Lecture 3

Solving Equations

Ye Ding (丁 烨)

Email: y.ding@sjtu.edu.cn

School of Mechanical Engineering

Shanghai Jiao Tong University

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Solving Equations

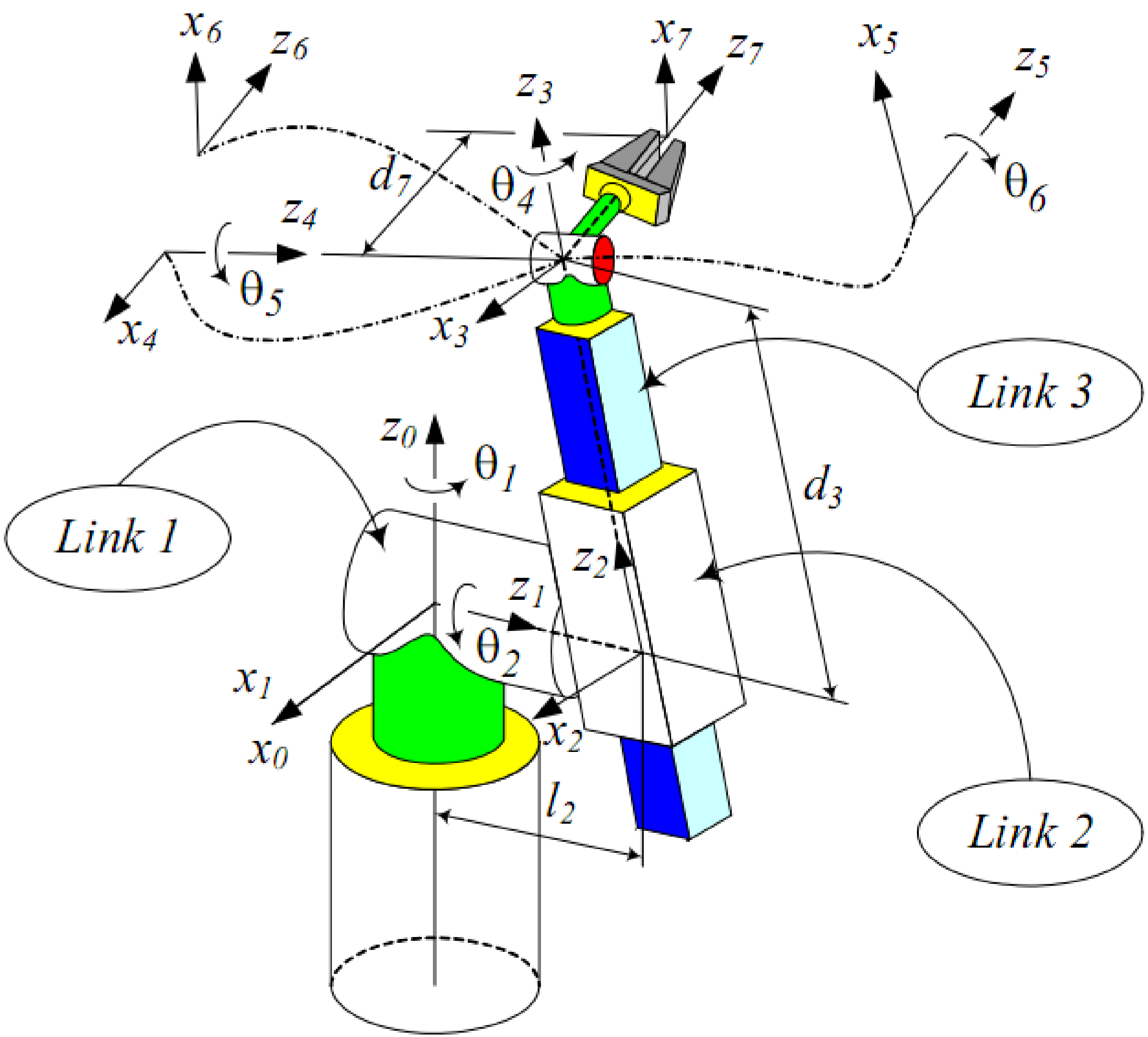
- **References for Solving Equations**

[1] Timothy Sauer, Numerical analysis (2nd ed.), Pearson Education, 2012. Chapter 1

[2] Cleve Moler, Numerical Computing with MATLAB, Society for Industrial and Applied Mathematics, 2004. Chapter 4

Solving Equations

Motivation: Example from Robotics

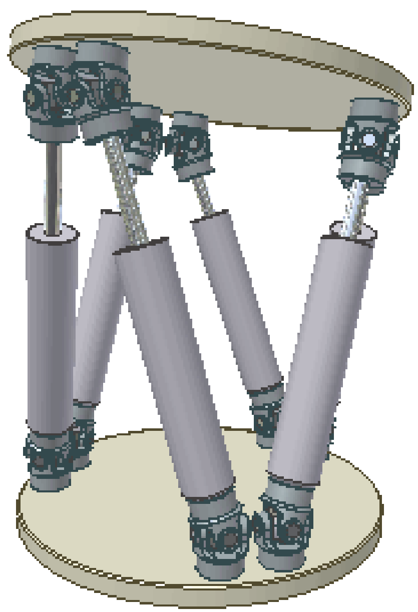


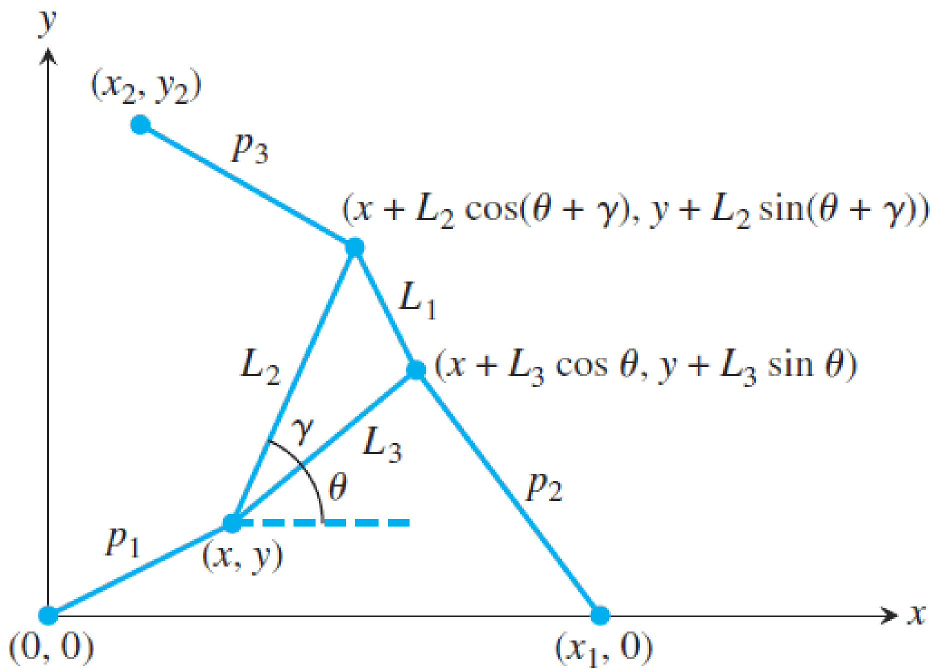
运动学方程:

$$\mathbf{f}(\boldsymbol{\Theta}) = \mathbf{x}_E$$

Solving Equations

Motivation: Example from Robotics





运动学方程:

$$\mathbf{f}(\mathbf{x}_E) = \mathbf{p}$$

Solving Equations

Motivation

Given a continuous function $f(x)$, find the values of x that satisfy the equation:

$$f(x) = 0.$$

The solutions are called the **zeros** or the **roots** of the equation. In general, the equation is impossible to solve exactly.

