International University
School of Electrical Engineering

Introduction to Computers for Engineers

Dr. Hien Ta

Lecturely Topics

Lecture 1 - Basics – variables, arrays, matrices
Lecture 2 - Basics – matrices, operators, strings, cells
Lecture 3 - Functions & Plotting
Lecture 4 - User-defined Functions
Lecture 5 - Relational & logical operators, if, switch statements
Lecture 6 - For-loops, while-loops
Lecture 7 - Review on Midterm Exam
Lecture 8 - Solving Equations & Equation System (Matrix algebra)
Lecture 9 - Data Fitting & Integral Computation
Lecture 10 - Representing Signal and System
Lecture 11 - Random variables & Wireless System
Lecture 12 - Review on Final Exam

References: H. Moore, *MATLAB for Engineers*, 4/e, Prentice Hall, 2014
G. Recktenwald, *Numerical Methods with MATLAB*, Prentice Hall, 2000
A. Gilat, *MATLAB, An Introduction with Applications*, 4/e, Wiley, 2011

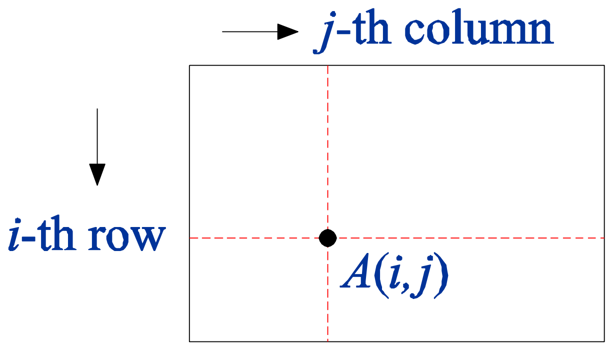
Arrays and Matrices

arrays and matrices are the most
important data objects in MATLAB

Last week we discussed one-dimensional arrays,
i.e., column or row vectors.

Next, we discuss matrices, which are two-dimensional arrays.

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



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)          % [N,M] = size(A), NxM matrix
ans =
     3     3
```

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
```

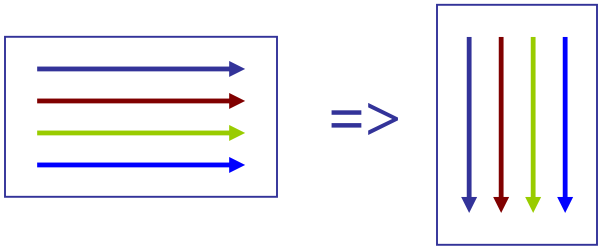
$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 0 & 8 & 5 \end{bmatrix}$$

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

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

transposing a matrix:
rows become columns and vice versa

```
>> 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
```



transposition operation

For more information on elementary matrices see:

```
>> help elmat

  Elementary matrices and matrix manipulation.

  Elementary matrices:

    zeros       - matrix of zeros
    ones        - matrix of ones
    eye         - identity matrix
    repmat      - replicate and tile an array
    linspace    - linearly spaced vector
    logspace    - logarithmically spaced vector

    etc.
```

7. Operators and Expressions

operation	element-wise	matrix-wise
addition	`+`	`+`
subtraction	`-`	`-`
multiplication	`.*`	`*`
division	`./`	`/`
left division	`.\`	`\`
exponentiation	`.^`	`^`
transpose w/o complex conjugation		`.'`
transpose with complex conjugation		`'`

these must follow the rules of linear algebra

```
>> help /
>> help precedence
```

```
>> a = [1  2 5];
>> b = [4 -5 1];

>> a+b
ans =
     5    -3      6

>> a.*b
ans =
     4   -10      5

>> a./b
ans =
    0.2500   -0.4000    5.0000

>> a.\b
ans =
    4.0000   -2.5000    0.2000

% note: (a./b).*(a.\b) = [1,1,1]
```

