

International University
School of Electrical Engineering

Introduction to Computers for Engineers

Dr. Hien Ta

This course is an introduction to MATLAB, a powerful programming language and development environment for engineers and scientists.

Syllabus and other course materials can be found in:

<https://sites.google.com/site/taquanghien18787>

exam dates,
lecture notes,
homeworks, etc.

MATLAB = **Matrix** **L**aboratory (created by **Cleve Moler**)

MATLAB ® is a registered trademark of The Mathworks Inc., <http://www.mathworks.com>

MATLAB & Simulink Student Version

Main Features of MATLAB

- Easy and efficient programming in a high-level language, with an interactive interface for rapid development.
- Vectorized computations for efficient programming, and automatic memory allocation.
- Built-in support for state-of-the-art numerical computing methods.
- Has variety of modern data structures and data types, including complex numbers.
- High-quality graphics and visualization.

- Symbolic math toolbox for algebraic and calculus operations, and solutions of differential equations.
- Simulation capability with SIMULINK.
- Portable program files across platforms.
- Large number of add-on toolboxes for applications and simulations.
- Huge database of user-contributed files & toolboxes, including a large number of available tutorials & demos.
- Allows extensions based on other languages, such as C/C++, supports Java and object-oriented programming.

MATLAB Toolbox Application Areas

- Parallel Computing (2)
- Math, Statistics, and Optimization (8)
- Control System Design and Analysis (6)
- Signal Processing and Communications (7)
- Image Processing and Computer Vision (4)
- Test and Measurement, Data Acquisition (5)
- Computational Finance, Datafeeds (7)
- Computational Biology (2)
- Code Generation and Application Deployment (11)
- Database Connectivity (2)

(54 toolboxes + 35 simulink products)

SIMULINK Applications

- Fixed-Point and Event-Based Modeling
- Physical Modeling (mechanics, driveline, hydraulics, RF, electronics, power systems, biology)
- Control Systems (design, optimization, aerospace)
- Signal & Image Processing and Computer Vision
- Communication Systems (digital, analog, wireless)
- Code Generation (for embedded systems, DSP chips and FPGAs)
- more

Web Resources


- [Getting Started with MATLAB \(HTML\)](#)
- [Getting Started with MATLAB \(PDF\)](#)
- [MATLAB Examples](#)
- [MATLAB Online Tutorials and Videos](#)
- [MATLAB Interactive Tutorials](#)
- [MATLAB Toolbox Reference Manuals](#)
- [MATLAB Interactive CD](#)
- [Newsletters](#)

- [MATLAB User Community](#)
- [Other MATLAB Online Resources](#)
- [comp.soft-sys.matlab newsgroup](#)

- [Octave – a free look-alike version of MATLAB](#)
- [FreeMat – another free look-alike version](#)

- [NIST – Digital Library of Mathematical Functions](#)
- [NIST – Physical Constants](#)

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

1. MATLAB Desktop

choose desktop layout

set search path

doubleclick to edit M-file

current folder

view details about selected file

command window, enter commands at prompt

Help

navigate to desired folder

move, minimize, resize, close

to array editor

workspace window

command history

The screenshot shows the MATLAB 7.12.0 (R2011a) Desktop environment. The interface is divided into several panes: a File Explorer on the left showing the 'Current Folder' (E:\440-127\11\examples\octaves) with files like 'octaves.m' and 'xaxis.m'; a central Command Window with a MATLAB prompt and a 'Help' button; a Workspace window on the right showing variables 'a', 'ans', 'b', 'x', and 'y'; and a Command History window at the bottom right. The Command Window contains the following code and output:

```
>> a = 10
a =
    10

>> a = 10;
>> b = [10 20 30]
b =
    10    20    30

>> x = [0, pi/4, pi/3, pi/2, pi]; y = sin(x)
y =
     0    0.7071    0.8660    1.0000    0.0000

>> sin(sym(x))
ans =
[ 0, 2^(1/2)/2, 3^(1/2)/2, 1, 0]
```

