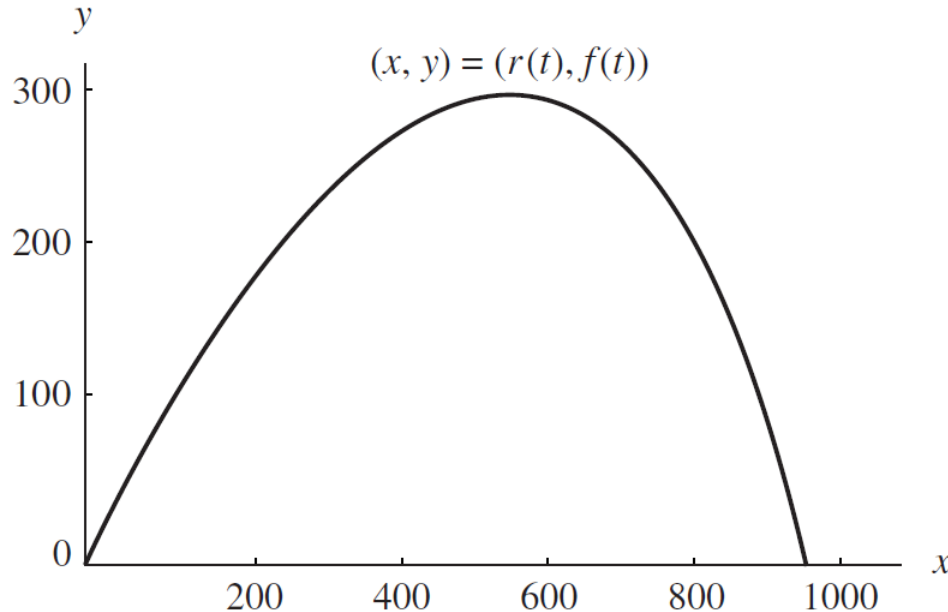
Further iterations produce  $p_k \approx 2.236067978$  for  $k > 4$ , thus convergence accurate to nine decimal places has been achieved.

# A real example: a fired projectile

Ideal model  $y = v_y t - 16t^2$  and  $x = v_x t$ ,

More realistic model  $y = f(t) = (Cv_y + 32C^2)(1 - e^{-t/C}) - 32Ct$

$$x = r(t) = Cv_x(1 - e^{-t/C})$$



**Figure 2.14** The path of a projectile with air resistance considered.



# A real example: a fired projectile

A projectile is fired with an angle of elevation  $b_0 = 45^\circ$ ,  $v_y = v_x = 160\text{ft/sec}$ , and  $C = 10$ . Find the elapsed time until impact and find the range.

The equations of motion are  $y = f(t) = 4800(1 - e^{-t/10}) - 320t$  and  $x = r(t) = 1600(1 - e^{-t/10})$ .

Since  $f(8) = 83.220972$  and  $f(9) = -31.534367$ , we will use the initial guess  $p_0 = 8$ . The derivative is  $f'(t) = 480e^{-t/10} - 320$ , and its value  $f'(p_0) = f'(8) = -104.3220972$  is used in formula (4) to get

$$p_1 = 8 - \frac{83.22097200}{-104.3220972} = 8.797731010.$$

A summary of the calculation is given in the Table below.

The value  $p_4$  has eight decimal places of accuracy, and the time until impact is  $t \approx 8.74217466$  seconds.

**Table 2.4** Finding the Time When the Height  $f(t)$  Is Zero

$k$	Time, $p_k$	$p_{k+1} - p_k$	Height, $f(p_k)$
0	8.00000000	0.79773101	83.22097200
1	8.79773101	-0.05530160	-6.68369700
2	8.74242941	-0.00025475	-0.03050700
3	8.74217467	-0.00000001	-0.00000100
4	8.74217466	0.00000000	0.00000000

# Speed of Convergence

- Simple root or multiple root

**Definition** Assume that  $f(x)$  and its derivatives  $f'(x), \dots, f^{(M)}(x)$  are defined and continuous on an interval about  $x = p$ . We say that  $f(x) = 0$  has a **root of order  $M$**  at  $x = p$  if and only if

$$(17) \quad f(p) = 0, \quad f'(p) = 0, \quad \dots, \quad f^{(M-1)}(p) = 0, \quad \text{and} \quad f^{(M)}(p) \neq 0.$$

A root of order  $M = 1$  is often called a *simple root*,  
if  $M > 1$ , it is called a *multiple root*.

A root of order  $M = 2$  is sometimes called a *double root*, and so on. The next will illuminate these concepts.

# Speed of Convergence

- Simple root or multiple root

**Lemma** If the equation  $f(x) = 0$  has a root of order  $M$  at  $x = p$ , then there exists a continuous function  $h(x)$  so that  $f(x)$  can be expressed as the product

$$f(x) = (x - p)^M h(x), \quad \text{where } h(p) \neq 0.$$

## Example

The function  $f(x) = x^3 - 3x + 2$  has a simple root at  $p = -2$  and a double root at  $p = 1$ . This can be verified by considering the derivatives  $f'(x) = 3x^2 - 3$  and  $f''(x) = 6x$ .

At the value  $p = -2$ , we have  $f(-2) = 0$  and  $f'(-2) = 9$ , so  $M = 1$  in Definition 2.4; hence  $p = -2$  is a simple root.

For the value  $p = 1$ , we have  $f(1) = f'(1) = 0$ , and  $f''(1) = 6$ , so  $M = 2$  in Definition 2.4; hence  $p = 1$  is a double root.