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

>> a.^2              % [2^2, 3^2, 4^2, 5^2]
ans =
     4     9    16    25

>> 2.^a              % [2^2, 2^3, 2^4, 2^5]
ans =
     4     8    16    32

>> a+10
ans =
    12    13    14    15
```

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

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

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

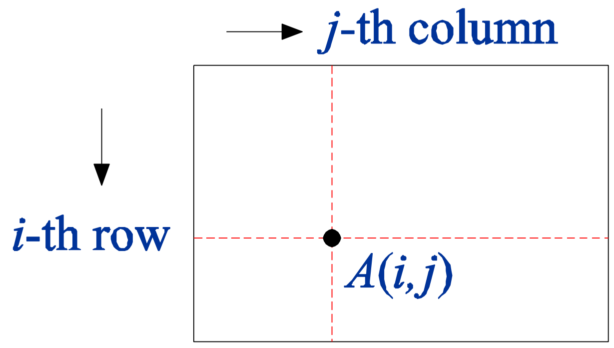
$$B = \begin{bmatrix} 10^1 & 10^2 \\ 10^3 & 10^4 \end{bmatrix}$$

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

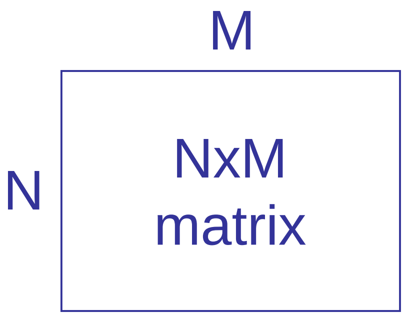
defining matrices

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



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)                % [N,M] = size(A), NxM matrix
ans =
     3     3
```

column dimension	row dimension
no. rows	no. columns

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
```

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

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

concatenating columns

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

```
>> 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(:)`

see also the built-in functions:

sub2ind, ind2sub

building a matrix column-wise

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

>> 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
```

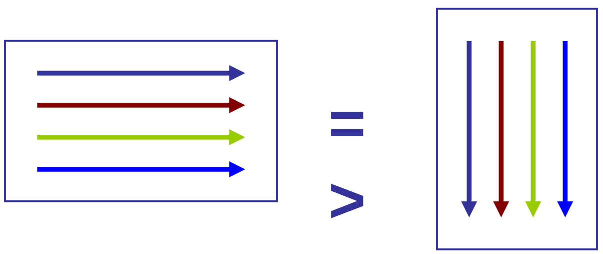
$$A = \begin{bmatrix} 2 & 4 & 1 & 3 & 5 \\ 8 & 6 & 7 & 4 & 9 \\ 3 & \boxed{2 & 5 & 2} & 1 \\ 5 & 6 & 1 & 8 & 4 \end{bmatrix}$$

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

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

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
```



transposition operation

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
```

inserting rows or columns

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

% insert new column between columns 2 & 3
A = [A(:,1:2), [10 20 30]', A(:,3)]
A =
     1     2    10     3
     2     0    20     4
     0     8    30     5

% insert new row between rows 1 & 2
A = [A(1,:); [60 70 80 90]; A(2:3,:)]
A =
     1     2    10     3
    60    70    80    90
     2     0    20     4
     0     8    30     5
```

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

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

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

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

general usage:

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

see also:

```
rand(N,M)
randn(N,M)
randi(I,N,M)
```

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
```

>> help diag

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

`diag(diag(A))` = what does it do?

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

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 – using repmat

```
>> 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

>> s = repmat('%7.4f   ', 1, 4)
s =
%7.4f   %7.4f   %7.4f   %7.4f
```

repmat works also with other data types, such as strings or cell arrays

replicated string

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
matrix operations

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

element-wise operation
matrix-wise operation