- **Symbolic Computation**
- **Numerical Computation**

Symbolic Computation

A new datatype: a symbolic object

Symbolic objects can be created with the sym and syms commands.

```
>> syms a b c x
>> whos
  Name      Size            Bytes  Class    Attributes

  a         1x1               112    sym
  b         1x1               112    sym
  c         1x1               112    sym
  x         1x1               112    sym
```

Symbolic Computation

- **A new datatype: a symbolic object**

Symbolic objects can be created with the sym and syms commands.

```
>> a = sym('a'); b = sym('b'); c = sym('c'); x = sym('x');
>> whos
  Name      Size            Bytes  Class    Attributes

  a         1x1               112    sym
  b         1x1               112    sym
  c         1x1               112    sym
  x         1x1               112    sym
```

Symbolic Computation

- **A new datatype: a symbolic object**

  Symbolic objects can be created with the sym and syms commands.

```
>> y= ' 1/sqrt(2*x) '
```

```
>> f=sym( ' 1/sqrt(2*x) ')
```

Symbolic Computation

- **Find symbolic variables in symbolic expression, matrix, or function**

```
>> syms a b c x k t y
>> f=a*(2*x-t)^3+b*sin(4*y)
f =
b*sin(4*y) - a*(t - 2*x)^3

>> findsym(f)
```

Symbolic Computation

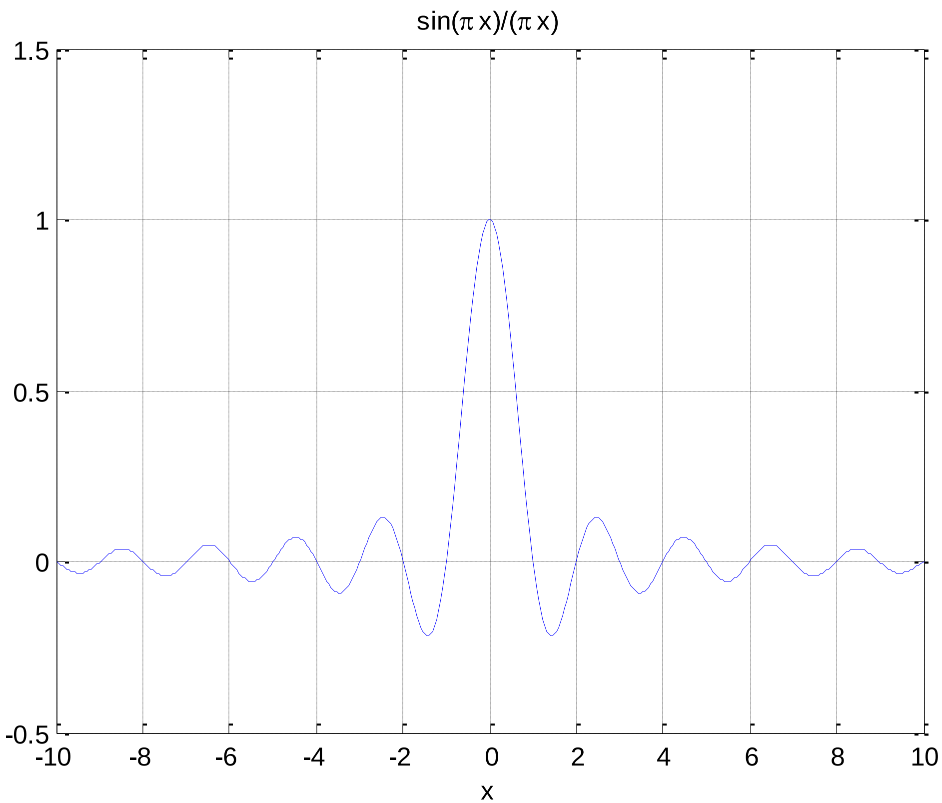
- **Algebraic simplification**

```
>> y = ((x^p)^(p+1))/x^(p-1)
y =
x^(1 - p)*(x^p)^(p + 1)
>> y_sim = simplify(y)
ans =
x*(x^p)^p
```

Symbolic Computation

- **Plot a symbolic expression**

```
>> syms x
>> f = sin(pi * x)/(pi * x);
>> ezplot(f,[-10,10]);axis([-10,10,-0.5,1.5]),grid on
```



Symbolic Computation

- **Evaluate a symbolic solution for numerical values**

```
>> syms x a b c
>> y = a*x^2 + b*x + c;
>> a = 1;b = -1;c = -1;
>> y
y =
a*x^2 + b*x + c
>> subs(y)
```

Symbolic Computation

- **Evaluate a symbolic solution for numerical values**

```
>> syms x a b c
>> y = a*x^2 + b*x + c;
>> y
y =
a*x^2 + b*x + c
>> subs(y,{a,b,c},{1,-1,-1})
ans =
x^2 - x - 1
```

Symbolic Computation

- **Evaluate a symbolic solution for numerical values**

```
>> syms x
>> y = x^2 + 1;
>> y_num = subs(y,x,[1:3])
```

Symbolic Computation

Linear Algebra

```
>> syms a b c
>> A=[a b c;b c a;c a b];
>> B=[1 1 1]';
>> x=A\B
x =
 1/(a + b + c)
 1/(a + b + c)
 1/(a + b + c)
```

```
>> L=eig(A)
L =
  (a^2 - a*b - a*c + b^2 - b*c + c^2)^(1/2)
 -(a^2 - a*b - a*c + b^2 - b*c + c^2)^(1/2)
                    a + b + c
```

Symbolic Computation

- **Polynomials and Rationals**

```
>> syms x
>> p = (2/3)*x^3-x^2-3*x+1
p =
(2*x^3)/3 - x^2 - 3*x + 1
>> [c, terms] = coeffs(p,x)
```

Symbolic Computation

Polynomials and Rationals

```
>> syms x
>> p = (2/3)*x^3-x^2-3*x+1;
>> [c, terms] = coeffs(p,x)
c =
[ 2/3, -1, -3, 1]
terms =
[ x^3, x^2, x, 1]
>> a = sym2poly(p)
```

Symbolic Computation

- **Polynomials and Rationals**

```
>> syms x
>> p = (2/3)*x^3-x^2-3*x+1;
>> a = sym2poly(p)
a =
    0.6667   -1.0000   -3.0000    1.0000
>> q = poly2sym(a)
```

Symbolic Computation

- **Solve equations**

```
>> syms a b x c
>> f = a*x^2+b*x+c;
>> s = solve(f)
s =
 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

```
>> ss = solve(f == 0)
ss =
 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

Symbolic Computation

- **Solve equations**

```
>> syms a b x c
>> f = a*x^2+b*x+c;
>> s = solve(f)
s =
 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

```
>> ss = solve(f,b)
ss =
-(a*x^2 + c)/x
```

Symbolic Computation

- **Solve equations**

```
>> syms x y
>> [x,y]=solve(x^2+x*y+y == 3,x^2-4*x+3 == 0)
x =
    1
    3
y =
    1
    -3/2
```

Symbolic Computation

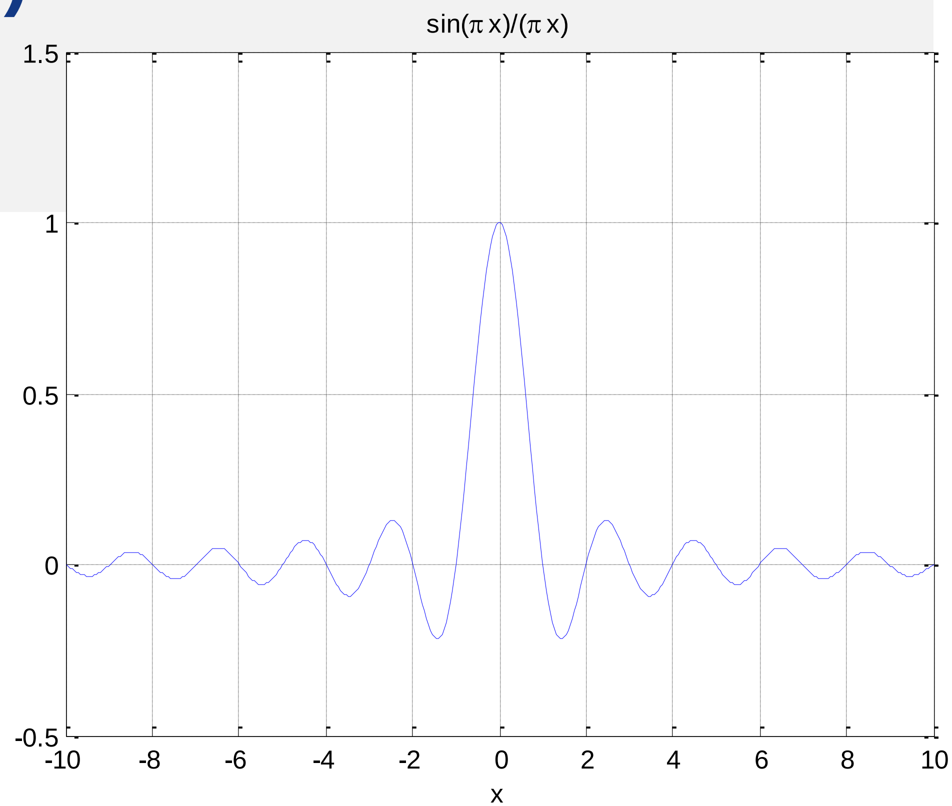
- **Solve equations**

```
>> syms x y
>> S = solve(y == 1/(1+x^2),y == 1.001 - 0.5*x);
>> S1_num = double([S.x(1), S.y(1)])
S1_num =
    0.0020    1.0000
>> S2_num = double([S.x(2), S.y(2)])
S2_num =
   1.0633 + 0.0000i   0.4693 - 0.0000i
>> [sol_x,sol_y] = solve(y == 1/(1+x^2),y ==
1.001 - 0.5*x)
```

Symbolic Computation

- **Solve equations**