Also, notice that  $f(x)$  has a factorization  $f(x) = (x + 2)(x - 1)^2$ .

# Speed of Convergence

- Order of convergence

**Definition** Assume that  $\{p_n\}_{n=0}^{\infty}$  converges to  $p$  and set  $E_n = p - p_n$  for  $n \geq 0$ . If two positive constants  $A \neq 0$  and  $R > 0$  exist, and

$$\lim_{n \rightarrow \infty} \frac{|p - p_{n+1}|}{|p - p_n|^R} = \lim_{n \rightarrow \infty} \frac{|E_{n+1}|}{|E_n|^R} = A$$

Then the sequence is said to converge to  $p$  with *order of convergence*  $R$ . The number  $A$  is called the *asymptotic error constant*.

The cases  $R = 1, 2$  are given special consideration.

If  $R = 1$ , the convergence of  $\{p_n\}_{n=0}^{\infty}$  is called linear.

If  $R = 2$ , the convergence of  $\{p_n\}_{n=0}^{\infty}$  is called quadratic.

- If  $R$  is large, converges more rapidly
- Some sequences converge at a rate that is not an integer. For example, the order of convergence of the secant method is  $R \approx 1.618$

# Speed of Convergence

- Order of convergence

**Theorem 2.6 (Convergence Rate for Newton-Raphson Iteration).** Assume that Newton-Raphson iteration produces a sequence  $\{p_n\}_{n=0}^{\infty}$  that converges to the root  $p$  of the function  $f(x)$ . If  $p$  is a simple root, convergence is quadratic and

$$(22) \quad |E_{n+1}| = \frac{|f''(p)|}{|f'(p)|} |E_n|^2 \quad \text{for } n \text{ sufficiently large}$$

If  $p$  is a multiple root of order  $M$ , convergence is linear and.

$$(22) \quad |E_{n+1}| = \frac{M-1}{M} |E_n| \quad \text{for } n \text{ sufficiently large}$$

# Example: Quadratic convergence

**Example 2.14 (Quadratic Convergence at a Simple Root).** Start with  $p_0 = -2.4$  and use Newton-Raphson iteration to find the root  $p = -2$  of the polynomial  $f(x) = x^3 - 3x + 2$ . The iteration formula for computing  $\{p_k\}$  is

$$p_k = g(p_{k-1}) = \frac{2p_{k-1}^3 - 2}{3p_{k-1}^2 - 3}.$$

**Table 2.5** Newton's Method Converges Quadratically at a Simple Root

$k$	$p_k$	$p_{k+1} - p_k$	$E_k = p - p_k$	$\frac{ E_{k+1} }{ E_k ^2}$
0	-2.400000000	0.323809524	0.400000000	0.476190475
1	-2.076190476	0.072594465	0.076190476	0.619469086
2	-2.003596011	0.003587422	0.003596011	0.664202613
3	-2.000008589	0.000008589	0.000008589	
4	-2.000000000	0.000000000	0.000000000	

# Example: Linear convergence

**Example (Linear Convergence at a Double Root).** Start with  $p_0 = 1.2$  and use Newton-Raphson iteration to find the double root  $p = 1$  of the polynomial  $f(x) = x^3 - 3x + 2$ . Using formula (20) to check for linear convergence, we get the values in Table 2.6.

**Table 2.6** Newton's Method Converges Linearly at a Double Root

$k$	$p_k$	$p_{k+1} - p_k$	$E_k = p - p_k$	$\frac{ E_{k+1} }{ E_k }$
0	1.200000000	-0.096969697	-0.200000000	0.515151515
1	1.103030303	-0.050673883	-0.103030303	0.508165253
2	1.052356420	-0.025955609	-0.052356420	0.496751115
3	1.026400811	-0.013143081	-0.026400811	0.509753688
4	1.013257730	-0.006614311	-0.013257730	0.501097775
5	1.006643419	-0.003318055	-0.006643419	0.500550093
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

# Algorithm

**Program 2.5 (Newton-Raphson Iteration).** To approximate a root of  $f(x) = 0$  given one initial approximation  $p_0$  and using the iteration

$$p_k = p_{k-1} - \frac{f(p_{k-1})}{f'(p_{k-1})} \quad \text{for } k = 1, 2, \dots$$

```
function [p0,err,k,y]=newton(f,df,p0,delta,epsilon,max1)
%Input  - f is the object function input as a string 'f'
%        - df is the derivative of f input as a string 'df'
%        - p0 is the initial approximation to a zero of f
%        - delta is the tolerance for p0
%        - epsilon is the tolerance for the function values y
%        - max1 is the maximum number of iterations
%Output - p0 is the Newton-Raphson approximation to the zero
%        - err is the error estimate for p0
%        - k is the number of iterations
%        - y is the function value f(p0)
for k=1:max1
    p1=p0-feval(f,p0)/feval(df,p0);
    err=abs(p1-p0);
    relerr=2*err/(abs(p1)+delta);
    p0=p1;
    y=feval(f,p0);
    if (err<delta)|(relerr<delta)|(abs(y)<epsilon),break,end
end
```