```
>> [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
```

```
>> 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
```

element-wise matrix operations

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
matrix 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 down each column

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

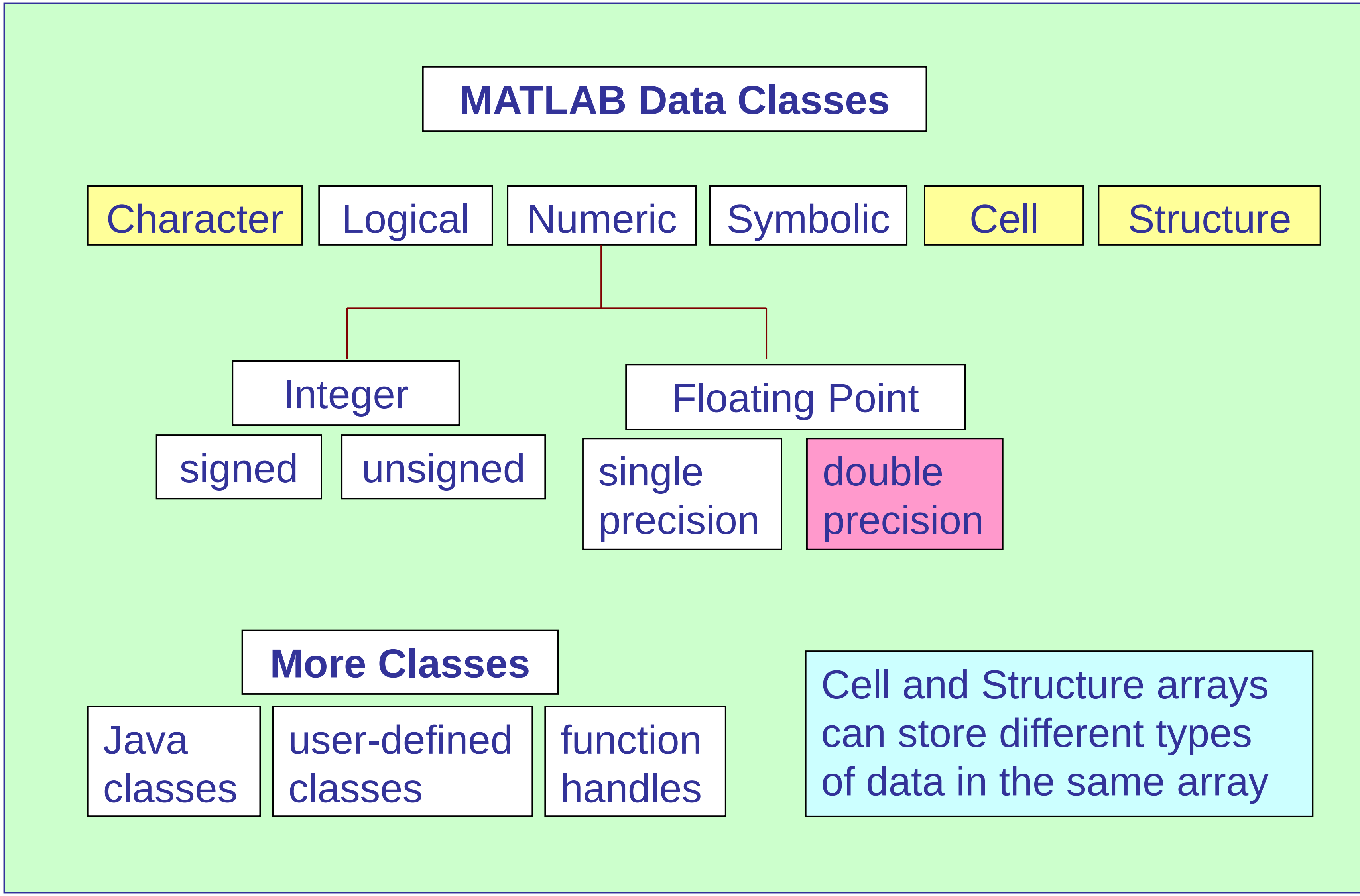
flipud(rot90(A)) and **rot90(fliplr(A))** are the same as **A'**

Strings, Cell Arrays, `fprintf`

- characters and strings
- concatenating strings
- using `num2str`
- comparing strings with `strcmp`

- cell arrays
- cell vs. content indexing

- `fprintf` – summary & examples


MATLAB Data Classes
Character
Logical
Numeric
Symbolic
Cell
Structure
Integer
Floating Point
signed
unsigned
single precision
double precision
More Classes
Java classes
user-defined classes
function handles
Cell and Structure arrays can store different types of data in the same array

Characters and Strings

Strings are arrays of characters.

Characters are represented internally by standardized numbers, referred to as ASCII (American Standard Code for Information Interchange) codes. see Wikipedia link: ASCII table

