```
% 6.1.1 A concrete example
% a supply & Demand example in Table of p. 128

c = [3 12 10; 17 18 35; 7 10 24];
x = [4 0 0; 6 6 0; 0 3 5];
total = c .* x     % pointwise operation

sum(total)
sum(sum( total ))

%==============================================================

% 6.1.2 Creating matrices
% 1-D subscript is column rank first
a = [1 2; 3 4];
x = [5 6];
a = [a; x]

a(3,2)
a(5)

a(3,3)

a(3,3) = 7

%==============================================================

% 6.1.4 Transpose

a = [1:3; 4:6]
b = a'


%==============================================================

% 6.1.5 The colon operator

a = [1:3; 4:6; 7:9]
```

```
a(2:3,1:2)

aa= floor( (rand(6,6).*10))+1
bb=aa(1:2:5,1:3) % colon generate a subscript vector

a(3,:) % ':' means all elements

a(1:2,2:3) = ones(2)    % a(1:2,2:3) is a 2*2 matrix


% To construct a table

format long
x = [0:30:180]'; % row vector transpose to a column vector
trig(:,1) = x;
trig(:,2) = sin(pi/180*x);
trig(:,3) = cos(pi/180*x);

trig
  % replaces the first and third columns of a by the fourth and second columns
% of b (a and b must have the same number of rows).

a = [1:3; 4:6; 7:9]
b = [11:14; 15:18; -4:-1]
a(:,[1 3]) = b(:,[4 2])

% A famous opertion in Linear algebra is the Gauss reduction
a=[1 -1 2; 2 1 -1; 3 0 1]
a(2,:) = a(2,:) - a(2,1)*a(1,:)

% The keyword 'end' refers to the last row or column of an array

r = ones(1,8)
sum(r(3:end)) % 'end' means till the last one.
```

```
% a(:) is different for the right or left,
% on the right: straight out to a single column vector.
a=[1 2; 3 4]
b=a(:)

% on the left, a(:) reassign a matric which is already exist, # of element
% must be the same.
a=zeros(3,2)
a(:)=[ 1:6]

  b = [1:3; 4:6]
a=zeros(3,2)
a(:) = b

  % 1-D sequence ranking of the matrix b is 1 4 2 5 3 6
  % reshape it to an 3*2 matrix note that column first
c = reshape(b,3,2)

a(:) = -1



%============================================================
% 6.1.6 Duplicating rows and columns: tiling by 'repmat'

a = [1 2 3]
a=[ 1 2 ; 3 4];
b = repmat(a, [2 3]) % repeat matric a twice in the row & three times in the column

% alternative syntax
c = repmat(a, 2, 3)

d = repmat(a, [4 2])
e = repmat(a, 4, 2)
```

```
%==============================================================
% 6.1.7 Deleting rows and columns

a = [1:3; 4:6; 7:9]
a(:,2) = [ ] % notes that it is different from a(:,2) = 0

% You cannot delete a single element from a matrix
a(1,2) = [ ]

% You can delete a sequence of elements from a matrix
% and reshape the remaining elements into a row vector

a = [1:3; 4:6; 7:9]
a(2:2:6) = [ ]

% You can use logical vectors to extract a selection of rows or columns from a matrix,

a = [1:3; 4:6; 7:9]
cc=logical([1 0 1])
b = a(:, logical([1 0 1]))

c = a(:, [1 3]) % [1 3] is the subsript vector of the matrix

%==============================================================
% 6.1.8 Elementary matrices

a = zeros(3,4)
a = ones(3,4)
a = rand(3,4)
a = eye(3)

% As an example, eye may be used to construct a tridiagonal matrix as follows.

a = 2 * eye(5);
a(1:4, 2:5) = a(1:4, 2:5) - eye(4);
a(2:5, 1:4) = a(2:5, 1:4) - eye(4)
```

```
%=====================================================================
% 6.1.9 Specialized matrices

% pascal(n) generates a Pascal matrix of order n.

a = pascal(4)

b = magic(4) % equal sum along any rows or columns


%=====================================================================
% 6.1.10 Using MATLAB functions with matrices

% For each column of a where all the elements are true (non-zero) all returns '1', otherwise it returns 0.
a = [1 0 1; 1 1 1; 0 0 1]
b = all(a)

% To test if all the elements of a are true, use all twice.
c = all(all(a))

d = any(a)
e = any(any(a))


%=====================================================================
% 6.1.11 Manipulating matrices

% diag extracts or creates a diagonal.
a = pascal(4)
b = diag(a)

% fliplr flips from left to right.
a = pascal(4)
b = fliplr(a)

% flipud flips from top to bottom.
a = pascal(4)
```

```
b = flipud(a)

% rot90 rotates.
a = pascal(4)
b = rot90(a)

% tril extracts the lower triangular part,
a = pascal(4)
b = tril(a)

% triu extracts the upper triangular part.
a = pascal(4)
b = triu(a)

%=============================================================
% 6.1.12 Pointwise (Array, element-by-element) operations on matrices

a = [1:3; 4:6]
b = a .^ 2
c = sin(a)


%=============================================================
% 6.1.13 Matrices and for

% the index v takes on the value of each column of the matrix expression a in turn.

a = [1:3; 4:6; 7:9]

for v = a(:,1:2)
disp(v')
end

% the index v takes on the value of each row of the matrix expression a in turn.
for v = a'
disp(v')
end
```

```
%=================================================================
% 6.1.15 Vectorizing nested fors: loan repayment tables
%%    forming the table of repayments for a ;oan of $1000 over 15 20 and 25 yrs
%    rate%     15 yrs     20 yrs     25 yrs
%      10       10.75       9.65       9.09
%      11       11.37      10.32       9.80
%      12       12.00      11.01      10.53
%      13       12.65      11.72      11.28
%% Exercise to write the expression in p.140 by matlab code

% Method 1:
A = 1000; % amount borrowed
n = 12; % number of payments per year
disp ([' rate%     15 yrs     20 yrs     25 yrs']);
for r = 0.1 : 0.01 : 0.2
    fprintf( '%4.0f%', 100 * r );
    for k = 15 : 5 : 25
        temp = (1 + r/n) ^ (n*k);
        P = r * A * temp / (n * (temp - 1));
        fprintf( '%10.2f', P );
    end;
    fprintf( '\n' ); % new line
end;

% The inner loop can easily be vectorized; the following code uses only one for:

% Method 2
format short
A = 1000; % amount borrowed
n = 12; % number of payments per year
disp (['    rate%          15 yrs     20 yrs     25 yrs']);
for r = 0.1 : 0.01 : 0.2
    k = 15 : 5 : 25;         % Now k is a vector 1*3
    temp = (1 + r/n) .^ (n*k); % temp is a vector with the same dim.
    P = r * A * temp / n ./ (temp - 1);
    disp( [100 * r P] ); % [100 * r P]    is a 1*4 vector
end;
```

```
 % The really tough challenge, however, is to vectorize the outer loop as well.

% Method 3
A = 1000; % amount borrowed
n = 12; % number of payments per year
r = [0.1:0.01:0.2]'    % r is a 11*1 matrix
% Now change this into a table with 3 columns each equal to r:
r = repmat(r, [1 3]) % r is 11*3 matrix

k = 15:5:25 % k is 3*1 matrix
k = repmat(k, [11 1])    % k is a 11*3 matrix
% r (11*3) matrix =
%
%         0.1000        0.1000        0.1000
%         0.1100        0.1100        0.1100
%         0.1200        0.1200        0.1200
% k (11*3) matrix =
%
%         15        20        25
%         15        20        25
%         15        20        25

% show the value of r & k

temp = (1 + r/n) .^ (n * k);
P = r * A .* temp / n ./ (temp - 1)

%% Exercise 1: Ex 6.1 in p.161
% Exercise 2: Do these methods for the Ex. 2-28 in p. 81 for L=10000:10000:50000; and
r=0.1:0.02:0.2, P=1000


%================================================================
% 6.1.16 Multi-dimensional arrays

a = [1:2; 3:4]

% You can add a third dimension to a with
```

```
% then a is a 3-dim matrix
% with a(:,:,1)=[1:2; 3:4]
a(:,:,2) = [5:6; 7:8]

%==============================================================
% 6.2 MATRIX OPERATIONS : check Table 6.2 for the matrix operation
% 6.2.1 Matrix multiplication

a = [1 2; 3 4]
b = [5 6; 0 -1]

% Note the important difference between the pointwise operation a .* b
% and the matrix operation a * b

c = a*b
d = a.*b

e = [2 3]'
f = a * e
%==============================================================
% 6.2.2 Matrix exponentiation : check the table 6.2

a = [1 2; 3 4]
b = a^2    % it means a*a & it is different from the pointwise operation a.^2
c = a*a

% Again, note the difference between the pointwise operation a .^ 2 and the matrix
operation a ^ 2.
d = a.*2
e = a.*a

%==============================================================
% 6.3 OTHER MATRIX FUNCTIONS

a = [5 11; 11 25]

b = det(a) % determinate of a
c = eig(a)    % eigenvalues of a
```

```
d = inv(a)      % inverse of a
% Singular value decomposition.of a; if a is a square matrix, then it is
% just a eigenvalue decomposition
b=repmat(a,2,1)
[U,S,V] = svd(b)

%=============================================================
% 6.4 POPULATION GROWTH: LESLIE MATRICES:    Textbook p.147

% Leslie matrix population model
n = 3;
L = zeros(n); % all elements set to zero
L(1,2) = 9;
L(1,3) = 12;
L(2,1) = 1/3;
L(3,2) = 0.5;
x = [0 0 1]'; % remember x must be a column vector!
for t = 1:24
        x = L * x;
        p(t) = sum(x);
        disp( [t x' sum(x)] ) % x¦ is a row
end

figure, plot(1:15, p(1:15)), xlabel('months'), ylabel('rabbits')
hold, plot(1:15, p(1:15),'o')

[a b] = eig(L)
```

```
%==================================================================
% 6.5 MARKOV PROCESSES : A random walk process
% check the textbook p. 150
% After few steps the random walk ended at either home or café
```