```
>> syms x
>> f = sin(pi * x)/(pi * x);
>> x_0 = solve(f == 0)
```



Numerical Computation

Numerical Methods

- ✓ Bisection Method
- ✓ Fixed-Point Iteration Method
- ✓ Newton’s Method
- ✓ The Secant Method

Numerical Computation

Bisection Method: Basic Idea

Let f be a continuous function on [a,b], satisfying $f(a)f(b) < 0$. Then f has a root between a and b, that is, there exists a number r satisfying a<r<b and $f(r) = 0$.

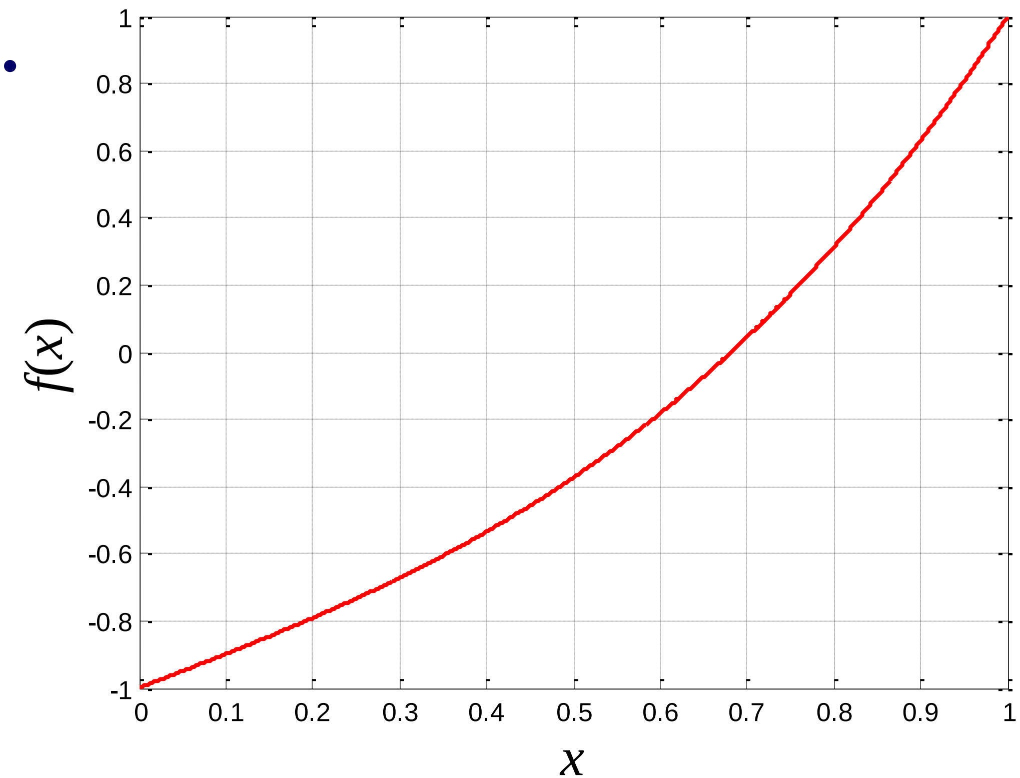
$$f(x) = x^3 + x - 1 = 0,\ x \in [0,1]$$

```
f = @(x) x.^3 + x - 1;
x = linspace(0,1,1000);
plot(x,f(x),'r-','LineWidth',2); grid on
```

Numerical Computation

Bisection Method: Basic Idea

Let f be a continuous function on [a,b], satisfying $f(a)f(b) < 0$. Then f has a root between a and b, that is, there exists a number r satisfying a<r<b and $f(r) = 0$.

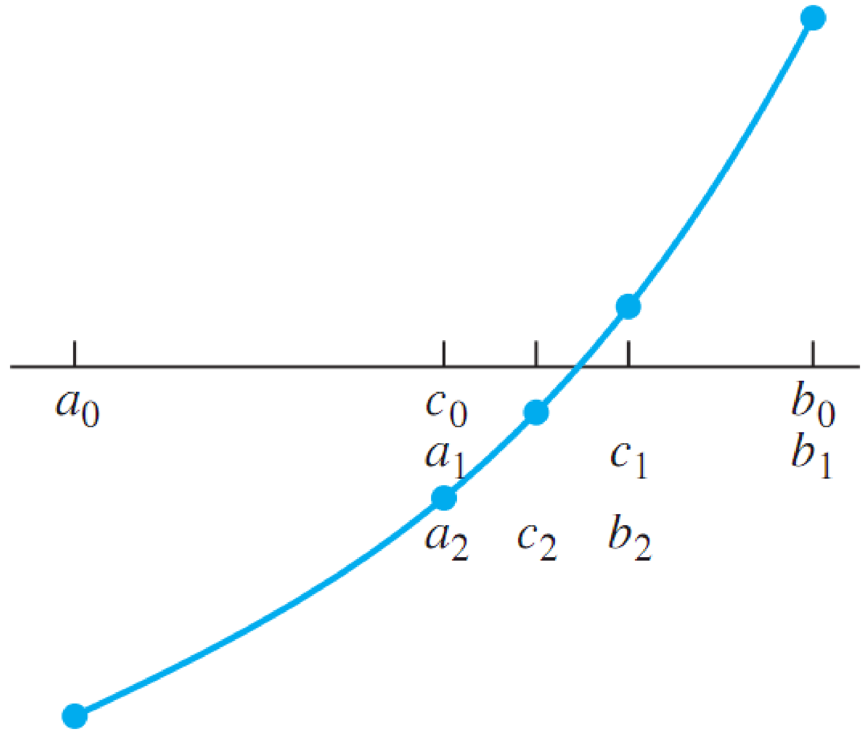


[0,1]

[0.5, 1]

Numerical Computation

Bisection Method: Basic Idea



Step 1: the sign of $f(c_0)$ is checked. Since $f(c_0)f(b_0)<0$, set $a_1 = c_0$, $b_1 = b_0$. $[a_0, b_0] \to [a_1, b_1]$

Step 2: $[a_1, b_1] \to [a_2, b_2]$

Step 3: …

Numerical Computation

Bisection Method: Pseudo-code

Given initial interval [a,b] such that *f*(a)*f*(b)< 0

while $(b-a)/2 > \mathrm{TOL}$
    $c = (a+b)/2$
    if $f(c) = 0$, **stop, end**
    if $f(a)f(c) < 0$
        $b = c$
    else
        $a = c$
    end
end

The approximate root is (a + b)/2.

Numerical Computation

Bisection Method: Matlab code

```
while (b-a)/2 > TOL
    c=(a+b)/2; fc=f(c);
    if (fc == 0)
        break;
    end
    if sign(fc) == sign(f(b))
        b = c;
    else
        a = c;
    end
end
xc = (a+b)/2;
```

Numerical Computation

Bisection Method: Example 1

```
x_star = 0.682327803828019; % Symbolic Sol

f = @(x) x.^3 + x - 1;
TOL = 0.5e-4;
xc = BisectionMethod(f,0,1,TOL)
sol_error = abs(x_star – xc)
```

```
xc =
    0.682342529296875
sol_error =
    1.472546885572523e-05
```

Numerical Computation

Bisection Method: Solution Error

If $[a, b]$ is the starting interval, then after n bisection steps, the interval $[a_n, b_n]$ has length $(b - a)/2^n$. Choosing the midpoint $x_c = (a_n + b_n)/2$ gives a best estimate of the solution r, which is within half the interval length of the true solution. After n steps of the Bisection Method, we find that

$$\text{Solution error} = |x_c - r| < \frac{b-a}{2^{n+1}}$$

Numerical Computation

Bisection Method: Solution Error

A solution is correct within *p* decimal places if the error is less than 0.5×10^{-p}.

$$f(x) = x^3 + x - 1 = 0, \; x \in [0,1]$$

TOL = 0.5e-4;

$$\frac{(b-a)}{2^{n+1}} = \frac{(1-0)}{2^{n+1}} = \frac{1}{2^{n+1}} < 0.5 \times 10^{-4} \Rightarrow n > 13.3$$

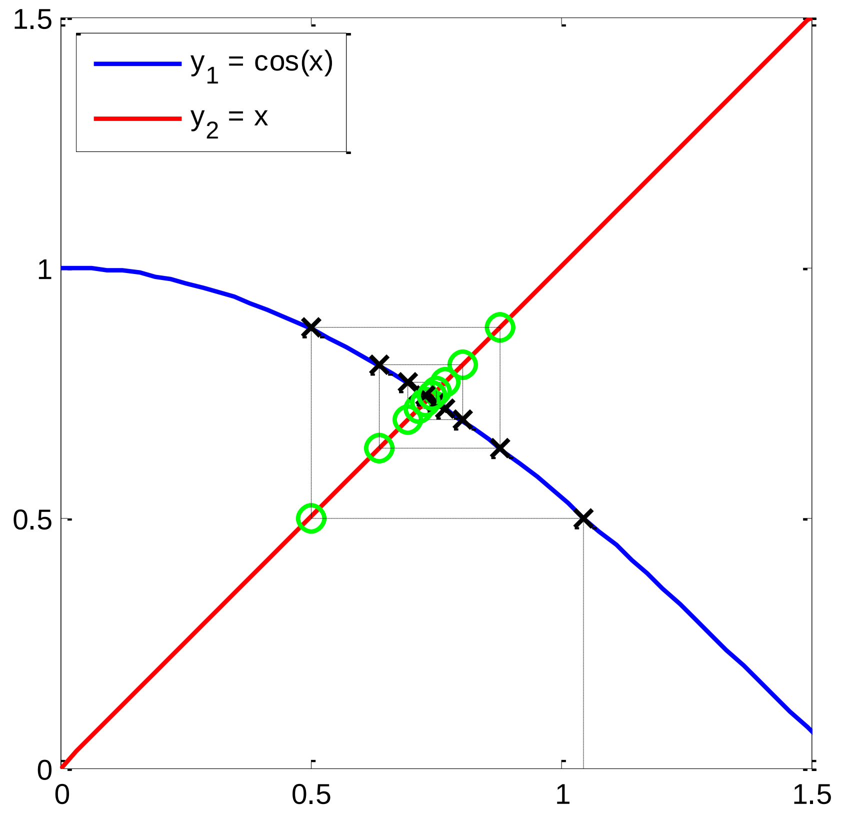
Numerical Computation

Fixed-Point Iteration Algorithm: Basic Idea