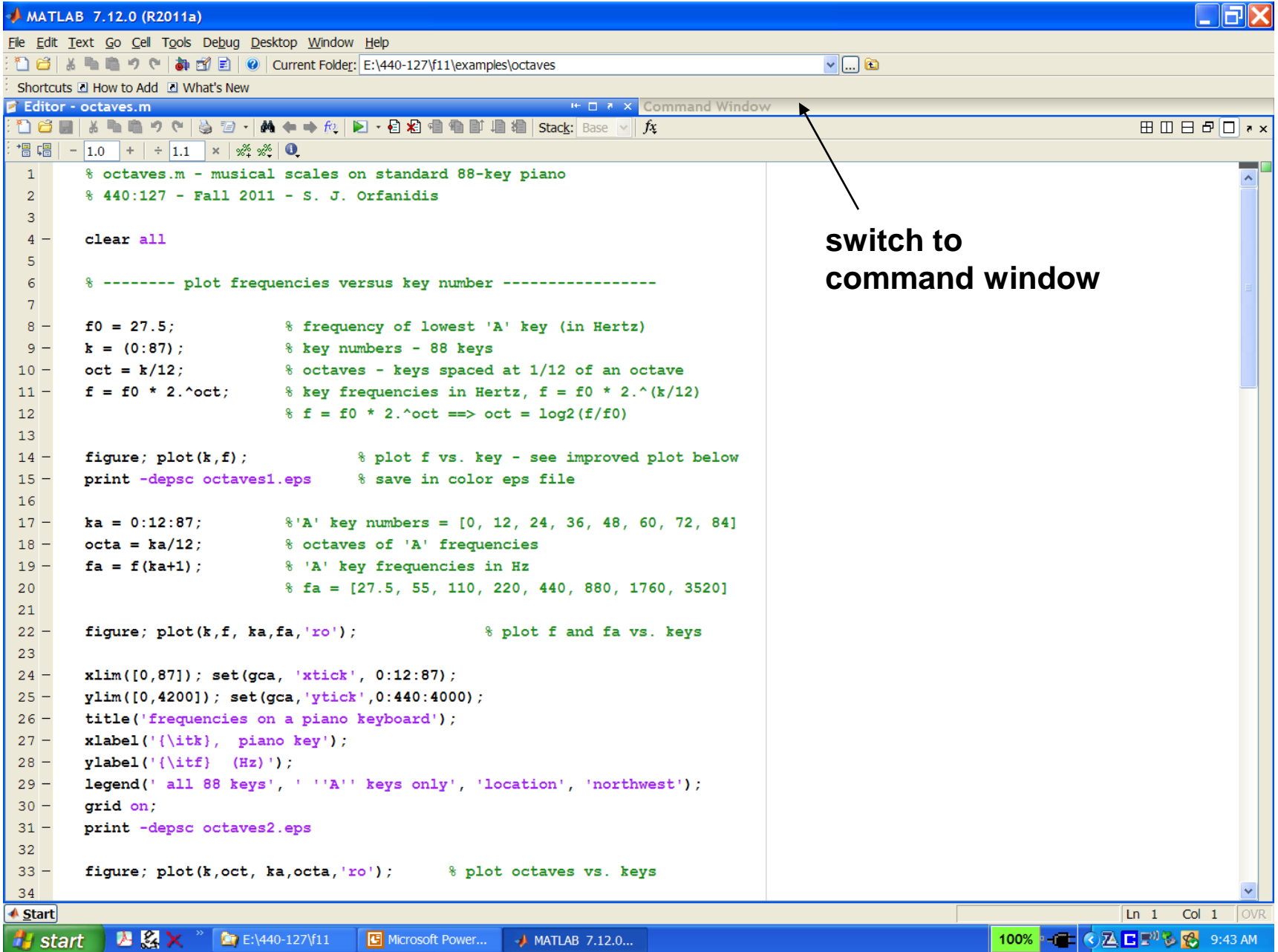
The Workspace window shows the following variables:

Name	Value
a	10
ans	<1x5 sym>
b	[10,20,30]
x	[0,0.7854,...
y	[0,0.7071,...