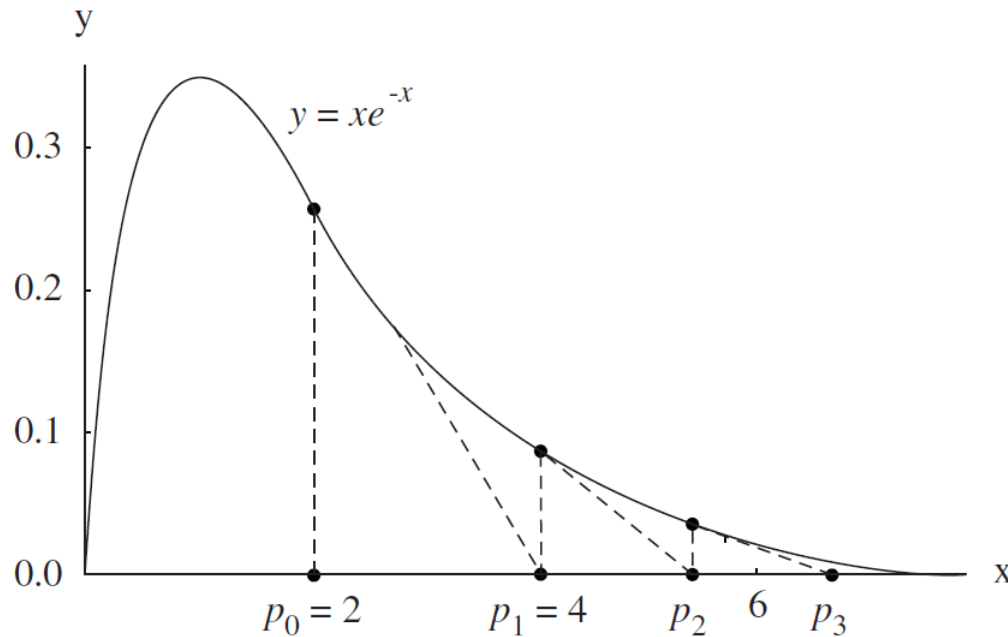
# Pitfalls

- One obvious pitfall of the Newton-Raphson method is the possibility of division by zero in formula (4), which would occur if  $f'(p_{k-1}) = 0$ .

$$(4) \quad p_k = g(p_{k-1}) = p_{k-1} - \frac{f(p_{k-1})}{f'(p_{k-1})} \quad \text{for } k = 1, 2, \dots$$

# Pitfalls

- Sometimes the initial approximation  $p_0$  is too far away from the desired root and the sequence  $\{p_k\}$  converges to some other root

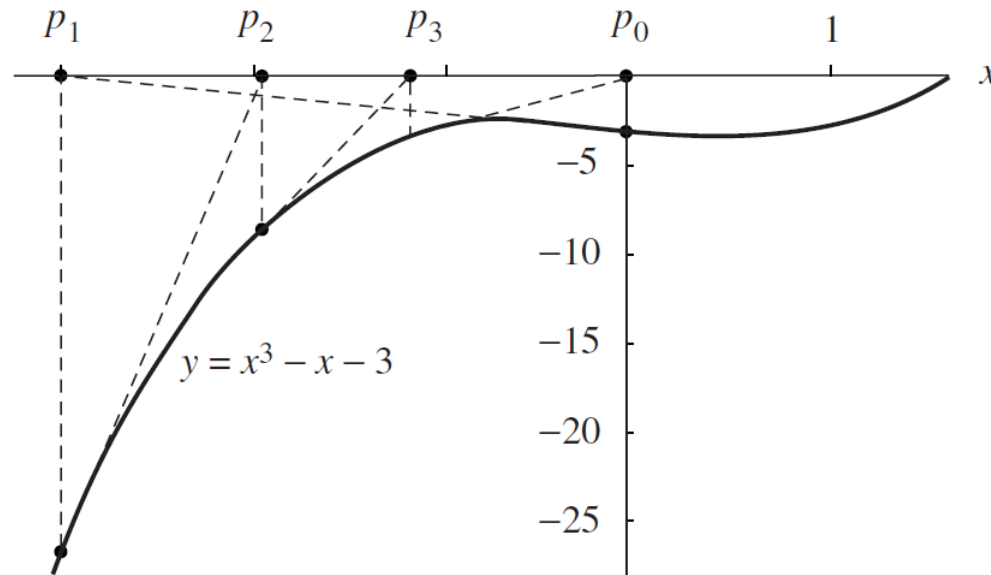


**Figure 2.15** (a) Newton-Raphson iteration for  $f(x) = xe^{-x}$  can produce a divergent sequence.

- This particular function has another surprising problem. The value of  $f(x)$  goes to zero rapidly as  $x$  gets large, for example,  $f(p_{15}) = 0.0000000536$ , and it is possible that  $p_{15}$  could be mistaken for a root.

# Pitfalls

- **Cycling**, occurs when the terms in the sequence  $\{p_k\}$  tend to repeat or almost repeat.

