Rutgers University
School of Engineering

Fall 2011

14:440:127 - Introduction to Computers for Engineers

Sophocles J. Orfanidis
ECE Department
orfanidi@ece.rutgers.edu

week 3

Weekly Topics

Week 1 - Basics – variables, arrays, matrices, plotting (ch. 2 & 3)
Week 2 - Basics – operators, functions, program flow (ch. 2 & 3)
Week 3 - Matrices (ch. 4)
Week 4 - Plotting – 2D and 3D plots (ch. 5)
Week 5 - User-defined functions (ch. 6)
Week 6 - Input-output formatting – fprintf, sprintf (ch. 7)
Week 7 - Program flow control & relational operators (ch. 8)
Week 8 - Matrix algebra – solving linear equations (ch. 9)
Week 9 - Structures & cell arrays (ch. 10)
Week 10 - Symbolic math (ch. 11)
Week 11 - Numerical methods – data fitting (ch. 12)
Week 12 – Selected topics

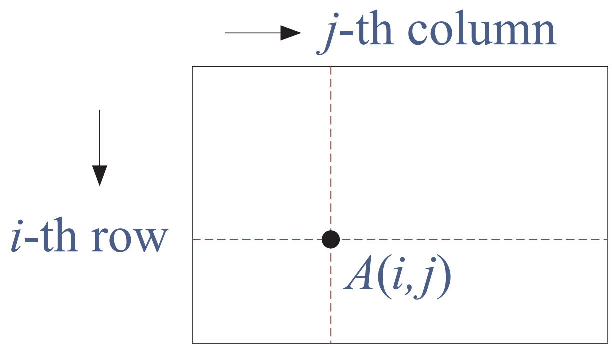
Textbook: H. Moore, *MATLAB for Engineers*, 2nd ed., Prentice Hall, 2009

Matrix Manipulation

defining matrices
accessing matrix elements
colon operator, submatrices
transposing a matrix
changing/adding/deleting entries
concatenating matrices
special matrices
diagonals, block-diagonal matrices
replicating and reshaping matrices
element-wise operations
functions of matrices (element & column operations)
meshgrid, ndgrid
examples: DTMF keypad, Taylor series, polynomials

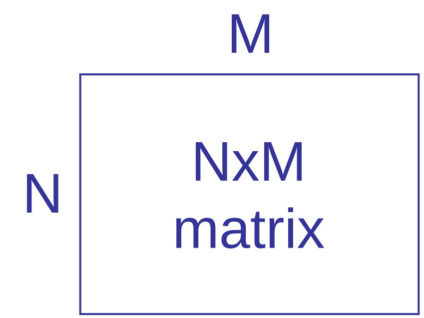
defining matrices

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$



matrix indexing convention

```
>> A = [1 2 3; 2 0 4; 0 8 5]
A =
     1     2     3
     2     0     4
     0     8     5
```




```
>> size(A)
ans =
     3     3
```

```
% [N,M] = size(A), NxM matrix
```

column dimension   row dimension

accessing matrix elements

```
>> A(1,1)        % 11 matrix element
ans =
     1
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

```
>> A(2,3)        % 23 matrix element
ans =
     4
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

```
>> A(:,2)        % second column
ans =
     2
     0
     8
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

```
>> A(3,:)         % third row
ans =
     0     8     5
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

concatenating columns

```
>> A = [1 2 3; 2 0 4; 0 8 5]
A =
      1     2     3
      2     0     4
      0     8     5
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

```
>> A(:)      % concatenate columns
ans =
      1
      2
      0
      2
      0
      8   <-- A(6)
      3
      4
      5   <-- A(9)
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

column-wise indexing

concatenating rows

```
B = A';
B(:)
```

building a matrix column-wise

```
>> A = zeros(3);

>> A(:) = [1 2 0  2 0 8  3 4 5]

A =
     1     2     3
     2     0     4
     0     8     5
```

define desired size

enter elements in a row (or column)

elements are re-arranged column-wise

sub-matrices

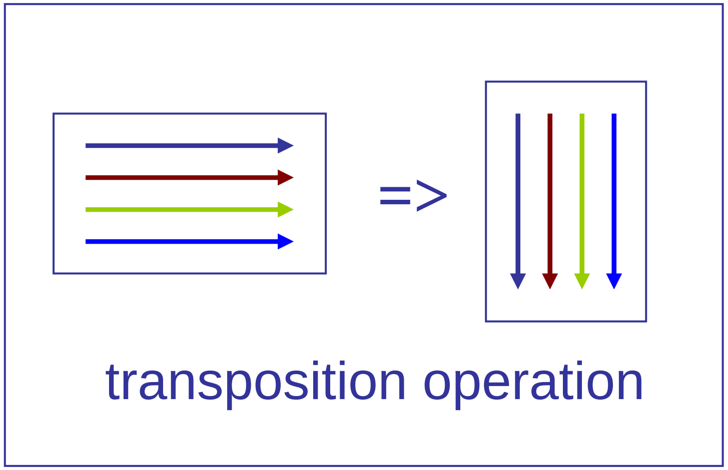
```
A = [ 2    4    1    3    5
      8    6    7    4    9
      3    2    5    2    1
      5    6    1    8    4 ];

A(3:4, 2:4)
ans =
      2    5    2
      6    1    8

A(1:3, [1,5])
ans =
      2    5
      8    9
      3    1
```