```
% cos_test_FPI.m
theta = 60 * pi/180;
cos_ini = cos(theta);
Ite = 100; % 100 iterations
for k = 1:Ite
    cos_ini = cos(cos_ini);
end
disp(sprintf('After %d iterations with θ = %s (deg):', ...
   Ite, num2str(theta * 180/pi)));
cos_ini
```

Numerical Computation

Fixed-Point Iteration Algorithm: Basic Idea



$$x = \cos(x) \quad \Rightarrow \quad x^* = 0.739085\ldots$$

Numerical Computation

- **Fixed-Point Iteration Algorithm: Basic Idea**

The real number r is a fixed point of the function g if $g(r) = r$.

Fixed-Point Iteration:

$$x_0 = \text{initial guess}$$

$$x_{i+1} = g(x_i) \text{ for } i = 0, 1, 2, \ldots$$

Numerical Computation

Fixed-Point Iteration Algorithm: Basic Idea

Can every equation $f(x) = 0$ be turned into a fixed-point problem $g(x) = x$?

Yes, and in many different ways.

$$f(x) = x^3 + x - 1 = 0, \; x \in [0,1]$$

$$x = 1 - x^3 \triangleq g_1(x)$$

$$x = \sqrt[3]{1-x} \triangleq g_2(x)$$

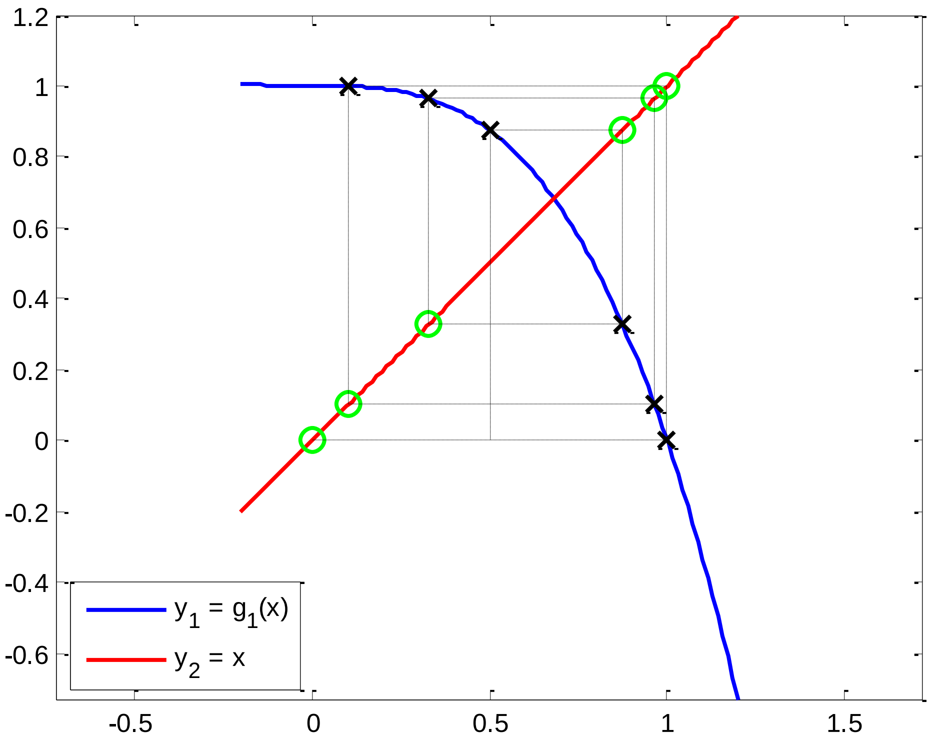
$$x = \frac{1+2x^3}{1+3x^2} \triangleq g_3(x)$$

Numerical Computation

Fixed-Point Iteration Algorithm: Basic Idea

$$f(x) = x^3 + x - 1 = 0,\ x \in [0,1]$$

$$x = 1 - x^3 \triangleq g_1(x) \qquad x_0 = 0.5$$



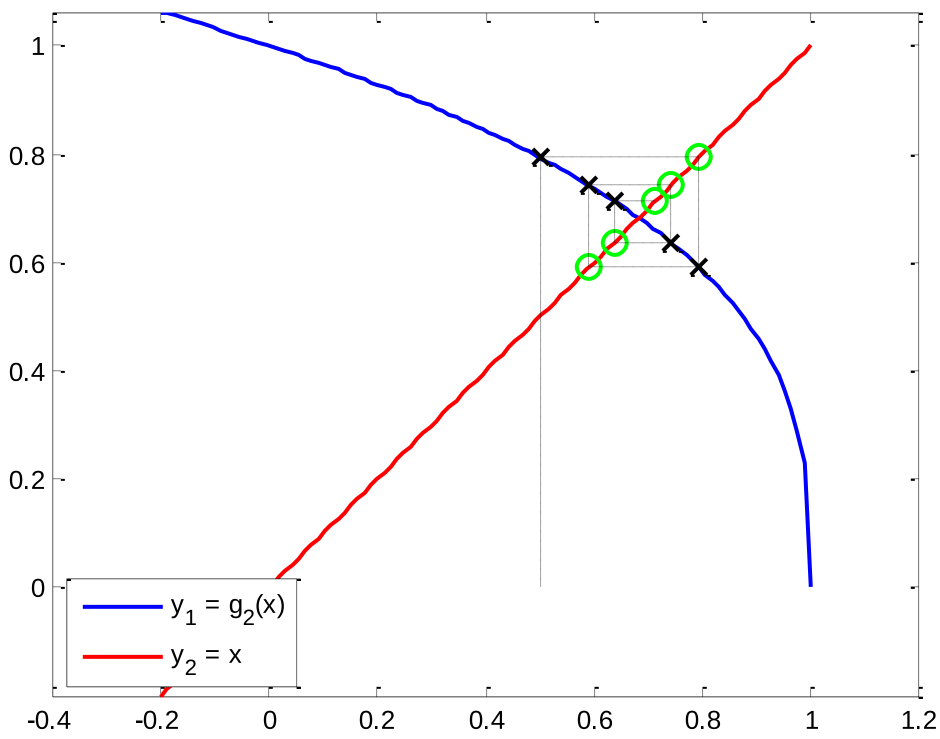
i	x_i
0	0.50000000
1	0.87500000
2	0.33007813
3	0.96403747
4	0.10405419
5	0.99887338
6	0.00337606
7	0.99999996
8	0.00000012
9	1.00000000
10	0.00000000
11	1.00000000
12	0.00000000

Numerical Computation

Fixed-Point Iteration Algorithm: Basic Idea

$$f(x) = x^3 + x - 1 = 0,\ x \in [0,1]$$

$$x = \sqrt[3]{1-x} \triangleq g_2(x) \quad x_0 = 0.5$$



i	x_i
0	0.50000000
1	0.79370053
2	0.59088011
3	0.74236393
4	0.63631020
5	0.71380081
6	0.65900615
7	0.69863261
8	0.67044850
9	0.69072912
10	0.67625892
11	0.68664554
12	0.67922234

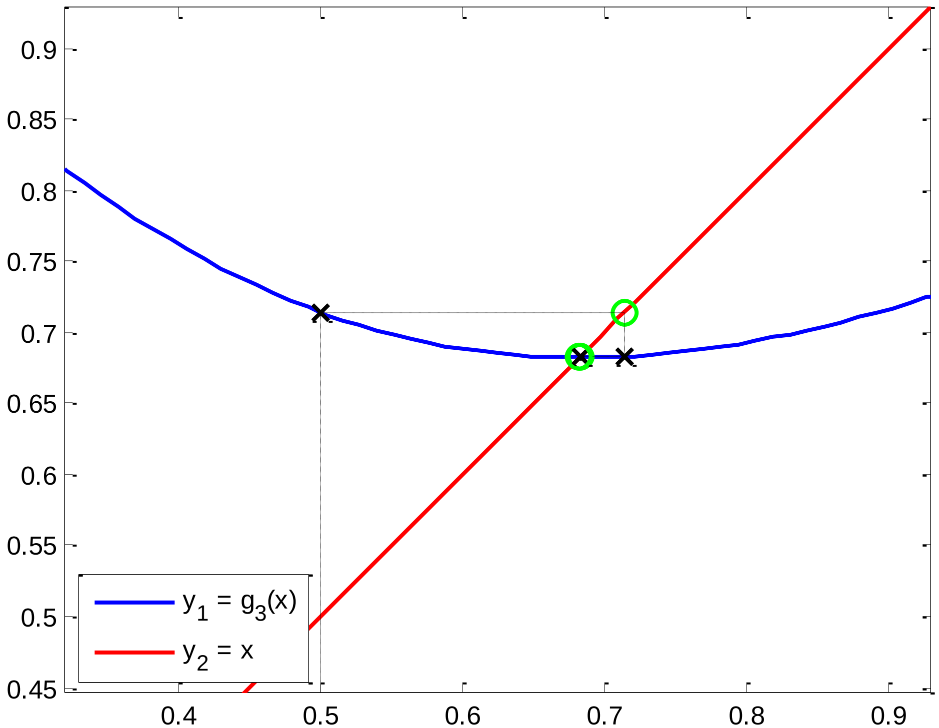
i	x_i
13	0.68454401
14	0.68073737
15	0.68346460
16	0.68151292
17	0.68291073
18	0.68191019
19	0.68262667
20	0.68211376
21	0.68248102
22	0.68221809
23	0.68240635
24	0.68227157
25	0.68236807

Numerical Computation

Fixed-Point Iteration Algorithm: Basic Idea