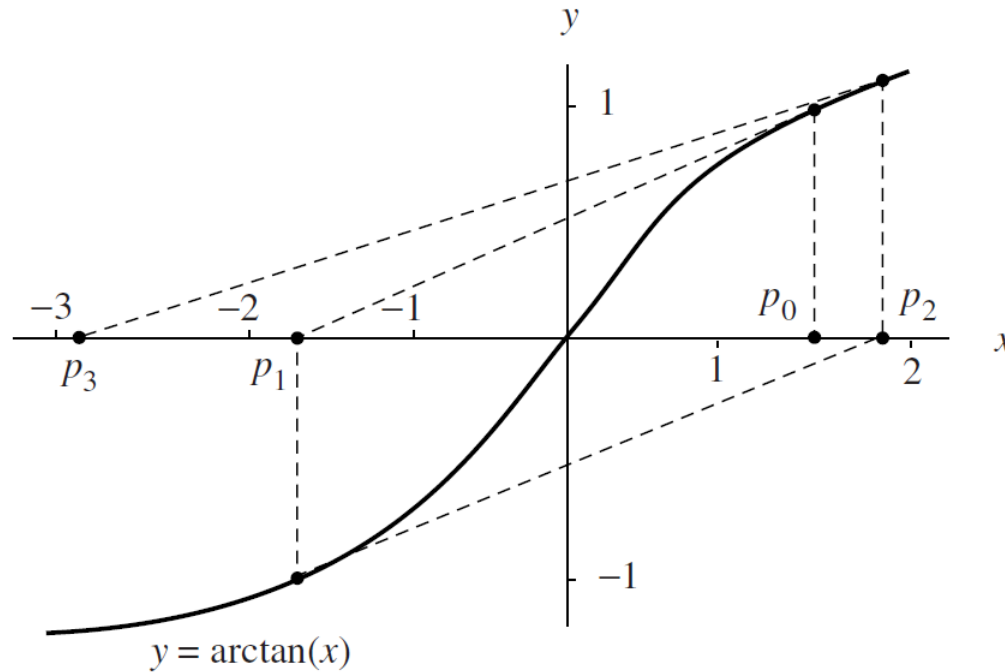


**Figure 2.15** (b) Newton-Raphson iteration for  $f(x) = x^3 - x - 3$  can produce a cyclic sequence.

- Chosen  $p_0 = 0$ , stuck in a cycle
- Chosen  $p_0 = 2$ , the sequence converges

# Pitfalls

- **Divergent oscillation**, occurs when  $|g'(x)| \geq 1$  on an interval containing the root  $p$



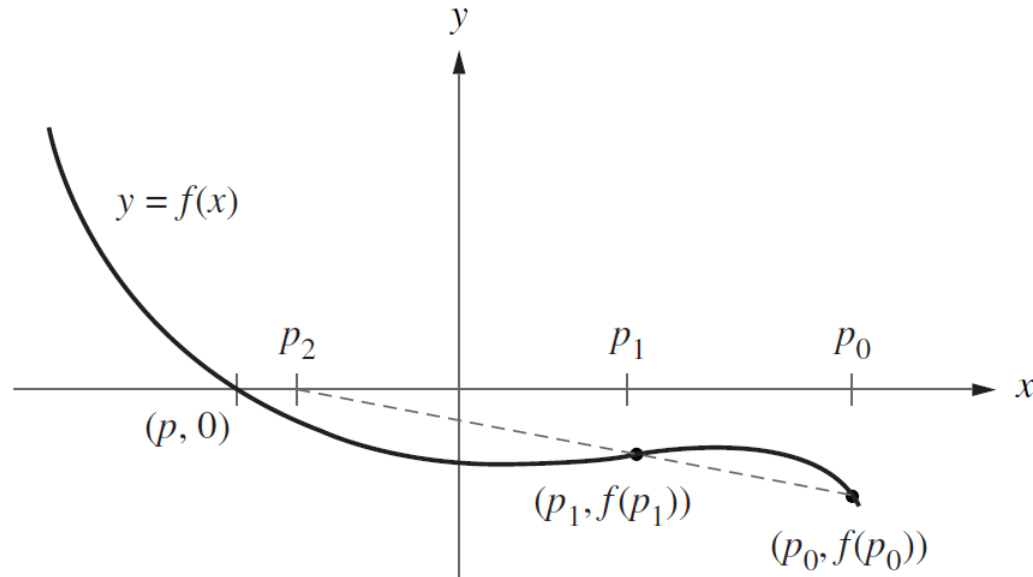
**Figure 2.15** (c) Newton-Raphson iteration for  $f(x) = \arctan(x)$  can produce a divergent oscillating sequence.

- But if the starting value is sufficiently close to the root  $p = 0$ , a convergent sequence results

# Secant Method

- The Newton-Raphson algorithm requires the evaluation of two functions per iteration,  $f(p_{k-1})$  and  $f'(p_{k-1})$
- Many functions have nonelementary forms (integrals, sums, etc.), and it is desirable to have a method that converges almost as fast as Newton's method yet involves only evaluations of  $f(x)$  and not of  $f'(x)$ .
- The Secant method will require only one evaluation of  $f(x)$  per step and at a simple root has an order of convergence  $R \approx 1.618033989$ . It is almost as fast as Newton's method, which has order 2.
- The formula involved in the secant method is the same one that was used in the false position method, except that the logical decisions regarding how to define each succeeding term are different.

# Secant Method



**Figure 2.16** The geometric construction of  $p_2$  for the secant method.

$$(25) \quad p_2 = g(p_1, p_0) = p_1 - \frac{f(p_1)(p_1 - p_0)}{f(p_1) - f(p_0)}.$$

The general term is given by the two-point iteration formula

$$(26) \quad p_{k+1} = g(p_k, p_{k-1}) = p_k - \frac{f(p_k)(p_k - p_{k-1})}{f(p_k) - f(p_{k-1})}$$

# Secant Method: An example

**Example 2.16 (Secant Method at a Simple Root).** Start with  $p_0 = -2.6$  and  $p_1 = -2.4$  and use the secant method to find the root  $p = -2$  of the polynomial function  $f(x) = x^3 - 3x + 2$ .