$$A = \begin{bmatrix} 2 & 4 & 1 & 3 & 5 \\ 8 & 6 & 7 & 4 & 9 \\ 3 & 2 & 5 & 2 & 1 \\ 5 & 6 & 1 & 8 & 4 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & 4 & 1 & 3 & 5 \\ 8 & 6 & 7 & 4 & 9 \\ 3 & 2 & 5 & 2 & 1 \\ 5 & 6 & 1 & 8 & 4 \end{bmatrix}$$

transposing a matrix

```
>> A = [1 2 3 4; 2 0 5 6; 0 8 7 9]    % size 3x4
A =
     1     2     3     4
     2     0     5     6
     0     8     7     9

>> A'                                 % size 4x3
ans =
     1     2     0
     2     0     8
     3     5     7
     4     6     9
```



adding/deleting rows or columns

```
>> A = [1 2 3; 2 0 4; 0 8 5]

>> A(5,:) = [7 8 9]      % add a fifth row
A =
     1     2     3
     2     0     4
     0     8     5
     0     0     0
     7     8     9
```

4^{th} row is automatically allocated

```
>> A(:,2) = []           % delete second column
A =
     1     3
     2     4
     0     5
     0     0
     7     9
```

[ ] denotes an empty 0x0 matrix

alternatively, redefine A by omitting its second column:

```
>> A = A(:,[1,3]);
```

replacing rows or columns

```
>> A = [1 2 3; 2 0 4; 0 8 5]
A =
     1     2     3
     2     0     4
     0     8     5

>> A(:,2) = [20 30 40]'    % replace second column
A =
     1    20     3
     2    30     4
     0    40     5

>> A(3,:) = [50 60 70]     % replace third row
A =
     1    20     3
     2    30     4
    50    60    70
```

concatenating matrices

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

```
>> A = [1 2; 3 4];
>> B = [5 6; 7 8];

>> C = [A, B]
C =
     1     2     5     6
     3     4     7     8
```

A,B must have same number of rows

```
>> C = [A; B]
C =
     1     2
     3     4
     5     6
     7     8
```

A,B must have same number of columns

appending columns or rows

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 7 \\ 7 \\ 7 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 8 \\ 8 \\ 8 \end{bmatrix}$$

```
>> A = [1 2; 3 4; 5 6];
>> b = [7; 7; 7];
>> c = [8 8 8]';

>> B = [A,b,c]
B =
        1    2    7    8
        3    4    7    8
        5    6    7    8

>> C = [b,A,c]
C =
        7    1    2    8
        7    3    4    8
        7    5    6    8
```

```
>> D = [A; [7 7];]
>> E = [[8 8]; A]
D =
        1      2
        3      4
        5      6
        7      7
E =
        8      8
        1      2
        3      4
        5      6
```

special matrices

```
eye(3)    % 3x3 identity matrix
ans =
     1     0     0
     0     1     0
     0     0     1
```

```
>> help eye
>> help zeros
>> help ones
```

```
zeros(3)  % 3x3 matrix of zeros
ans =
     0     0     0
     0     0     0
     0     0     0
```

general usage:

```
eye(N,M)
zeros(N,M)
ones(N,M)
```

```
ones(3)   % 3x3 matrix of ones
ans =
     1     1     1
     1     1     1
     1     1     1
```

for more information on elementary matrices see:

```
>> help elmat

  Elementary matrices and matrix manipulation.

  Elementary matrices.
    zeros       - Zeros array.
    ones        - Ones array.
    eye         - Identity matrix.
    repmat      - Replicate and tile array.
    linspace    - Linearly spaced vector.
    logspace    - Logarithmically spaced vector.

    etc.

>> help gallery    % various test matrices
```

diagonals

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> d = diag(A)          % main diagonal
d =
     1
     5
     9

>> d = diag(A,-1)       % first sub-diagonal
d =
     4
     8
```

```
>> help diag
```

how to make a diagonal matrix

```
>> d = [4 5 6];          % or, column d = [4 5 6]';

A = diag(d)              % d is main diagonal
A =
     4     0     0
     0     5     0
     0     0     6

>> d = [4 5];

A = diag(d,1)            % d is first upper-diagonal
A =
     0     4     0
     0     0     5
     0     0     0
```

```
>> A = [1 2; 3 4]; B = [5 6; 7 8];
>> C = [9 8 7; 6 5 4; 3 2 1];

>> blkdiag(A,B)
ans =
     1     2     0     0
     3     4     0     0
     0     0     5     6
     0     0     7     8

>> blkdiag(A,B,C)
ans =
     1     2     0     0     0     0     0
     3     4     0     0     0     0     0
     0     0     5     6     0     0     0
     0     0     7     8     0     0     0
     0     0     0     0     9     8     7
     0     0     0     0     6     5     4
     0     0     0     0     3     2     1
```

how to make block-diagonal matrices

matrix dimensions expand as necessary

replicating matrices

```
>> A=[1 2; 3 4]
A =
     1     2
     3     4

>> repmat(A,3,4)
ans =
     1     2     1     2     1     2     1     2
     3     4     3     4     3     4     3     4
     1     2     1     2     1     2     1     2
     3     4     3     4     3     4     3     4
     1     2     1     2     1     2     1     2
     3     4     3     4     3     4     3     4
```

reshaping a matrix or a vector

B = reshape(A,P,Q)

reshapes an NxM matrix into a PxQ matrix (must have PxQ=NxM)

B is formed column-wise from the elements of A