$$f(x) = x^3 + x - 1 = 0, \; x \in [0,1]$$

$$x = \frac{1+2x^3}{1+3x^2} \triangleq g_3(x) \quad x_0 = 0.5$$

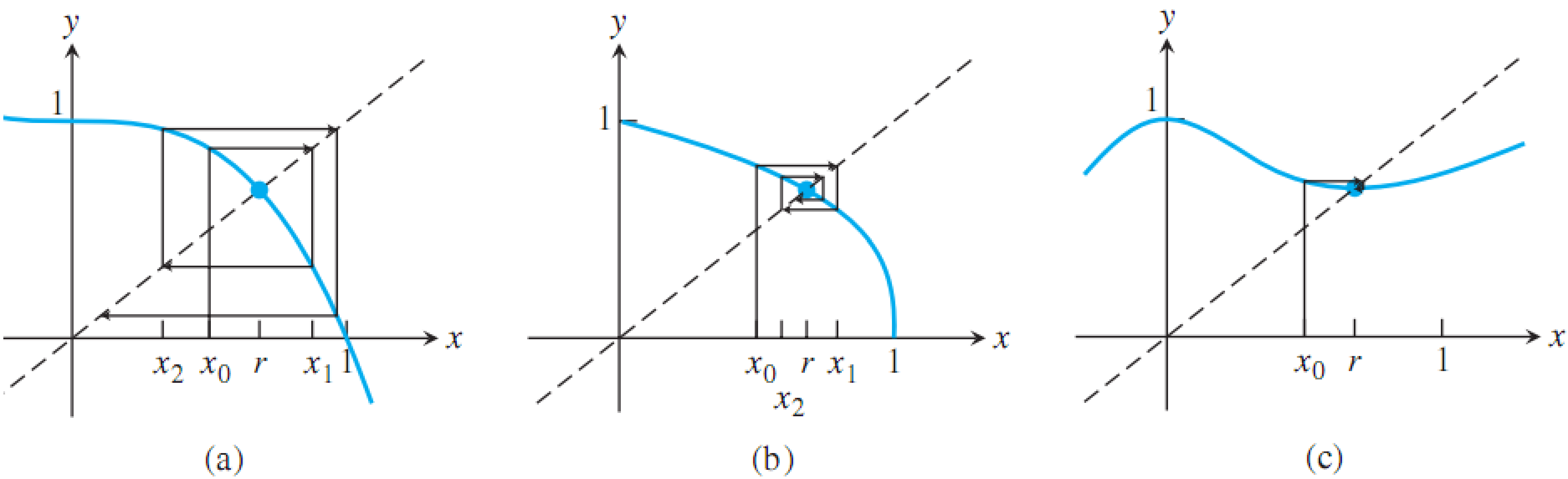


i	x_i
0	0.50000000
1	0.71428571
2	0.68317972
3	0.68232842
4	0.68232780
5	0.68232780
6	0.68232780
7	0.68232780

Numerical Computation

Fixed-Point Iteration Algorithm: Geometric view

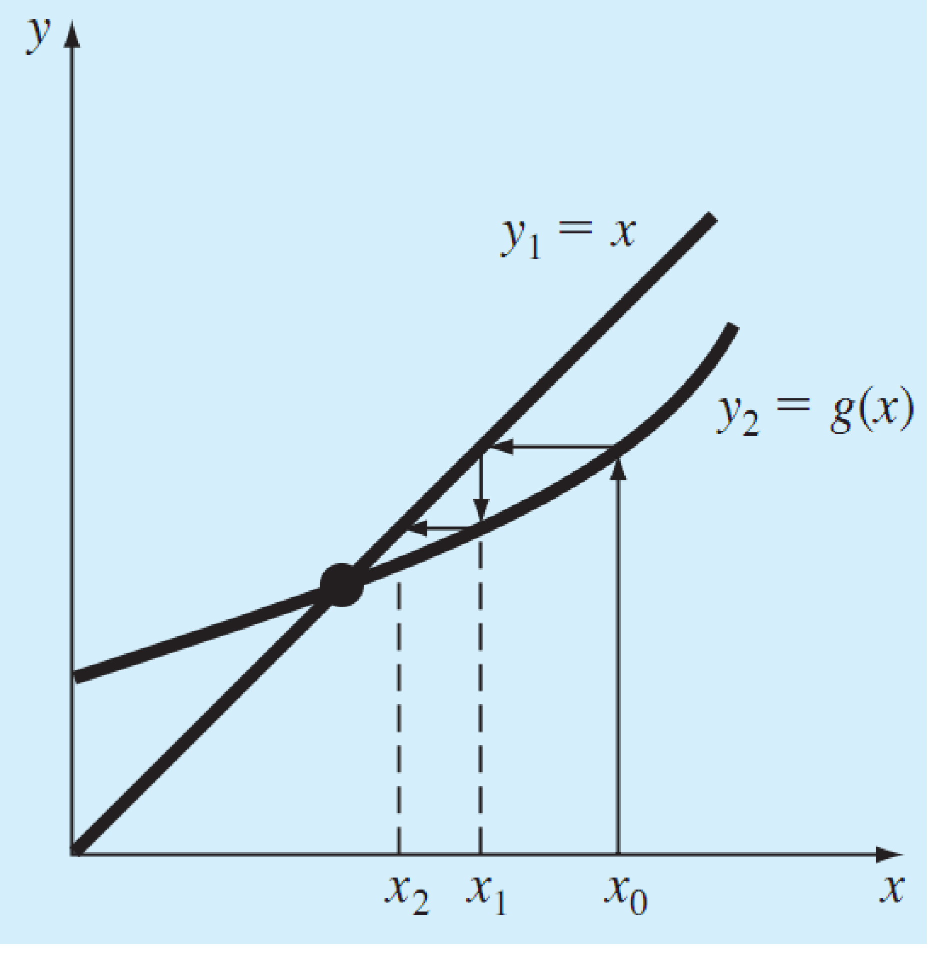
$$f(x) = x^3 + x - 1 = 0,\ x \in [0,1]$$

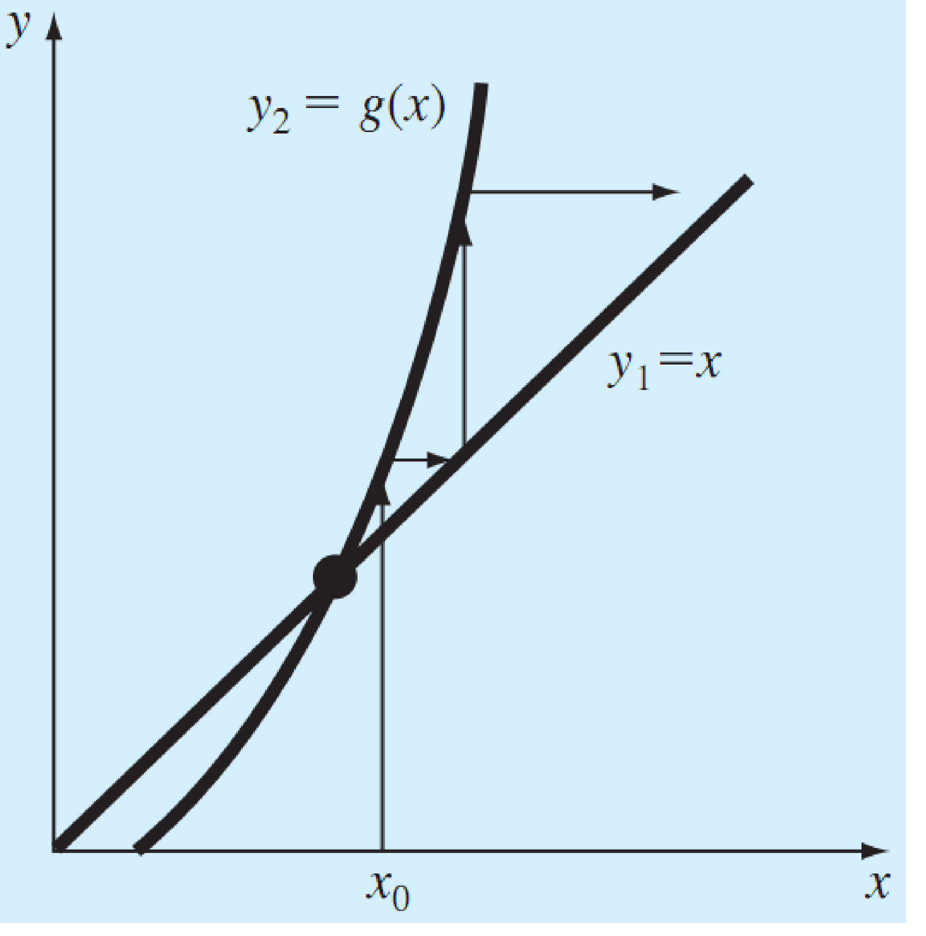


(a) $g(x) = 1 - x^3$ (b) $g(x) = (1 - x)^{1/3}$ (c) $g(x) = (1 + 2x^3)/(1 + 3x^2)$

Numerical Computation

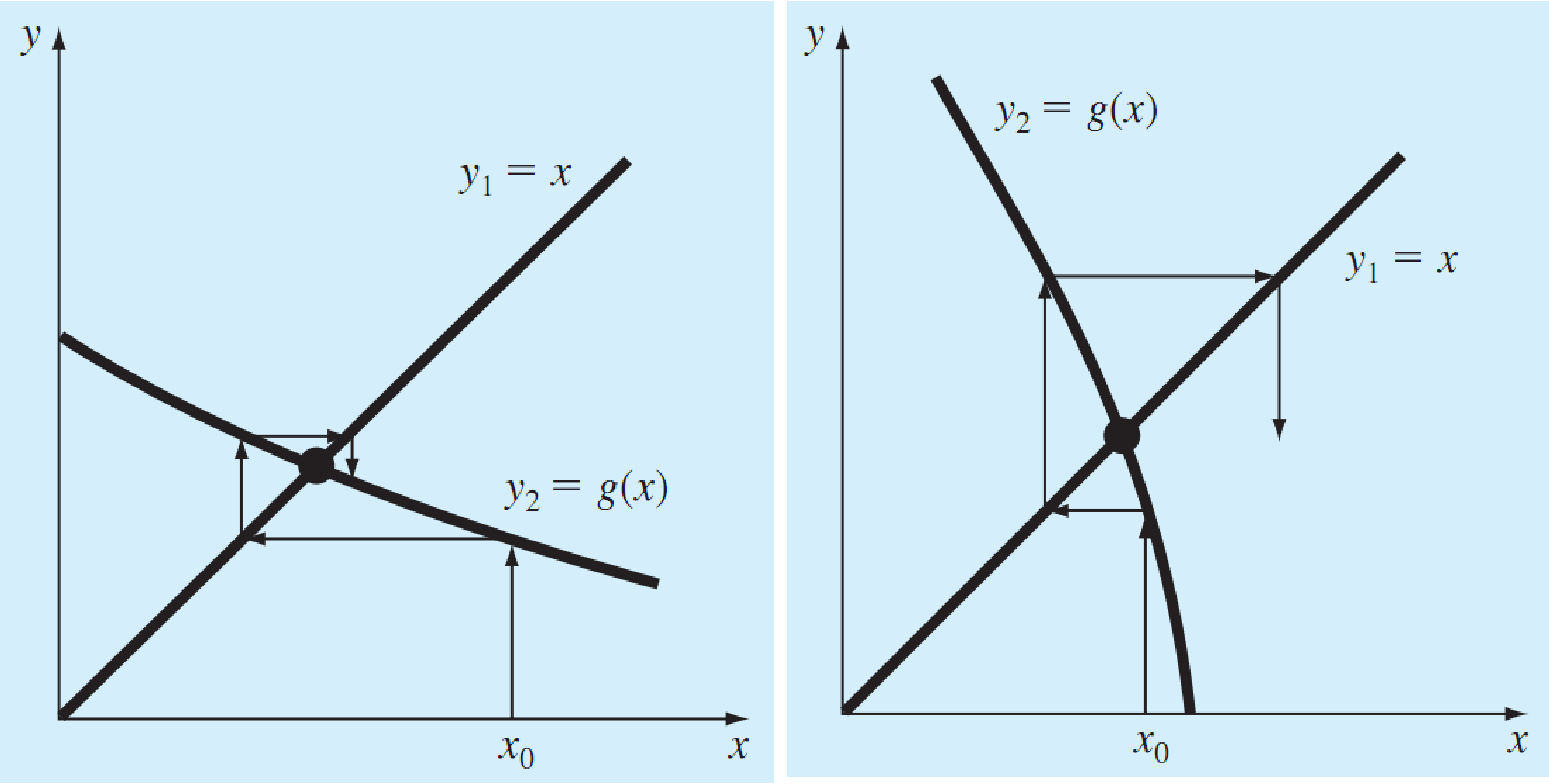
- **Fixed-Point Iteration Algorithm: Geometric view**





Numerical Computation

Fixed-Point Iteration Algorithm: Geometric view



Numerical Computation

- **Fixed-Point Iteration Algorithm: Theory**

Theorem (cf. Ref. [1], P.35):

Assume that g is continuously differentiable, that $g(r) = r$, and that $S = |g'(r)| < 1$.
Then Fixed-Point Iteration converges linearly with rate S to the fixed point r for initial guesses sufficiently close to r.

Numerical Computation

Fixed-Point Iteration Algorithm: Theory

The error at step *i*:

$$\boxed{e_i = |r - x_i|}$$

Let e_i denote the error at step i of an iterative method. If

$$\lim_{i \to \infty} \frac{e_{i+1}}{e_i} = S < 1,$$

the method is said to obey **linear convergence** with rate S.

Numerical Computation

Fixed-Point Iteration Algorithm: Example

$$f(x) = x^3 + x - 1 = 0, x \in [0,1] \qquad r \approx 0.6823$$

$$x = 1 - x^3 \triangleq g_1(x)$$

$$x = \sqrt[3]{1-x} \triangleq g_2(x)$$

$$x = \frac{1+2x^3}{1+3x^2} \triangleq g_3(x)$$

Numerical Computation

- **Fixed-Point Iteration Algorithm: Stopping criterion**

For a set tolerance, TOL, we may ask for an absolute error stopping criterion

$$|x_{i+1} - x_i| < \mathrm{TOL}$$

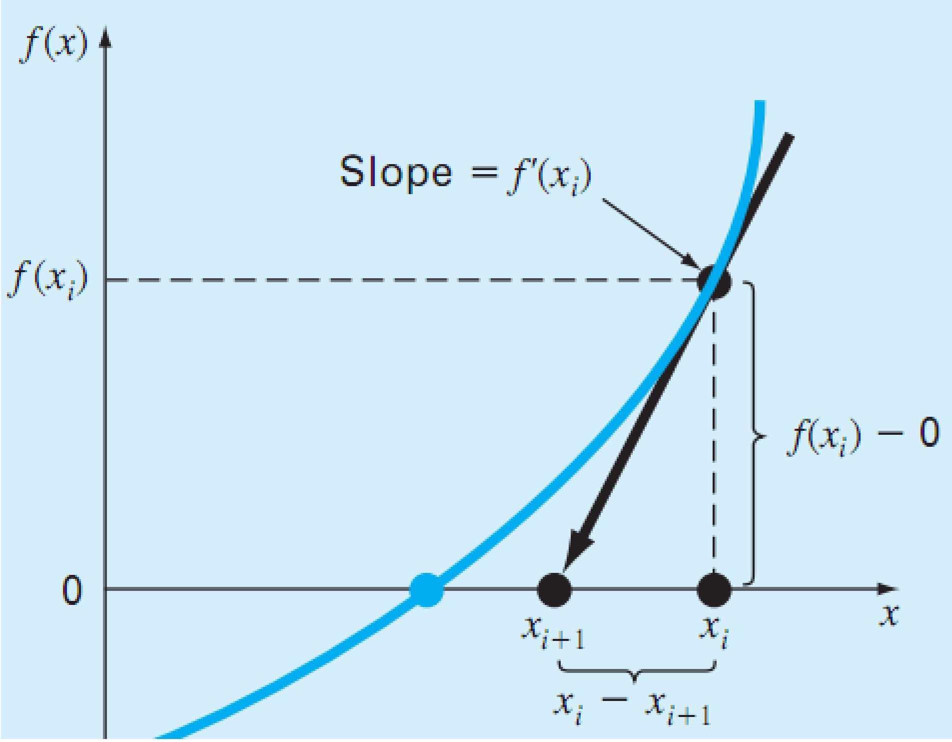
or, in case the solution is not too near zero, the relative error stopping criterion

$$\frac{|x_{i+1} - x_i|}{|x_{i+1}|} < \mathrm{TOL}.$$

Numerical Computation

Newton's Method: Basic Idea

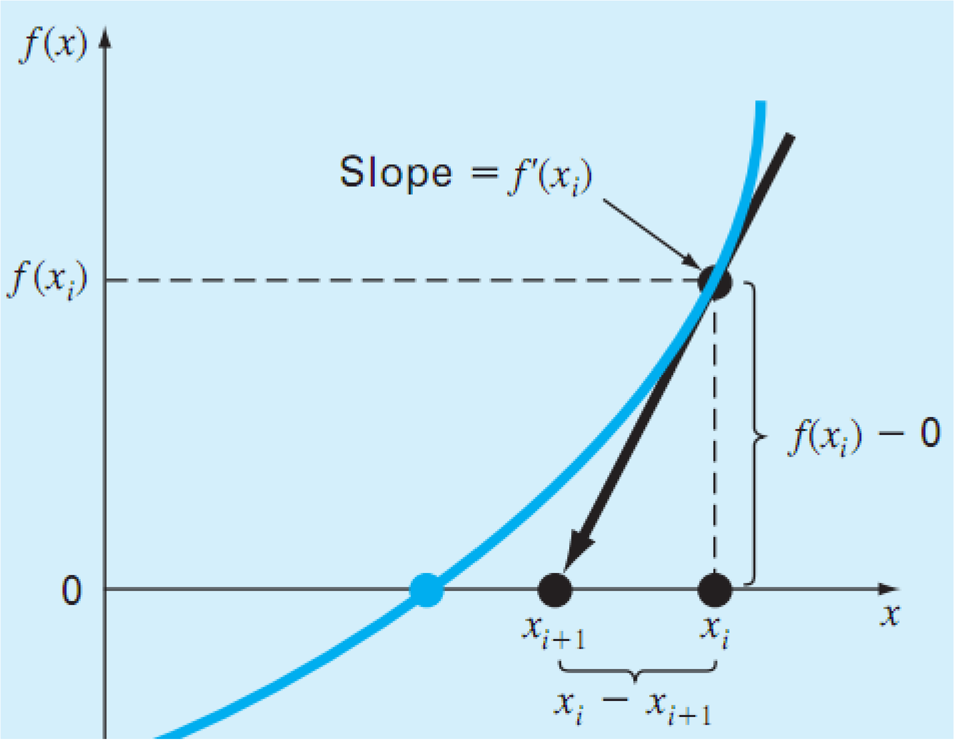