In this case the iteration formula (27) is

$$(27) \quad p_{k+1} = g(p_k, p_{k-1}) = p_k - \frac{(p_k^3 - 3p_k + 2)(p_k - p_{k-1})}{p_k^3 - p_{k-1}^3 - 3p_k + 3p_{k-1}}.$$

This can be algebraically manipulated to obtain

$$(28) \quad p_{k+1} = g(p_k, p_{k-1}) = \frac{p_k^2 p_{k-1} + p_k p_{k-1}^2 - 2}{p_k^2 + p_k p_{k-1} + p_{k-1}^2 - 3}$$

**Table 2.7** Convergence of the Secant Method at a Simple Root

$k$	$p_k$	$p_{k+1} - p_k$	$E_k = p - p_k$	$\frac{ E_{k+1} }{ E_k ^{1.618}}$
0	-2.600000000	0.200000000	0.600000000	0.914152831
1	-2.400000000	0.293401015	0.400000000	0.469497765
2	-2.106598985	0.083957573	0.106598985	0.847290012
3	-2.022641412	0.021130314	0.022641412	0.693608922
4	-2.001511098	0.001488561	0.001511098	0.825841116
5	-2.000022537	0.000022515	0.000022537	0.727100987
6	-2.000000022	0.000000022	0.000000022	
7	-2.000000000	0.000000000	0.000000000	

# Algorithm

**Program 2.6 (Secant Method).** To approximate a root of  $f(x) = 0$  given two initial approximations  $p_0$  and  $p_1$  and using the iteration

$$p_{k+1} = p_k - \frac{f(p_k)(p_k - p_{k-1})}{f(p_k) - f(p_{k-1})} \quad \text{for } k = 1, 2, \dots$$

```
function [p1,err,k,y]=secant(f,p0,p1,delta,epsilon,max1)
%Input  - f is the object function input as a string 'f'
%        - p0 and p1 are the initial approximations to a zero
%        - delta is the tolerance for p1
%        - epsilon is the tolerance for the function values y
%        - max1 is the maximum number of iterations
%Output - p1 is the secant method approximation to the zero
%        - err is the error estimate for p1
%        - k is the number of iterations
%        - y is the function value f(p1)
for k=1:max1
    p2=p1-feval(f,p1)*(p1-p0)/(feval(f,p1)-feval(f,p0));
    err=abs(p2-p1);
    relerr=2*err/(abs(p2)+delta);
    p0=p1;
    p1=p2;
    y=feval(f,p1);
    if (err<delta)|(relerr<delta)|(abs(y)<epsilon),break,end
end
```



# Accelerated Convergence

**Theorem (Acceleration of Newton-Raphson Iteration).** Suppose that the Newton-Raphson algorithm produced a sequence that converges linearly to the root  $x = p$  of order  $M > 1$ . Then the Newton-Raphson iteration formula

$$(30) \quad p_k = p_{k-1} - \frac{M f(p_{k-1})}{f'(p_{k-1})}$$

will produce a sequence  $\{p_k\}_{k=0}^{\infty}$  that converges quadratically to  $p$ .

**Example (Acceleration of Convergence at a Double Root).** Start with  $p_0 = 1.2$  and use accelerated Newton-Raphson iteration to find the double root  $p = 1$  of  $f(x) = x^3 - 3x + 2$ .

Since  $M = 2$ , the acceleration formula (31) becomes

$$p_k = p_{k-1} - 2 \frac{f(p_{k-1})}{f'(p_{k-1})} = \frac{p_{k-1}^3 + 3p_{k-1} - 4}{3p_{k-1}^2 - 3}.$$

**Table 2.8** Acceleration of Convergence at a Double Root

$k$	$p_k$	$p_{k+1} - p_k$	$E_k = p - p_k$	$\frac{ E_{k+1} }{ E_k ^2}$
0	1.200000000	-0.193939394	-0.200000000	0.151515150
1	1.006060606	-0.006054519	-0.006060606	0.165718578
2	1.000006087	-0.000006087	-0.000006087	
3	1.000000000	0.000000000	0.000000000	

# Comparison of the Speed of Convergence

**Table 2.9** Comparison of the Speed of Convergence

Method	Special considerations	Relation between successive error terms
Bisection		$E_{k+1} \approx \frac{1}{2} E_k $
Regula falsi		$E_{k+1} \approx A E_k $
Secant method	Multiple root	$E_{k+1} \approx A E_k $
Newton-Raphson	Multiple root	$E_{k+1} \approx A E_k $
Secant method	Simple root	$E_{k+1} \approx A E_k ^{1.618}$
Newton-Raphson	Simple root	$E_{k+1} \approx A E_k ^2$
Accelerated Newton-Raphson	Multiple root	$E_{k+1} \approx A E_k ^2$

# **Aitken's Process and Steffensen's and Muller's Methods**

# Aitken's Process

A technique called *Aitken's  $\Delta^2$  process* can be used to speed up convergence of any sequence that is linearly convergent.

**Definition 2.6.** Given the sequence  $\{p_n\}_{n=0}^{\infty}$ , define the forward difference  $\Delta p_n$  by

$$(1) \quad \Delta p_n = p_{n+1} - p_n \quad \text{for } n \geq 0.$$

Higher powers  $\Delta^k p_n$  are defined recursively by

$$(2) \quad \Delta^k p_n = \Delta^{k-1} (\Delta p_n) \quad \text{for } k \geq 2.$$

# Aitken's Process

**Theorem 2.8 (Aitken's Acceleration).** Assume that the sequence  $\{p_n\}_{n=0}^{\infty}$  converges linearly to the limit  $p$  and that  $p - p_n \neq 0$  for all  $n \geq 0$ . If there exists a real number  $A$  with  $|A| < 1$  such that

$$(3) \quad \lim_{n \rightarrow \infty} \frac{p - p_{n+1}}{p - p_n} = A.$$

Then the sequence  $\{q_n\}_{n=0}^{\infty}$  defined by

$$(4) \quad q_n = p_n - \frac{(\Delta p_n)^2}{\Delta^2 p_n} = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}$$

converges to  $p$  faster than  $\{p_n\}_{n=0}^{\infty}$ , in the sense that

$$(5) \quad \lim_{n \rightarrow \infty} \left| \frac{p - q_n}{p - p_n} \right| = 0.$$