```
>> a = [1 2 3 4 5 6];

>> reshape(a,2,3)
ans =
     1     3     5
     2     4     6

>> reshape(a,3,2)
ans =
     1     4
     2     5
     3     6

>> reshape(a,6,1)
ans =
     1
     2
     3
     4
     5
     6
```

reshaping a matrix or a vector

```
A = [1    5    9
     2    6    5
     3    7    0
     4    8    4];

>> reshape(A,3,4)
ans =
     1    4    7    5
     2    5    8    0
     3    6    9    4


>> reshape(A,2,6)
ans =
     1    3    5    7    9    0
     2    4    6    8    5    4
```

```
>> reshape(A,6,2)
ans =
     1      7
     2      8
     3      9
     4      5
     5      0
     6      4
```

element-wise operations

```
>> A = [1 2; 3 4]
A =
     1     2
     3     4

>> [A, A.^2; 2.^A, A^2]      % form sub-blocks
ans =
     1     2     1     4
     3     4     9    16
     2     4     7    10      % note A.^2 ~= A^2
     8    16    15    22

>> B = 10.^A;
>> [B, log10(B)]
ans =
       10         100            1            2
     1000       10000            3            4
```

element-wise operations

```
>> A=[1 4; 8 2], B=[1 2; 2 1]
A =
     1     4
     8     2
B =
     1     2
     2     1

>> A./B
ans =
     1     2
     4     2

>> A.\B
ans =
    1.0000    0.5000
    0.2500    0.5000
```

But note the matrix operations:

```
>> sym(A/B)   % A*inv(B)
ans =
   [  7/3, -2/3]
   [ -4/3, 14/3]

>> A\B         % inv(A)*B
ans =
     0.2    0.0
     0.2    0.5
```

element-wise operations

```
>> A=[1 4; 8 2], B=[1 2; 2 1]
A =
     1     4
     8     2
B =
     1     2
     2     1

>> A.*B
ans =
     1     8
    16     2
```

```
>> A.^B
ans =
     1    16
    64     2
```

```
>> B.^A
ans =
     1    16
   256     1
```

functions of matrices

```
>> X = [pi/2, pi/3; pi/4, pi/8]
X =
    1.5708    1.0472
    0.7854    0.3927

>> sin(X)
ans =
    1.0000    0.8660
    0.7071    0.3827

>> sin(sym(X))
ans =
[         1,               3^(1/2)/2]
[ 2^(1/2)/2, (2 - 2^(1/2))^(1/2)/2]
```

many functions operate
element-wise on matrices
e.g., trig, exp, log functions

others, operate column-wise
e.g., min, max, sort, diff,
mean, std, median,
sum, cumsum, prod, cumprod

functions of matrices

functions that operate column-wise can also operate row-wise by using a second argument

```
A = [8     5     8
     9     1     3
     2     4     5
     6     2     2];

>> sum(A)
ans =
    25    12    18

>> cumsum(A)
ans =
     8     5     8
    17     6    11
    19    10    16
    25    12    18
```

```
>> sum(A,2)
ans =
    21
    13
    11
    10

>> cumsum(A,2)
ans =
     8    13    21
     9    10    13
     2     6    11
     6     8    10
```

functions of matrices

```
A = [8    5    8
     9    1    3
     2    4    5
     6    2    2];

>> mean(A), mean(A,2)
ans =
    6.25  3.00  4.50
ans =
    7.0000
    4.3333
    3.6667
    3.3333
```

means computed across rows

```
>> [m,i]=min(A)
m =
     2    1    2
i =
     3    2    4

>> [m,i]=min(A,[],2);
>> [m,i]
ans =
     5  2
     1  2
     2  1
     2  2
```

`min`, `max` require a sightly different syntax for row-wise operation, similarly for **`diff`, `std`**

functions of matrices

```
A = [8     5     8
     9     1     3
     2     4     5
     6     2     2];

>> fliplr(A)
ans =
     8     5     8
     3     1     9
     5     4     2
     2     2     6

>> flipud(A)
ans =
     6     2     2
     2     4     5
     9     1     3
     8     5     8
```