	Home	2	3	4	5	Café
Home	1.0000	0.3333	0	0	0	0
2	0	0	0.3333	0	0	0
3	0	0.6667	0	0.3333	0	0
4	0	0	0.6667	0	0.3333	0
5	0	0	0	0.6667	0	0
Café	0	0	0	0	0.6667	1.0000

```
n = 6;
P = zeros(n); % all elements set to zero
for i = 3:6
    P(i,i-1) = 2/3;
    P(i-2,i-1) = 1/3;
end
P(1,1) = 1;
P(6,6) = 1;
x = [0 1 0 0 0 0]'; % remember x must be a column vector!
for t = 1:50
    x = P * x;
    disp( [t x'] )
end
```

```
%==============================================================
% 6.6 LINEAR EQUATIONS


A = [3 2 -1; -1 3 2; 1 -1 -1]
b = [10 5 -1]'
x = A \ b    % solve a linear equation Ax=b

% The same solution can be obtained by the following equation
z = inv(A) * b
% A \ b is actually more accurate and efficient

% 6.6.2 The residual
r = A*x - b

% 6.6.3 Over-determined systems
% more equations then unknowns (No solution case in L.A,)
% one can find the minimum meansquare solution (MMSE)
% which lead the rtotal = sqrt(r'*r), r = A*x - b minimum
A = [1 -1; 0 1; 1 0]
b = [0 2 1]'
x = A \ b

r = A*x - b
rtotal = sqrt(r'*r)
```