# Aitken's Process

**Recall the previous example**

Consider the convergent iteration

$$p_0 = 0.5 \quad \text{and} \quad p_{k+1} = e^{-p_k} \quad \text{for } k = 0, 1, \dots$$

The first 10 terms are obtained by the calculations

$$\begin{aligned} p_1 &= e^{-0.500000} = 0.606531 \\ p_2 &= e^{-0.606531} = 0.545239 \\ p_3 &= e^{-0.545239} = 0.579703 \\ &\vdots \\ p_9 &= e^{-0.566409} = 0.567560 \\ p_{10} &= e^{-0.567560} = 0.566907 \end{aligned}$$

The sequence is converging, and further calculation reveal that

$$\lim_{n \rightarrow \infty} p_n = 0.567143 \dots$$

Thus, we found an approximation for the fixed point of the function  $y = e^{-x}$ .

# Aitken's Process

Now we will show that the sequence  $\{p_n\}$  in that example exhibits linear convergence, and show that the sequence  $\{q_n\}$  obtained by Aitken's  $\Delta^2$  process converges faster.

The sequence  $\{p_n\}$  was obtained by fixed-point iteration using the function  $g(x) = e^{-x}$  and starting with  $p_0 = 0.5$ . After convergence has been achieved, the limit is  $P \approx 0.567143290$ . The values  $p_n$  and  $q_n$  are given in Tables 2.10 and 2.11. For illustration, the value of  $q_1$  is given by the calculation

$$q_1 = p_1 - \frac{(p_2 - p_1)^2}{p_3 - 2p_2 + p_1} = 0.567298989$$

# Aitken's Process

**Table 2.10** Linearly Convergent Sequence  $\{p_n\}$

$n$	$p_n$	$E_n = p_n - p$	$A_n = \frac{E_n}{E_{n-1}}$
1	0.606530660	0.039387369	−0.586616609
2	0.545239212	−0.021904079	−0.556119357
3	0.579703095	0.012559805	−0.573400269
4	0.560064628	−0.007078663	−0.563596551
5	0.571172149	0.004028859	−0.569155345
6	0.564862947	−0.002280343	−0.566002341

**Table 2.11** Derived Sequence  $\{q_n\}$  Using Aitken's Process

$n$	$q_n$	$q_n - p$
1	0.567298989	0.000155699
2	0.567193142	0.000049852
3	0.567159364	0.000016074
4	0.567148453	0.000005163
5	0.567144952	0.000001662
6	0.567143825	0.000000534

Although the sequence  $\{q_n\}$  in Table 2.11 converges linearly, it converges faster than  $\{p_n\}$  in the sense of Theorem 2.8, and usually Aitken's method gives a better improvement than this.



# Steffensen's acceleration

When Aitken's process is combined with fixed-point iteration, the result is called *Steffensen's acceleration*.

**Program 2.7 (Steffensen's Acceleration).** To quickly find a solution of the fixed-point equation  $x = g(x)$  given an initial approximation  $p_0$ ; where it is assumed that both  $g(x)$  and  $g'(x)$  are continuous,  $|g'(x)| < 1$ , and that ordinary fixed-point iteration converges slowly (linearly) to  $p$ .

```
function [p,Q]=steff(f,df,p0,delta,epsilon,max1)
%Input  - f is the object function input as a string 'f'
%        - df is the derivative of f input as a string 'df'
%        - p0 is the initial approximation to a zero of f
%        - delta is the tolerance for p0
%        - epsilon is the tolerance for the function values y
%        - max1 is the maximum number of iterations
%Output - p is the Steffensen approximation to the zero
%        - Q is the matrix containing the Steffensen sequence

%Initialize the matrix R
R=zeros(max1,3);
R(1,1)=p0;
```


# Steffensen's acceleration

```
for k=1:max1
    for j=2:3
        %Denominator in Newton-Raphson method is calculated
        nrdenom=feval(df,R(k,j-1));

        %Calculate Newton-Raphson approximations
        if nrdenom==0
            'division by zero in Newton-Raphson method'
            break
        else
            R(k,j)=R(k,j-1)-feval(f,R(k,j-1))/nrdenom;
        end

        %Denominator in Aitken's acceleration process calculated
        aadenom=R(k,3)-2*R(k,2)+R(k,1);

        %Calculate Aitken's acceleration approximations
        if aadenom==0
            'division by zero in Aitken's acceleration'
            break
        else
            R(k+1,1)=R(k,1)-(R(k,2)-R(k,1))^2/aadenom;
        end
    end
end
```