The Command History window shows the following commands:

```
%-- 6/18/2011 4:21
clc
a = 10
a = 10;
b = [10 20 30]
x = [0, pi/4, pi
sin(sym(x))
whos
```

2. MATLAB Editor



The image shows the MATLAB 7.12.0 (R2011a) Editor window. The main window displays a script named `octaves.m` with the following code:

```
1 % octaves.m - musical scales on standard 88-key piano
2 % 440:127 - Fall 2011 - S. J. Orfanidis
3
4 clear all
5
6 % ----- plot frequencies versus key number -----
7
8 f0 = 27.5;           % frequency of lowest 'A' key (in Hertz)
9 k = (0:87);         % key numbers - 88 keys
10 oct = k/12;          % octaves - keys spaced at 1/12 of an octave
11 f = f0 * 2.^oct;     % key frequencies in Hertz, f = f0 * 2.^(k/12)
12                     % f = f0 * 2.^oct ==> oct = log2(f/f0)
13
14 figure; plot(k,f);   % plot f vs. key - see improved plot below
15 print -depsc octaves1.eps % save in color eps file
16
17 ka = 0:12:87;        % 'A' key numbers = [0, 12, 24, 36, 48, 60, 72, 84]
18 octa = ka/12;        % octaves of 'A' frequencies
19 fa = f(ka+1);        % 'A' key frequencies in Hz
20                     % fa = [27.5, 55, 110, 220, 440, 880, 1760, 3520]
21
22 figure; plot(k,f, ka,fa,'ro'); % plot f and fa vs. keys
23
24 xlim([0,87]); set(gca, 'xtick', 0:12:87);
25 ylim([0,4200]); set(gca,'ytick',0:440:4000);
26 title('frequencies on a piano keyboard');
27 xlabel('{\itk}, piano key');
28 ylabel('{\itf} (Hz)');
29 legend(' all 88 keys', ' ''A'' keys only', 'location', 'northwest');
30 grid on;
31 print -depsc octaves2.eps
32
33 figure; plot(k,oct, ka,octa,'ro'); % plot octaves vs. keys
34
```

The Command Window is visible on the right side of the editor. An arrow points to the Command Window tab, with the text "switch to command window" next to it.

The Windows taskbar at the bottom shows the Start button, several open applications (including MATLAB 7.12.0), and the system clock showing 9:43 AM.


3. Getting Help

Several ways of getting help:

1) help **menu** item on MATLAB desktop opens up searchable help browser window

2) from the following commands:

comments begin with %




```
>> helpdesk           % open help browser
>> help topic        % e.g., help log10
>> doc topic         % e.g., doc plot
>> help              % get list of all help topics
>> help dir          % get help on entire directory
>> help syntax       % get help on MATLAB syntax
>> help /            % operators & special characters
>> docsearch text    % search HTML browser for 'text'
>> lookfor topic     % e.g., lookfor acos
```

4. Variables, Constants, Keywords

Variables require no special declarations of type or storage. Examples:

```
>> x = 3; % simple scalar
>> y = [4, 5, 6]; % row vector of length 3
>> z = [4; 5; 6]; % column vector of length 3
>> A = [1,2,3; 4,5,6]; % 2x3 matrix
>> s = 'abcd efg'; % string
>> C = {'abc', 'defg', '123-456'}; % 1x3 cell array
```

math notation


$$y = [4, 5, 6], \quad z = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

the functions **class** and **size** tell you the type and dimensions of the defined object, e.g.,

```
>> class(C)
>> size(C)
```

```
>> x = 3
```

```
x =
```

```
3
```

```
>> y = [4, 5, 6]
```

```
y =
```

```
4
```

```
5
```

```
6
```

```
>> z = [4; 5; 6] % note, z = y'
```

```
z =
```

```
4
```

```
5
```

```
6
```