```
>> rot90(A)
ans =
     8     3     5     2
     5     1     4     2
     8     9     2     6
```

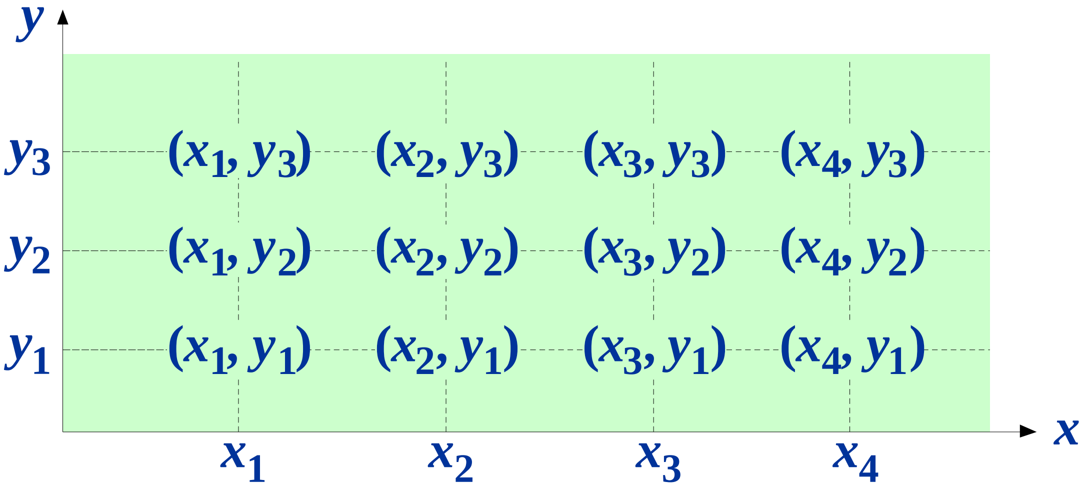
rotate by 90 degrees

reverse each row

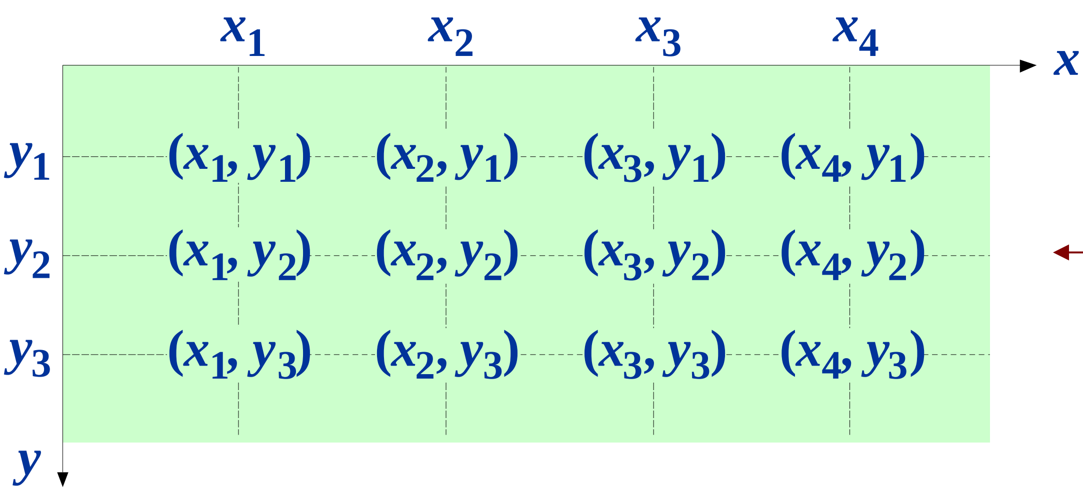
reverse each column

`flipud(rot90(A))` and **`rot90(fliplr(A))`** are the same as **`A'`**

meshgrid
ndgrid



a function
$z = f(x,y)$
defines a surface



meshgrid

x = [x1, x2, x3, x4]
y = [y1, y2, y3]

[X,Y] = meshgrid(x,y)

$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 & y_1 & y_1 & y_1 \\ y_2 & y_2 & y_2 & y_2 \\ y_3 & y_3 & y_3 & y_3 \end{bmatrix}$$

meshgrid
ndgrid

```
>> x = [1 2 3 4];   % N-dim
>> y = [5 6 7];     % M-dim

>> [X,Y] = meshgrid(x,y)
X =
   1   2   3   4
   1   2   3   4
   1   2   3   4
Y =
   5   5   5   5
   6   6   6   6
   7   7   7   7
```

rows are copies of **x**

columns are copies of **y**

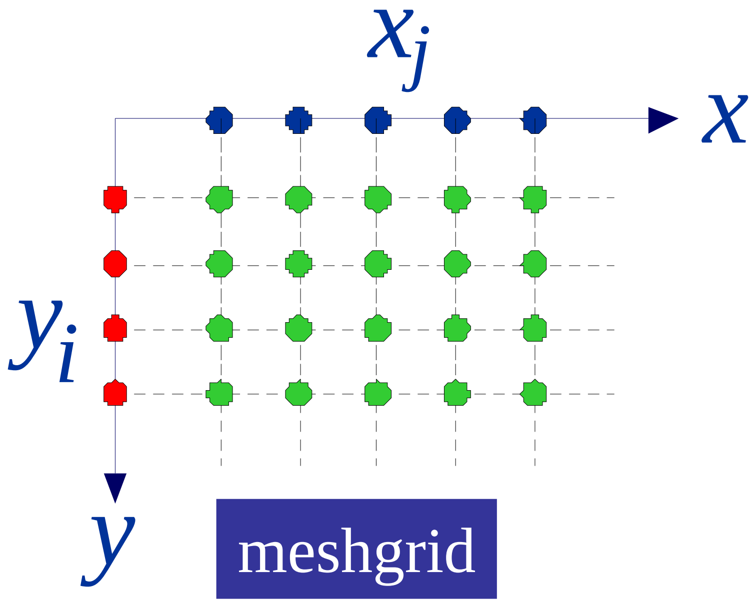
X,Y have size MxN

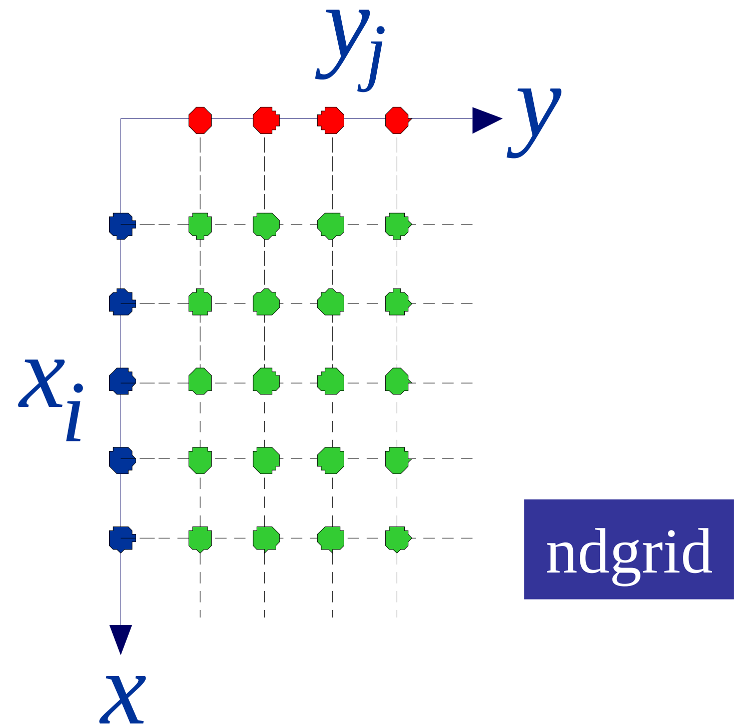
```
[Y,X] = meshgrid(y,x)
```

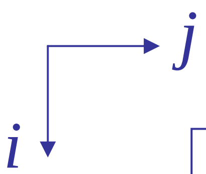
equivalent