A starting guess x_0 is given, and the tangent line to the function f at x_0 is drawn. The tangent line will approximately follow the function down to the x-axis toward the root.



Numerical Computation

Newton's Method: Basic Idea

If the initial guess at the root is x_i, a tangent can be extended from the point $[x_i, f(x_i)]$.



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

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

Numerical Computation

Newton's Method: Example

$$f(x) = x^3 + x - 1 = 0,\ x \in [0,1]$$

Since $f'(x) = 3x^2 + 1$,

$$x_{i+1} = x_i - \frac{x_i^3 + x_i - 1}{3x_i^2 + 1}$$

$$= \frac{2x_i^3 + 1}{3x_i^2 + 1}.$$

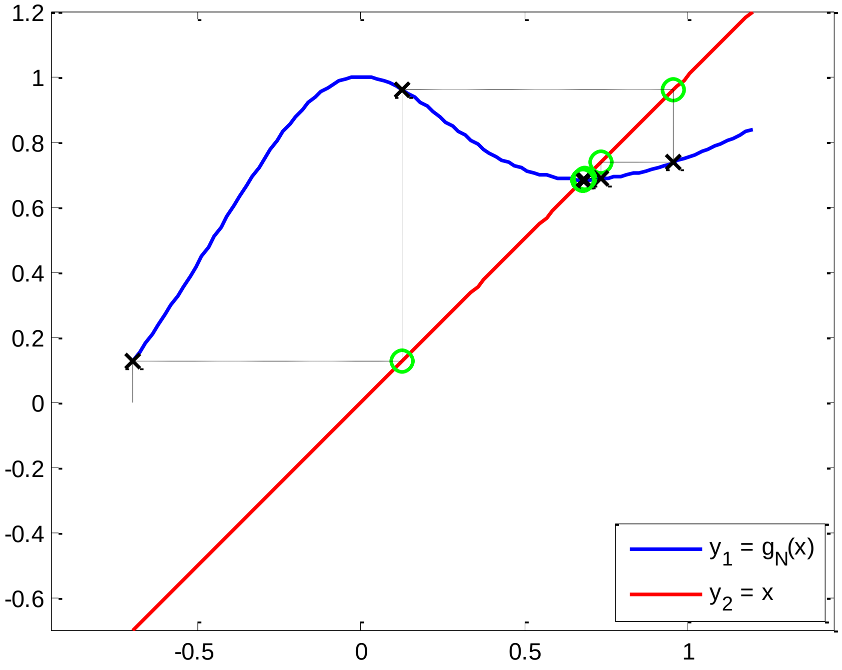
Fixed-Point Iteration: $x = \dfrac{1 + 2x^3}{1 + 3x^2} \triangleq g_3(x)$

Numerical Computation

Newton's Method: Example

$$f(x) = x^3 + x - 1 = 0, x \in [0,1]$$

$$x_{i+1} = \frac{2x_i^3 + 1}{3x_i^2 + 1} \quad \textbf{with} \quad x_0 = -0.7$$



i	x_i	$e_i = \lvert x_i - r \rvert$	e_i / e_{i-1}^2
0	−0.70000000	1.38232780	
1	0.12712551	0.55520230	0.2906
2	0.95767812	0.27535032	0.8933
3	0.73482779	0.05249999	0.6924
4	0.68459177	0.00226397	0.8214
5	0.68233217	0.00000437	0.8527
6	0.68232780	0.00000000	0.8541
7	0.68232780	0.00000000	

Numerical Computation

Newton's Method: Theory

The error at step *i*:

$$\boxed{e_i = |r - x_i|}$$

Let e_i denote the error after step i of an iterative method. The iteration is **quadratically convergent** if

$$M = \lim_{i \to \infty} \frac{e_{i+1}}{e_i^2} < \infty$$

Numerical Computation

Newton's Method: Theory

Theorem (cf. Ref. [1], P.53):

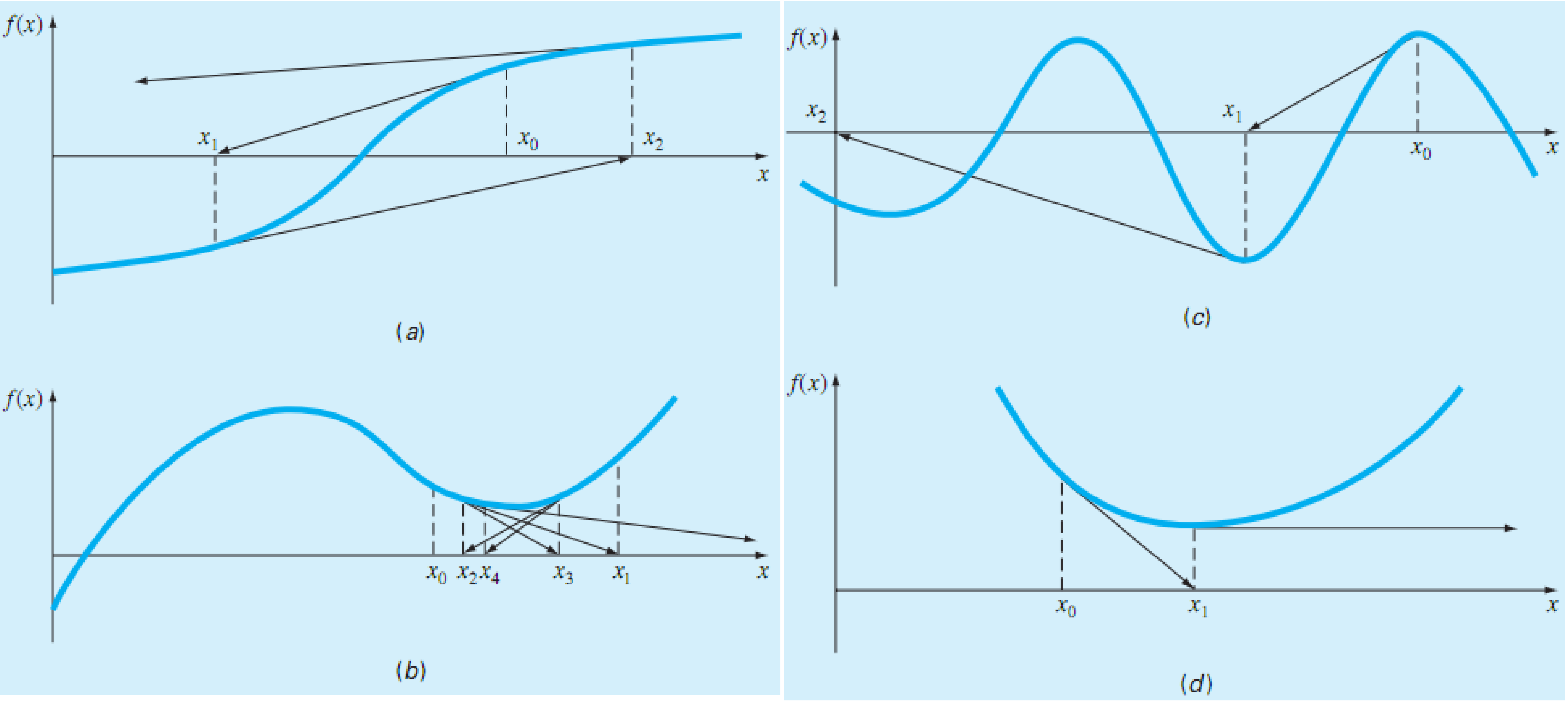
Let f be twice continuously differentiable and $f(r) = 0$. If $f'(r) \neq 0$, then Newton's Method is locally and quadratically convergent to r. The error e_i at step i satisfies

$$\lim_{i \to \infty} \frac{e_{i+1}}{e_i^2} = M$$

$$M = \frac{f''(r)}{2f'(r)}$$

Numerical Computation

Newton's Method: Four Special Cases



Poor Convergence!

Numerical Computation

The Secant Method: Basic Idea

The Secant Method is similar to the Newton's Method, but replaces the derivative by a difference quotient.