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

```
A =
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

What are your variables? How to clear them?
Use workspace window, or the commands:

who, whos, clear, clc, close

```
>> who
```

```
Your variables are:
```

```
A   y   z
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
A	2x3	48	double	
y	1x3	24	double	
z	3x1	24	double	

```
>> clear all           % clear all variables from memory
```

```
>> clc                 % clear command window
```

```
>> close all           % close all open figures
```

Operating system commands:

```
>> path                % display search path
>> pathtool            % modify search path
>> addpath dir          % add directory to path

>> cd dir              % change directory
>> pwd                 % print working directory

>> dir                 % list all files in current dir
>> what                % list MATLAB files only
>> which file          % display location of file

>> edit file           % invoke MATLAB editor

>> quit                % quit MATLAB
>> exit                % quit MATLAB
```

Special built-in math constants that should not (though they can) be re-defined as variables:

eps	% machine epsilon - floating-point accuracy
i,j	% imaginary unit, i.e., sqrt(-1)
Inf,inf	% infinity
intmax	% largest value of specified integer type
intmin	% smallest value of specified integer type
NaN,nan	% not-a-number, e.g., 0/0, inf/inf
pi	% pi
realmax	% largest positive floating-point number
realmin	% smallest positive floating-point number

Note: **i,j** are commonly used for array and matrix indices. If you're dealing with complex-valued data, avoid redefining both **i,j**.

Values of special constants:

```
>> eps                                % equal to 2-52
ans =
    2.2204e-016                        % MATLAB's floating-point accuracy
                                        % i.e., 2.2204 * 10-16

>> intmax                             % 2(31)-1 for 32-bit integers
ans =
    2147483647

>> intmin                             % equal to -2(31)
ans =
   -2147483648

>> realmax                            % equal to (2-eps)*2(1023)
ans =
    1.7977e+308                        % i.e., 1.7977 * 10(308)

>> realmin                            % 2(-1022) = 2.2251 * 10(-308)
ans =
    2.2251e-308
```

Special keywords that cannot be used
as variable names:

```
>> iskeyword
```

```
ans =
```

```
'break'          'function'  
'case'           'global'  
'catch'          'if'  
'classdef'       'otherwise'  
'continue'       'parfor'  
'else'           'persistent'  
'elseif'         'return'  
'end'            'switch'  
'for'            'try'  
                 'while'
```

```
'true' , 'false'
```

5. Numbers and Formats

MATLAB by default uses **double-precision** (64-bit) floating-point numbers following the IEEE floating-point standard. You may find more information on this standard in:

Representation of Floating-Point Numbers

C. Moler, "Floating Points," MATLAB News and Notes, Fall, 1996 (PDF file)

$$x = (-1)^s * (1+f) * 2^{(e-1023)}$$

1 bit sign 52 bits mantissa 11 bits exponent

$1 \leq e \leq 2046, e=0, e=2047$

$$0 \leq f < 1$$
$$f_{\min} = \text{eps} = 2^{-52}$$

↑
machine epsilon

MATLAB can also use **single-precision** (32-bit) floating point numbers if so desired.

There are also several **integer** data types that are useful in certain applications, such as image processing or programming DSP chips. The integer data types have 8, 16, 32, or 64 bits and are signed or unsigned:

```
int8,    int16,    int32,    int64  
uint8,   uint16,   uint32,   uint64
```

For more information do:

```
>> help datatypes  
>> help class           % determine datatype
```

Complex Numbers

By default, MATLAB treats all numbers and expressions as complex (even if they are real).