```
>> [X,Y] = ndgrid(x,y)
X =
   1   1   1
   2   2   2
   3   3   3
   4   4   4
Y =
   5   6   7
   5   6   7
   5   6   7
   5   6   7
```

X,Y have size NxM







[X,Y] = meshgrid(x,y)

x = rows
y = columns

[X,Y] = ndgrid(x,y)

x = columns
y = rows

equivalent

[Y,X] = meshgrid(y,x)

[Y,X] = ndgrid(y,x)

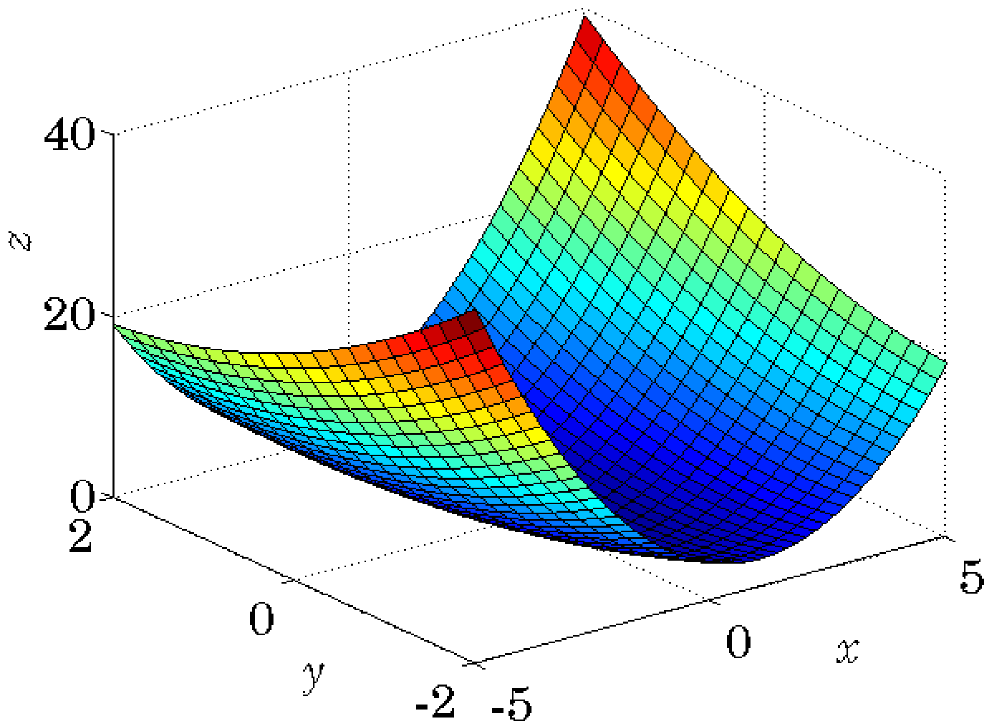
Example 1: Surface plot

```
x = linspace(-5,5,51);
y = linspace(-2,2,21);

[X,Y] = meshgrid(x,y);