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

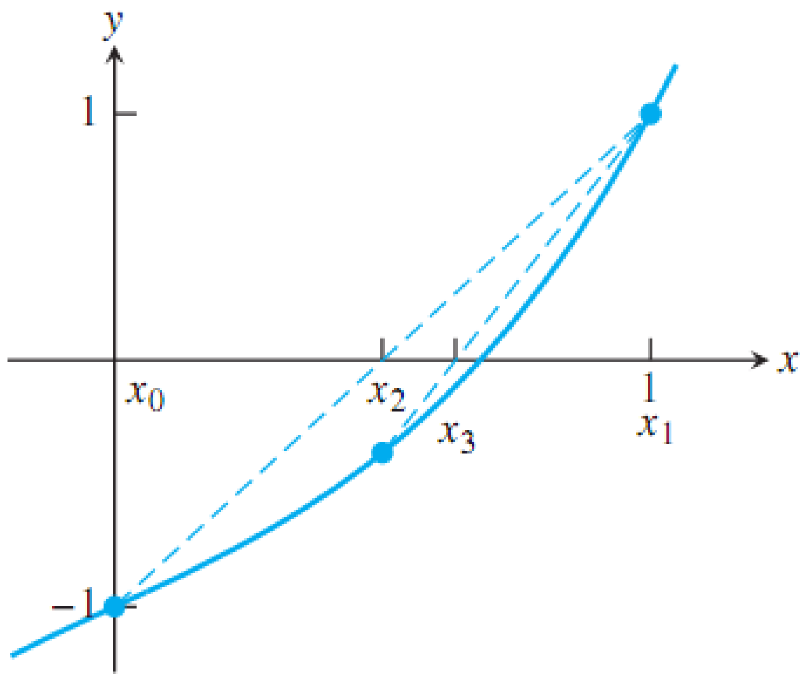
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \Rightarrow x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

Numerical Computation

The Secant Method: Example

$$f(x) = x^3 + x - 1 = 0,\ x \in [0,1]$$

$$x_{i+1} = x_i - \frac{(x_i^3 + x_i - 1)(x_i - x_{i-1})}{x_i^3 + x_i - (x_{i-1}^3 + x_{i-1})} \quad \textbf{with} \quad \begin{cases} x_0 = 0 \\ x_1 = 1 \end{cases}$$



i	x_i
0	0.00000000000000
1	1.00000000000000
2	0.50000000000000
3	0.63636363636364
4	0.69005235602094
5	0.68202041964819
6	0.68232578140989
7	0.68232780435903
8	0.68232780382802
9	0.68232780382802

Numerical Computation

- **Systems of Nonlinear Equations: Basic Idea**

An Equation: $$f(x)=0$$

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

Taylor Expansion:

$$f(x) \approx f(x_i) + f'(x_i)(x - x_i) = 0$$

Numerical Computation

Systems of Nonlinear Equations: Basic Idea

Systems of Equations:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Taylor Expansion:

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_i) + \mathbf{f}'(\mathbf{x}_i)(\mathbf{x} - \mathbf{x}_i) = \mathbf{0}$$

$$\Downarrow$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \left[\mathbf{f}'(\mathbf{x}_i)\right]^{-1} \mathbf{f}(\mathbf{x}_i)$$

Multivariate Newton's Method

Numerical Computation

- **Systems of Nonlinear Equations: Jacobian matrix**

Systems of Equations:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad \left\{ \begin{array}{l} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0. \end{array} \right.$$

Jacobian matrix of f:

$$\mathbf{J}(\mathbf{x}) \triangleq \mathbf{f}'(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Numerical Computation

- **Systems of Nonlinear Equations: Procedure**

Newton's method for the $n \times n$ nonlinear system $\mathbf{f}(\mathbf{x}) = 0$

Step 1: Choose a starting point $\mathbf{x}_0$; ε_1, ε_2; let $k = 0$;

Step 2: Calculate $\mathbf{f}(\mathbf{x}_k)$ and $\mathbf{J}(\mathbf{x}_k)$;