# Steffensen's acceleration

```
%End program if division by zero occurred
if (nrdenom==0)|(aadenom==0)
    break
end

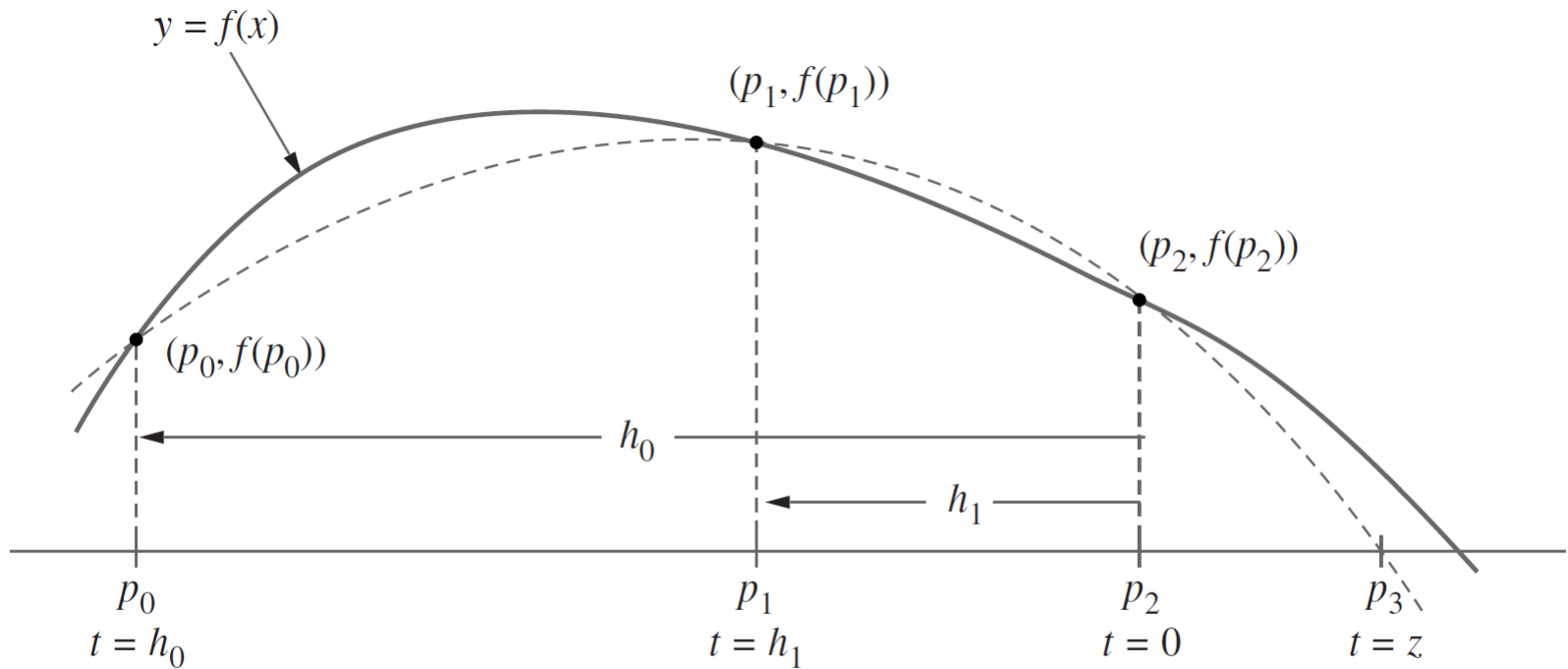
%Stopping criteria are evaluated
err=abs(R(k,1)-R(k+1,1));
relerr=err/(abs(R(k+1,1))+delta);
y=feval(f,R(k+1,1));
if (err<delta)|(relerr<delta)|(y<epsilon)
    % p and the matrix Q are determined
    p=R(k+1,1);
    Q=R(1:k+1,:);
    break
end

end
```

# Muller's Method

- Muller's method is a generalization of the secant method, in the sense that it does not require the derivative of the function.
- It is an iterative method that requires three starting points  $(p_0, f(p_0))$ ,  $(p_1, f(p_1))$ , and  $(p_2, f(p_2))$ .
- A parabola is constructed that passes through the three points; then the quadratic formula is used to find a root of the quadratic for the next approximation.
- It has been proved that near a simple root Muller's method converges faster than the secant method and almost as fast as Newton's method.
- The method can be used to find real or complex zeros of a function and can be programmed to use complex arithmetic.

# Muller's Method



**Figure 2.17** The starting approximations  $p_0$ ,  $p_1$ , and  $p_2$  for Muller's method, and the differences  $h_0$  and  $h_1$ .

# Muller's Method

We assume that  $p_2$  is the best approximation to the root and consider the parabola through the three starting values. Make the change of variable

$$(9) \quad t = x - p_2,$$

and use the differences

$$(10) \quad h_0 = p_0 - p_2 \quad \text{and} \quad h_1 = p_1 - p_2.$$

Consider the quadratic polynomial involving the variable  $t$ :

$$(11) \quad y = at^2 + bt + c.$$

Each point is used to obtain an equation involving  $a$ ,  $b$ , and  $c$ :

$$(12) \quad \begin{aligned} \text{At } t = h_0: \quad & ah_0^2 + bh_0 + c = f_0, \\ \text{At } t = h_1: \quad & ah_1^2 + bh_1 + c = f_1, \\ \text{At } t = 0: \quad & a0^2 + b0 + c = f_2. \end{aligned}$$

From the third equation in (12), we see that

$$(13) \quad c = f_2.$$

Substituting (13) into the first two equations in (12) and using the definition  $e_0 = f_0 - c$  and  $e_1 = f_1 - c$  results in the linear system

$$(14) \quad \begin{aligned} ah_0^2 + bh_0 &= f_0 - c = e_0, \\ ah_1^2 + bh_1 &= f_1 - c = e_1. \end{aligned}$$

# Muller's Method

Solving the linear system for  $a$  and  $b$  results in

$$(15) \quad \begin{aligned} a &= \frac{e_0 h_1 - e_1 h_0}{h_1 h_0^2 - h_0 h_1^2}, \\ b &= \frac{e_1 h_0^2 - e_0 h_1^2}{h_1 h_0^2 - h_0 h_1^2}. \end{aligned}$$

The quadratic formula is used to find the roots  $t = z_1, z_2$  of (11):

$$(16) \quad z = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}.$$

Formula (16) is equivalent to the standard formula for the roots of a quadratic and is better in this case because we know that  $c = f_2$ .