Z = X.*Y + Y.^2 + X.^2;
figure; surf(X,Y,Z);
```

element-wise operations make sense because X,Y have the same dimensions.

meshgrid is typically used for surface plots, which are graphs of functions of two variables, z = f(x,y), e.g.,

z = f(x,y) = x*y + x^2 + y^2



Example 2:

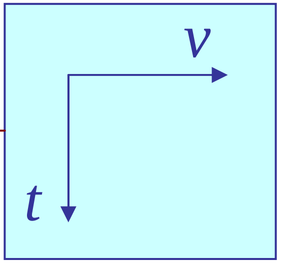
calculate $x = vt$

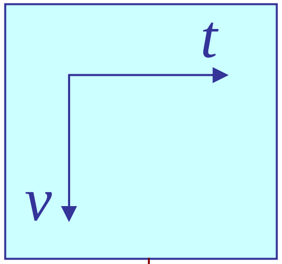
at a vector of times t

for different values of v

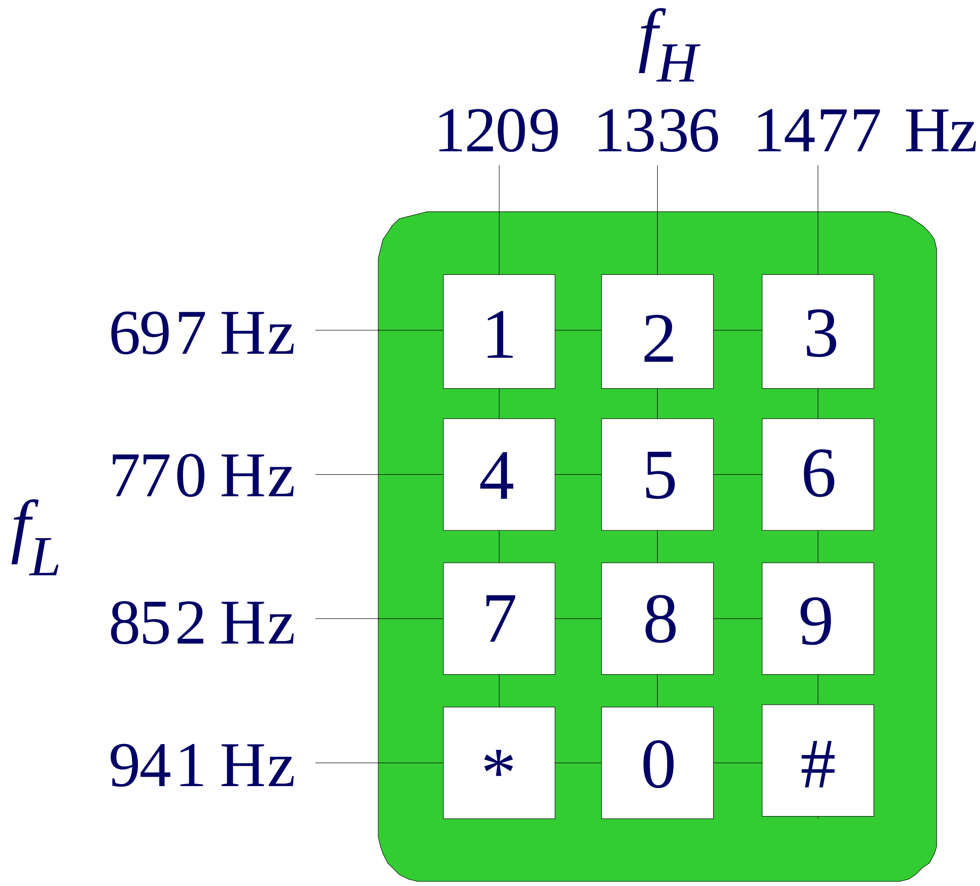
```
t = [0 1 2 3 4];
v = [1 -2 3];

[T,V] = meshgrid(t,v);
X1 = V.*T

[V,T] = meshgrid(v,t);
X2 = V.*T

[T,V] = ndgrid(t,v);
X3 = V.*T

[V,T] = ndgrid(v,t);
X4 = V.*T
```

```
X1 =
     0     1     2     3     4
     0    -2    -4    -6    -8
     0     3     6     9    12
X2 =
     0     0     0
     1    -2     3
     2    -4     6
     3    -6     9
     4    -8    12
X3 =
     0     0     0
     1    -2     3
     2    -4     6
     3    -6     9
     4    -8    12
X4 =
     0     1     2     3     4
     0    -2    -4    -6    -8
     0     3     6     9    12
```





Example 3: Dual-Tone-Multi-Frequency (DTMF) keypad (aka, touch-tone phone)



each key is mapped to a pair of low & high frequencies (fL,fH)

we may use ndgrid to replicate fL across columns, and fH along rows

$$y(t) = \sin(2\pi f_L t) + \sin(2\pi f_H t)$$

```
K = ['1'    '2'    '3'
     '4'    '5'    '6'
     '7'    '8'    '9'
     '*'    '0'    '#'];

% equivalently,
% K = ['123'; '456'; '789'; '*0#'];

fL = [697, 770, 852, 941];
fH = [1209, 1336, 1477];

[FL,FH] = ndgrid(fL,fH);
```

keypad matrix

complete code in keypad.m

$$F_L = \begin{bmatrix} 697 & 697 & 697 \\ 770 & 770 & 770 \\ 852 & 852 & 852 \\ 941 & 941 & 941 \end{bmatrix}, \quad F_H = \begin{bmatrix} 1209 & 1336 & 1477 \\ 1209 & 1336 & 1477 \\ 1209 & 1336 & 1477 \\ 1209 & 1336 & 1477 \end{bmatrix}$$

```
s = input('enter number to dial: ', 's');

fs = 8192;                  % soundcard sampling rate
T = 1/fs;                   % sampling time interval
Td = 1/4;                   % duration of each key, sec
Tb = 1/10;                  % blanking time, sec
Nb = round(Tb*fs);          % blanking time samples
yb = zeros(1,Nb);           % blanking signal samples

t = 0:T:Td;                 % time vector for each key

y = [];                     % initialize overall signal
```

enter a phone number to dial, e.g., **`s = 2125551212`**, set up sampling rate for sound card, set playing time of each key, and blanking time between keys