Step 3: Solve the linear system $\mathbf{J}(\mathbf{x}_k)\,\delta = -\,\mathbf{f}(\mathbf{x}_k)$;

Step 4: Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta$;

Step 5: Check $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \varepsilon_1$ and $\|\mathbf{f}(\mathbf{x}_k)\| < \varepsilon_2$; if they are satisfied, stop and $\mathbf{r} = \mathbf{x}_{k+1}$, else, go to Step 6;

Step 6: Set $k = k + 1$, go to Step 2.

Numerical Computation

- **Systems of Nonlinear Equations: Stopping criterion**

For a set tolerance, TOL, we may ask for an absolute error stopping criterion

$$\left\| \mathbf{x}_{i+1} - \mathbf{x}_i \right\| < \mathrm{TOL}$$

or, the relative error stopping criterion

$$\frac{\left\| \mathbf{x}_{i+1} - \mathbf{x}_i \right\|}{\left\| \mathbf{x}_i \right\|} < \mathrm{TOL}$$

Matlab built-in function *norm*

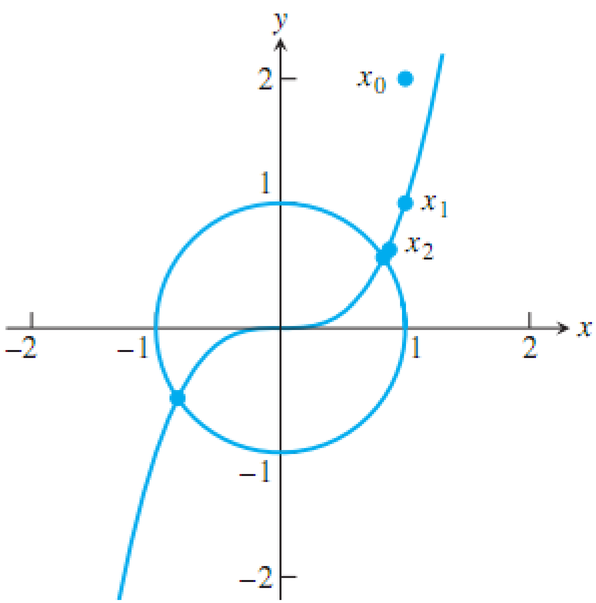
Numerical Computation

Systems of Nonlinear Equations: Example

Use Newton's Method to find the solutions of the system with the starting point $x_0 = (1,2)^T$

$$v - u^3 = 0$$
$$u^2 + v^2 - 1 = 0$$

$$\mathbf{J}(u, v) = \begin{bmatrix} -3u^2 & 1 \\ 2u & 2v \end{bmatrix}$$



step	u	v
0	1.00000000000000	2.00000000000000
1	1.00000000000000	1.00000000000000
2	0.87500000000000	0.62500000000000
3	0.82903634826712	0.56434911242604
4	0.82604010817065	0.56361977350284
5	0.82603135773241	0.56362416213163
6	0.82603135765419	0.56362416216126
7	0.82603135765419	0.56362416216126

MATLAB Built-in Functions

➔ **fzero**

Find root of continuous function of one variable

```
>> f = @(x)x.^3-2*x-5;
>> z = fzero(f,2)
```

```
>> y = @(x) fzero(@(y) exp(y) + log(y) - x^2,1);
>> y1 = y(1)
```

➔ **roots**

Find the roots of a polynomial

➔ **fsolve**

Solve system of nonlinear equations

Summary

This lecture introduces a number of methods for locating solutions x of the equation $f(x) = 0$.

- **Symbolic Computation Method**
- **Numerical Methods**
  - ✓ **Bisection Method**
  - ✓ **Fixed-Point Iteration Method**
  - ✓ **Newton's Method**
  - ✓ **The Secant Method**

Thank You !