```
>> c = 'A'
c =
A

>> x = double(c)
x =
    65    % ASCII code for 'A'

>> char(x)
ans =
A

>> class(c)
ans =
char
```

`char()` creates a character string

```
>> doc char
>> doc class
```

```
>> s = 'ABC DEFG'
s =
ABC DEFG

>> x = double(s)
x =
    65 66 67 32 68 69 70 71
```

ASCII codes

```
>> char(x)
ans =
ABC DEFG
```

convert ASCII codes to characters

```
>> size(s)
ans =
     1     8
>> class(s)
ans =
char
```

s is a row vector of 8 characters

```
>> s(2), s(3:5)
ans =
B
ans =
C D
```

Concatenating Strings

```
s = ['Albert', 'Einstein']
s =
AlbertEinstein

>> s = ['Albert', ' Einstein']
s =
Albert Einstein

>> s = ['Albert ', 'Einstein']
s =
Albert Einstein

>> size(s)
ans =
     1    15
```

preserve leading and trailing spaces

```
>> doc strcat
>> doc strvcat
>> doc num2str
>> doc strcmp
>> doc findstr
```

Concatenating Vertically

```
s = ['Apple'; 'IBM'; 'Microsoft'];

??? Error using ==> vertcat
CAT arguments dimensions are not consistent.

s = ['Apple    '; 'IBM      '; 'Microsoft']

s =
Apple
IBM
Microsoft

>> size(s)
ans =
     3     9
```

padded with spaces to the length of the longest string

Concatenating Vertically

```
s = strvcat('Apple', 'IBM', 'Microsoft');
s = char('Apple', 'IBM', 'Microsoft');
```

```
s =
Apple
IBM
Microsoft
```

```
>> size(s)
ans =
     3     9
```

strvcat,char
both pad spaces as necessary

Recommendation:
use **char** to concatenate vertically,
and **[]** to concatenate horizontally

```
a = [143.87, -0.0000325, -7545]';
```

num2str

```
>> s = num2str(a)
s =
   143.87
-3.25e-005
    -7545
```

```
s = num2str(A)
s = num2str(A, precision)
s = num2str(A, format)
```

```
>> s = num2str(a,4)
s =
    143.9
-3.25e-005
    -7545
```

max number of digits

```
>> s = num2str(a, '%12.6f')
s =
  143.870000
   -0.000032
-7545.000000
```

format spec

Comparing Strings

Strings are arrays of characters, so the condition **`s1==s2`** requires both **`s1`** and **`s2`** to have the same length

```
>> s1 = 'short'; s2 = 'shore';
>> s1==s1
ans =
     1     1     1     1     1

>> s1==s2
ans =
     1     1     1     1     0

>> s1 = 'short'; s2 = 'long';
>> s1==s2
??? Error using ==> eq
Matrix dimensions must agree.
```

Comparing Strings

Use **strcmp** to compare strings of unequal length, and get a binary decision

```
>> s1 = 'short'; s2 = 'shore';
>> strcmp(s1,s1)
ans =
     1

>> strcmp(s1,s2)
ans =
     0

>> s1 = 'short'; s2 = 'long';
>> strcmp(s1,s2)
ans =
     0
```

```
>> doc strcmp
>> doc strcmpi
```

case-insensitive

Useful String Functions

`sprintf`	–	write formatted string
`sscanf`	–	read formatted string
`deblank`	–	remove trailing blanks
`strcmp`	–	compare strings
`strcmpi`	–	compare strings
`strmatch`	–	find possible matches
`upper`	–	convert to upper case
`lower`	–	convert to lower case
`blanks`	–	string of blanks
`strjust`	–	left/right/center justify string
`strtrim`	–	remove leading/trailing spaces
`strrep`	–	replace strings
`findstr`	–	find one string within another

Cell Arrays

Cell arrays are containers of all kinds of data: vectors, matrices, strings, structures, other cell arrays, functions.

A cell is created by putting different types of objects in curly brackets { }, e.g.,