```
for k=1:length(s)

    [i,j] = find(K==s(k));

    x = sin(2*pi* FL(i,j) * t) +...
        sin(2*pi * FH(i,j) * t);

    y = [y, x, yb];

end

sound(y,fs);
```

find the location of key within the keypad matrix

append to overall signal

play overall signal

Note: the phone number **s** encoded into the signal **y** can be retrieved at the receiver by appropriate post-processing of **y**.

One method for doing this is via the *discrete-time Fourier transform* and is included in the program keypad.m. However, we will not cover it in this course (you'll learn about it in your DSP courses).

```
% y = [];
% for k=1:length(s)
%    q = find(K==s(k));
%    x = sin(2*pi* FL(q) * t) +...
%        sin(2*pi * FH(q) * t);
%    y = [y, x, yb];
% end
% sound(y,fs)
```

equivalent method using column-wise matrix indexing

included in keypad.m

```
% y = [];
% for k=1:length(s)
%    [i,j] = find(K==s(k));
%     x = sin(2*pi* fL(i) * t) +...
%         sin(2*pi * fH(j) * t);
%   y = [y, x, yb];
% end
% sound(y,fs)
```

equivalent method w/o using ndgrid

```
>> K = ['1'    '2'    '3'
        '4'    '5'    '6'
        '7'    '8'    '9'
        '*'    '0'    '#'];
```

understanding the comparison method and the use of **find**

```
>> K=='8'
ans =
     0     0     0
     0     0     0
     0     1     0
     0     0     0
```

compares every element of K with '8'

find the location of the correct element of K

```
>> [i,j] = find(K=='8')
i =
     3
j =
     2
```

i,j matrix indices of the location of '8'

```
>> q = find(K=='8')
q =
     7
```

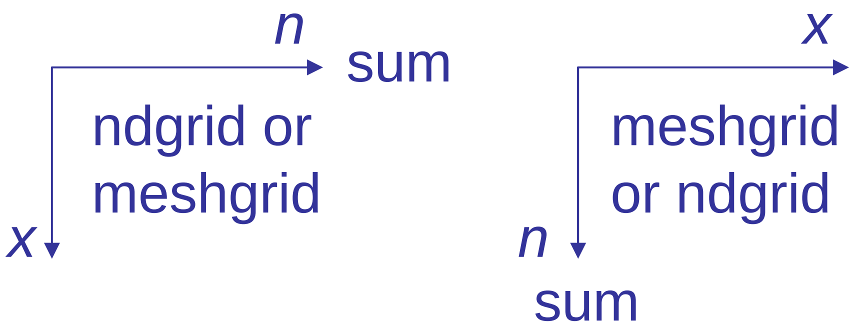
q is the column-wise index of '8' in K

Example 4: Vectorized Taylor series calculations

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \lim_{N \to \infty} \sum_{n=0}^{N} \frac{x^n}{n!}$$

$$S(x, N) = \sum_{n=0}^{N} \frac{x^n}{n!}$$

may be viewed as a matrix in the variables (x,n), and summed along the n dimension



x can be a column, row, or matrix, but it is used as x(:) inside ndgrid

```
[X,n] = ndgrid(x, 0:N);

S = sum(X.^n ./ factorial(n), 2);
```

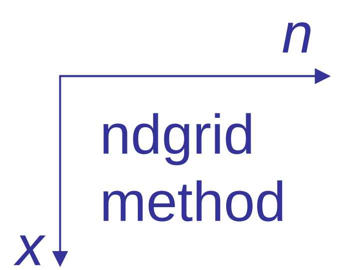
ndgrid method:
x = columns
n = rows
sum row-wise

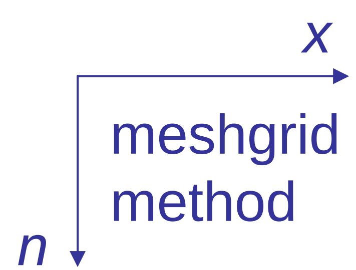
```
[X,n] = meshgrid(x, 0:N);

S = sum(X.^n ./ factorial(n));
```

meshgrid method:
n = columns
x = rows
sum column-wise

```
S = reshape(S,size(x));
```

reshape S into column, row, or matrix to match the given shape of x





alternative methods with partial or no vectorization

```
S = zeros(size(x));
n = 0:N;
```

n is vectorized, but not x

```
for i=1:length(x),
   S(i) = sum(x(i).^n ./ factorial(n));
end
```

x is vectorized, but not n

```
for n=0:N,
   S = S + x.^n / factorial(n);
end
```

no vectorization

```
for i=1:length(x),
   for n=0:N,
      S(i) = S(i) + x(i)^n / factorial(n);
   end
end
```

```
x = [1 3 0 -4 10]'; N = 30;          % x is column

[X,n] = meshgrid(x, 0:N);
S = sum(X.^n ./ factorial(n));
S = reshape(S,size(x));

fprintf('\n     x          exp(x)         S(x,N)\n');
fprintf('   --------------------------------\n');
fprintf('% 7.2f  %12.6f  %12.6f\n', [x,exp(x),S]');
```