Example of overdetermined system. For the least square fitting of a straight line.

When $Ax = b$ has no solution, multiply by A^{T} and solve $A^{\mathrm{T}}A\hat{x} = A^{\mathrm{T}}b$.

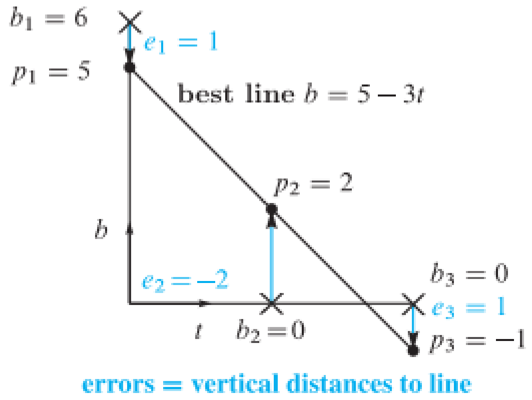
Example 1 A crucial application of least squares is fitting a straight line to m points. Start with three points: *Find the closest line to the points* $(0, 6)$, $(1, 0)$, *and* $(2, 0)$.

No straight line $b = C + Dt$ goes through those three points. We are asking for two numbers C and D that satisfy three equations. Here are the equations at $t = 0, 1, 2$ to match the given values $b = 6, 0, 0$:

$t = 0$	The first point is on the line $b = C + Dt$ if	$C + D \cdot 0 = 6$
$t = 1$	The second point is on the line $b = C + Dt$ if	$C + D \cdot 1 = 0$
$t = 2$	The third point is on the line $b = C + Dt$ if	$C + D \cdot 2 = 0.$

This 3 by 2 system has *no solution*: $b = (6, 0, 0)$ is not a combination of the columns $(1, 1, 1)$ and $(0, 1, 2)$. Read off A, x, and b from those equations:

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \quad x = \begin{bmatrix} C \\ D \end{bmatrix} \quad b = \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix} \quad Ax = b \text{ is } \textit{not} \text{ solvable.}$$




```
% 6.6.4 Under-determined systems (Infinity many solutions)

A = [1 -1 5; 3 1 2] % A 2*3 matrix; two equations, 3 unknown;
b = [2 1]'
x = A \ b



% 6.6.5 Ill conditioning

A = [10 7 8 7; 7 5 6 5; 8 6 10 9; 7 5 9 10]
b = [32 23 33 31]'
x = A \ b

% perturbed b vector
b = [32.1 22.9 32.9 31.1]'
x = A \ b

% rcond ; is an estimate for the reciprocal of the
%         condition of X in the 1-norm obtained by the LAPACK
%         condition estimator. If X is well conditioned, rcond(X)
%         is near 1.0. If X is badly conditioned, rcond(X) is
%         near EPS.
rcond(A)
```