```
c = {A, B, C, D};      % 1x4 cell
c = {A; B; C; D};      % 4x1 cell
c = {A, B; C, D};      % 2x2 cell
```

where **A,B,C,D** are arbitrary objects

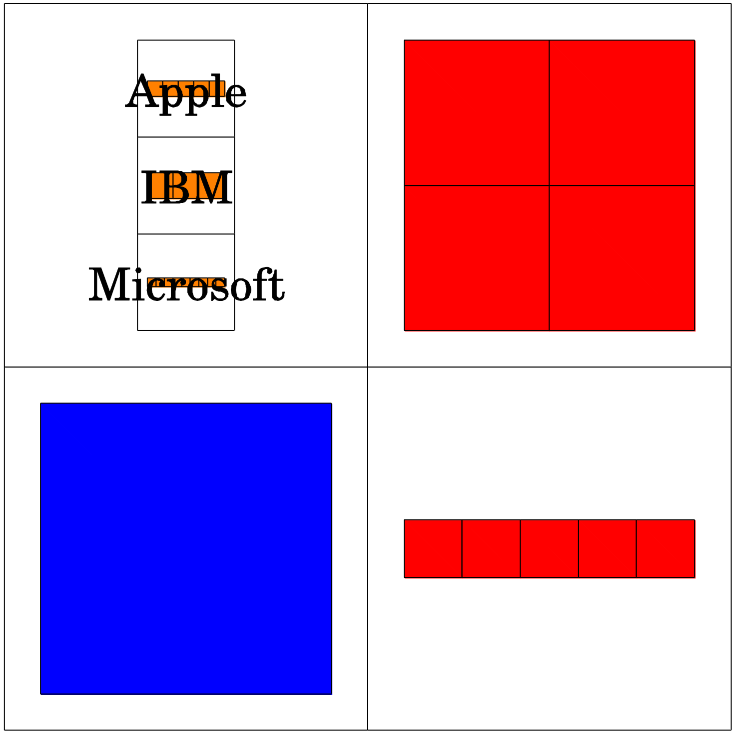
c{i,j} accesses the data in **i,j** cell

c(i,j) is the cell object in the **i,j** position

(see text Ch.11)

cell vs. content indexing

```
A = {'Apple'; 'IBM'; 'Microsoft'};   % cells
B = [1 2; 3 4];                      % matrix
C = @(x) x.^2 + 1;                   % function
D = [10 20 30 40 50];                % row

c = {A,B;C,D}    % define 2x2 cell array

c =
   {3x1 cell}   [2x2 double]
   @(x)x.^2+1   [1x5 double]

  >> cellplot(c);
```



visual representation of the cell array

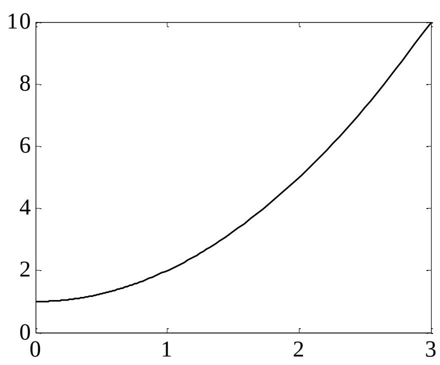
```
>> celldisp(c)

c{1,1}{1} =
Apple
c{1,1}{2} =
IBM
c{1,1}{3} =
Microsoft

c{2,1} =
    @(x)x.^2+1

c{1,2} =
     1     2
     3     4

c{2,2} =
    10    20    30    40    50
```

content indexing with { }

```
>> c{1,1}
ans =
    'Apple'
    'IBM'
    'Microsoft'