```
     x          exp(x)         S(x,N)
   --------------------------------
   1.00      2.718282      2.718282
   3.00     20.085537     20.085537
   0.00      1.000000      1.000000
  -4.00      0.018316      0.018316
  10.00  22026.465795  22026.464036
```

needs larger N

later on we'll discuss how to pick N to achieve any desired degree of convergence

Example 5: Polynomial evaluation

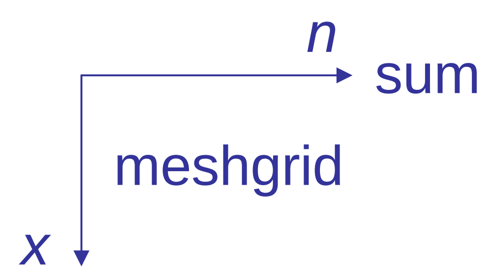
$$P(x) = c_0 x^M + c_1 x^{M-1} + \cdots + c_{M-1} x + c_M$$

$$= \sum_{n=0}^{M} c_n x^{M-n}$$

may be viewed as a matrix in the variables (x,n), and summed along the n dimension

$$\mathbf{c} = [c_0, c_1, \ldots, c_M]$$

```
M = length(c)-1;
[n,X] = meshgrid(0:M, x);
C = meshgrid(c,x);

P = sum(C .* X.^(M-n), 2);

P = reshape(P,size(x));
```



why did we use **`[n,X]`** instead of **`[X,n]`?**

$$P(x) = x^3 - 3x^2 + 4x + 2$$

```
x = [1, 2, 3]; c = [1, -3, 4, 2];

M = length(c)-1;
[n,X] = meshgrid(0:M, x);
C = meshgrid(c,x);

P = sum(C .* X.^(M-n), 2);

P = reshape(P,size(x))
P =
     4     6    14

polyval(c,x)
ans =
     4     6    14
```

polyval is the standard built-in function for polynomial evaluation

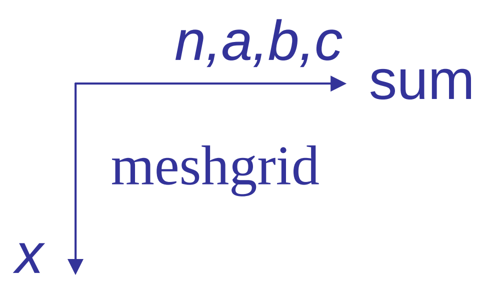
Example 6: Peaks

the polynomial method easily generalizes to other parametric curves

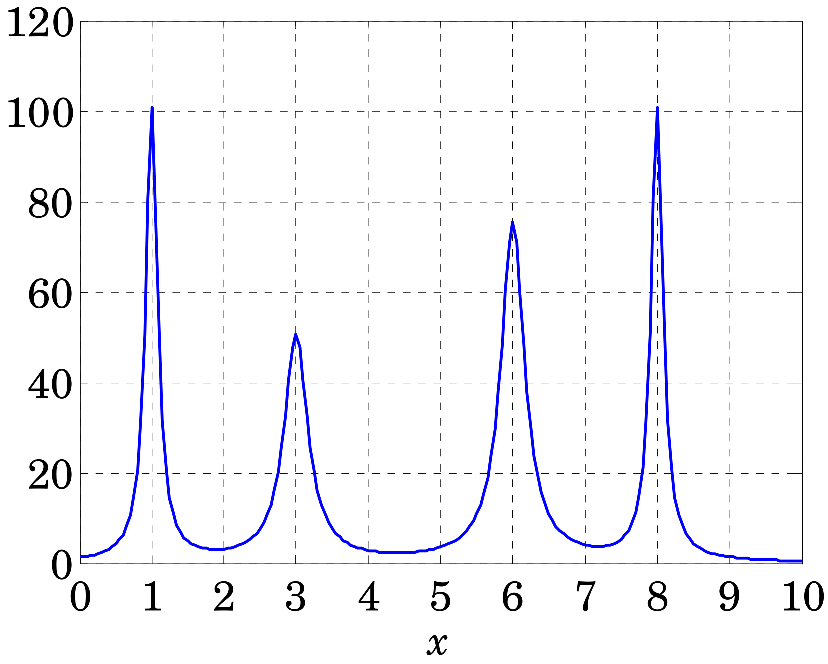
$$S(x) = \sum_{n=1}^{M} \frac{c_n}{(x-a_n)^2 + b_n^2} = \frac{c_1}{(x-a_1)^2 + b_1^2} + \cdots + \frac{c_M}{(x-a_M)^2 + b_M^2}$$

known as Lorentzian curves, used for modeling chemical spectral peaks

```
x = linspace(0,10,201);
a = [1 3 6 8];
b = [0.1, 0.2, 0.2, 0.1];
c = [1 2 3 1];

[A,X] = meshgrid(a,x);
B = meshgrid(b,x);
C = meshgrid(c,x);

S = sum(C./((X-A).^2 + B.^2), 2);
S = reshape(S,size(x));
```



```
figure; plot(x,S,'b-');

xaxis(0,10, 0:10);
yaxis(0,120, 0:20:120);
xlabel('\itx'); grid;
```




```
S = zeros(size(x));

for n=1:length(c),
   S = S + c(n)./((x-a(n)).^2 + b(n)^2);
end
```

conventional method
w/o using meshgrid