To ensure the stability of the method, we choose the root in (16) that has the smallest absolute value. If  $b > 0$ , use the positive sign with the square root, and if  $b < 0$ , use the negative sign. Then  $p_3$  is shown in Figure 2.17 and is given by

$$(17) \quad p_3 = p_2 + z.$$

To update the iterates, choose the new  $p_0$  and the new  $p_1$  to be the two values selected from among the old  $\{p_0, p_1, p_3\}$  that lie closest to  $p_3$  (i.e., throw out the one that is farthest away). Then take new  $p_2$  to be old  $p_3$ . Although a lot of auxiliary calculations are done in Muller's method, it only requires one function evaluation per iteration.

# Muller's Method

**Program 2.8 (Muller's Method).** To find a root of the equation  $f(x) = 0$  given three distinct initial approximations  $p_0$ ,  $p_1$ , and  $p_2$ .

```
function [p,y,err]=muller(f,p0,p1,p2,delta epsilon,max1)
%Input  - f is the object function input as a string 'f'
%        - p0, p1, and p2 are the initial approximations
%        - delta is the tolerance for p0, p1, and p2
%        - epsilon the the tolerance for the function values y
%        - max1 is the maximum number of iterations
%Output - p is the Muller approximation to the zero of f
%        - y is the function value y = f(p)
%        - err is the error in the approximation of p.

%Initialize the matrices P and Y
P=[p0 p1 p2];
Y=feval(f,P);

%Calculate a and b in formula (15)
for k=1:max1
    h0=P(1)-P(3);h1=P(2)-P(3);e0=Y(1)-Y(3);e1=Y(2)-Y(3);c=Y(3);
    denom=h1*h0^2-h0*h1^2;
    a=(e0*h1-e1*h0)/denom;
    b=(e1*h0^2-e0*h1^2)/denom;
```



# Muller's Method

```
%Suppress any complex roots
if b^2-4*a*c > 0
    disc=sqrt(b^2-4*a*c);
else
    disc=0;
end

%Find the smallest root of (17)
if b < 0
    disc=-disc;
end

z=-2*c/(b+disc);
p=P(3)+z;

%Sort the entries of P to find the two closest to p
if abs(p-P(2))<abs(p-P(1))
    Q=[P(2) P(1) P(3)];
    P=Q;
    Y=feval(f,P);
end
```

# Muller's Method

```
if abs(p-P(3))<abs(p-P(2))
    R=[P(1) P(3) P(2)];
    P=R;
    Y=feval(f,P);
end

%Replace the entry of P that was farthest from p with p
P(3)=p;
Y(3) = feval(f,P(3));
y=Y(3);

%Determine stopping criteria
err=abs(z);
relerr=err/(abs(p)+delta);
if (err<delta)|(relerr<delta)|(abs(y)<epsilon)
    break
end
end
```

# Comparison of Methods

**Example 2.19 (Convergence near a Simple Root).** This is a comparison of methods for the function  $f(x) = x^3 - 3x + 2$  near the simple root  $p = -2$ .

Newton's method and the secant method for this function were given in Examples 2.14 and 2.16, respectively. Table 2.12 provides a summary of calculations for the methods.

**Table 2.12** Comparison of Convergence near a Simple Root

$k$	Secant method	Muller's method	Newton's method	Steffensen with Newton
0	-2.600000000	-2.600000000	-2.400000000	-2.400000000
1	-2.400000000	-2.500000000	-2.076190476	-2.076190476
2	-2.106598985	-2.400000000	-2.003596011	-2.003596011
3	-2.022641412	-1.985275287	-2.000008589	-1.982618143
4	-2.001511098	-2.000334062	-2.000000000	-2.000204982
5	-2.000022537	-2.000000218		-2.000000028
6	-2.000000022	-2.000000000		-2.000002389
7	-2.000000000			-2.000000000

# Comparison of Methods

**Example 2.19 (Convergence near a Double Root).** This is a comparison of methods for the function  $f(x) = x^3 - 3x + 2$  near the double root  $p = 1$ . Table 2.13 provides a summary of calculations for the methods.

**Table 2.13** Comparison of Convergence near a Double Root

$k$	Secant method	Muller's method	Newton's method	Steffensen with Newton
0	1.400000000	1.400000000	1.200000000	1.200000000
1	1.200000000	1.300000000	1.103030303	1.103030303
2	1.138461538	1.200000000	1.052356417	1.052356417
3	1.083873738	1.003076923	1.026400814	0.996890433
4	1.053093854	1.003838922	1.013257734	0.998446023
5	1.032853156	1.000027140	1.006643418	0.999223213
6	1.020429426	0.999997914	1.003325375	0.999999193
7	1.012648627	0.999999747	1.001663607	0.999999597
8	1.007832124	1.000000000	1.000832034	0.999999798
9	1.004844757		1.000416075	0.999999999
	$\vdots$		$\vdots$	

# Comparison of Methods

- Newton's method is the best choice for finding a simple root (see Table 2.12).
- At a double root, either Muller's method or Steffensen's method with the Newton-Raphson formula is a good choice (see Table 2.13).
- Note in the Aitken's acceleration formula (4) that division by zero can occur as the sequence  $\{p_k\}$  converges. In this case, the last calculated approximation to zero should be used as the approximation to the zero of  $f$ .