>> c{2,1}
ans =
    @(x)x.^2+1
```

content indexing with { }

```
>> c{1,1}{3}
ans =
Microsoft
>> c{1,1}{3}(6)
ans =
s

>> x = [1 2 3];
>> c{2,1}(x)
ans =
     2     5    10

>> fplot(c{2,1},[0,3]);
```




```
>> c{1,2}(2,:)
ans =
     3     4

>> c{1,2}(1,2)
ans =
     2

>> c{2,2}(3)
ans =
    30
```

cell indexing ( )
content indexing { }

```
>> c(2,2)
ans =
    [1x5 double]
```

cell

```
>> class(c(2,2))
ans =
cell
```

```
>> c{2,2}
ans =
    10      20     30      40      50
```

cell contents

```
>> class(c{2,2})
ans =
double
```

num2cell - converts numerical array into cell array
cell2mat - converts cell array into numerical array

```
>> D = cell2mat(c(1,2))

D =
     1      2
     3      4


>> class(D)

ans =
    double


>> E = num2cell(D)

E =
    [1]     [2]
    [3]     [4]
```

```

>> size(E)

ans =
     2   2


>> size(c(1,2))

ans =
     1   1   % why?
```

`fprintf` – summary & examples

```
fprintf('format_specs', variables);
```

print format specifications

list of variables, arrays, or matrices to be printed

```
>> doc fprintf
>> doc sprintf
```

see Sect.7.2 of text

Example 1

```
>> fprintf('%10.6f\n',  100*pi)
>> fprintf('% 10.6f\n', 100*pi)
>> fprintf('%-10.6f\n', 100*pi)
>> fprintf('%+10.6f\n', 100*pi)
>> fprintf('%10.0f\n',  100*pi)
>> fprintf('%#10.0f\n', 100*pi)
>> fprintf('%010.0f\n', 100*pi)
```

```
314.159265
 314.159265
314.159265
+314.159265
       314
      314.
0000000314
```

10 spaces

different ways of printing 100*pi

```
%10.6f        % width 10, 6 decimal places
% 10.6f       % leave space before field
%-10.6f       % left-justify field
%+10.6f       % print + or - signs
%10.0f        % no decimals
%#10.0f       % print decimal point
%010.0f       % pad with zeros
```

flag

field width
& precision

conversion character:

`d, i`	integer format
`f`	fixed-point format
`e, E, g`	exponential format
`c, s`	character or string
`x`	hexadecimal format

Example 2

```
a = [1; -2; 3; 4;];
b = [10; 20; -30; 40];
c = [100; 200; 300; -400];
```

```
>> [a, b, c]

ans =
     1    10   100
    -2    20   200
     3   -30   300
     4    40  -400
```

need at least %6.3f to align first column

```
fprintf('%9.3f %9.3f %9.3f\n', [a, b, c]');
```

```
    1.000     10.000    100.000
   -2.000     20.000    200.000
    3.000    -30.000    300.000
    4.000     40.000   -400.000
```

vectorized version

loop version

```
for i=1:4,
  fprintf('%9.3f %9.3f %9.3f\n', a(i),b(i),c(i));
end
```

Example 3 - text & numbers

```
a = [1; -2; 3; 4;];
b = [10; 20; -30; 40];
s = {'a', 'bb', 'ccc', 'dddd'};
```

cell array of strings

```
for i=1:4,
  fprintf('%9.3f %9.3f  %4s\n', a(i),b(i),s{i});
end
```

note curly brackets

```
    1.000    10.000     a
   -2.000    20.000    bb
    3.000   -30.000   ccc
    4.000    40.000  dddd
```

max 4 characters

```
N = [num2cell([a';b']); s];
fprintf('%9.3f %9.3f  %-4s\n', N{:});
```

using a single **fprintf** command

left-justified

```
    1.000    10.000  a
   -2.000    20.000  bb
    3.000   -30.000  ccc
    4.000    40.000  dddd
```

```
>> N{:}           % fprintf('%9.3f %9.3f  %-4s\n', N{:});
ans =
     1
     10
     a

     -2
     20
     bb

     3
     -30
     ccc

     4
     40
     dddd
```

use the three format specs to print each group of three entries in new row

```
 1.000     10.000   a
-2.000     20.000   bb
 3.000    -30.000   ccc
 4.000     40.000   dddd
```

```
>> N
N =
    [ 1]   [-2]   [  3]   [   4]
    [10]   [20]   [-30]   [  40]
    'a'    'bb'   'ccc'     'dddd'
```