No special declarations are needed to handle complex-number operations. Examples:

```
>> z = 3+4i;           % or, 3+4j, 3+4*i, 3+4*j
>> x = real(z);        % real part of z
>> y = imag(z);        % imaginary part of z
>> R = abs(z);         % absolute value of z
>> theta = angle(z);   % phase angle of z in radians
>> w = conj(z);        % complex conjugate, w=3-4i
>> isreal(z);          % test if z is real or complex
```

$$z = x + jy = Re^{j\theta}, \quad R = |z| = \sqrt{x^2 + y^2}, \quad \theta = \arctan \frac{y}{x}$$

cartesian & polar forms

math notation: $\theta = \text{Arg}(z)$

```
>> z = 3+4j
z =
    3.0000 + 4.0000i
```

```
>> x = real(z)
x =
    3
```

```
>> y = imag(z)
y =
    4
```

```
>> R = abs(z)
R =
    5
```

```
>> theta = angle(z) % in radians
theta =
    0.9273
```

```
>> abs(z - R*exp(j*theta)) + abs(z-x-j*y) % test
ans =
    6.2804e-016
```

equivalent definitions:

```
z = 3+4*j
z = 3+4i
z = 3+4*i
z = complex(3,4)
```

Display Formats

>> format	% default - 4 decimal places
>> format short	% same as the default
>> format long	% 15 decimal places
>> format short e	% 4 decimal - exponential format
>> format short g	% 4 decimals - exponential or fixed
>> format long e	% 15 decimals - exponential
>> format long g	% exponential or fixed
>> format shorteng	% 4 decimals, engineering
>> format longeng	% 15 decimals, engineering
>> format hex	% hexadecimal
>> format rat	% rational approximation
>> format compact	% conserve vertical spacing
>> format loose	% default vertical spacing
>> vpa(x,digits)	% variable-precision-arithmetic

These affect only the display format – internally all computations are done with full (double) precision

Example - displayed value of 10π in different formats:

31.4159	% format, or format short
31.415926535897931	% format long
3.1416e+001	% format short e
31.416	% format short g
3.141592653589793e+001	% format long e
31.4159265358979	% format long g
31.4159e+000	% format shorteng
31.4159265358979e+000	% format longeng
>> vpa(10*pi)	% symbolic toolbox
ans =	
31.415926535897932384626433832795	
>> vpa(10*pi,20)	% specify number of digits
ans =	
31.415926535897932385	

```
>> help format
>> help vpa
>> help digits
```


input/output functions: **disp**, **input**

```
>> x = 10; disp('the value of x is:'); disp(x);  
the value of x is:  
10
```

```
>> x = input('enter x: ')           % numerical input  
enter x: 100                        % 100 entered by user  
x =  
100
```

prompt string in single quotes

```
>> y = input('enter string: ', 's'); % string input  
enter string: abcd efg  
>> y = input('enter string: ')  
enter string: 'abcd efg'  
y =  
abcd efg
```

string entered with no quotes
string entered in quotes

```
>> help fprintf  
>> help sprintf
```

```
>> help disp  
>> help input  
>> help menu
```

6. Arrays and Matrices

arrays and matrices are the most important data objects in MATLAB

We discuss briefly:

- a) row and column vectors
- b) transposition operator, '
- c) colon operator, :
- d) equally-spaced elements, linspace
- e) accessing array elements
- f) dynamic allocation & de-allocation
- g) pre-allocation

The key to efficient MATLAB programming
can be summarized in three words:

vectorize, vectorize, vectorize

and avoid all loops

Compare the two alternative computations:

```
x = [2,-3,4,1,5,8];  
y = zeros(size(x));  
for n = 1:length(x)  
    y(n) = x(n)^2;  
end
```

```
x = [2,-3,4,1,5,8];  
y = x.^2;
```

element-wise exponentiation **.^**
ordinary exponentiation **^**

answer: y = [4,9,16,1,25,64]

```
>> x = [0 1 2 3 4 5]           % row vector
```

```
x =  
    0     1     2     3     4     5
```

```
>> x = 0:5                     % row vector
```

```
x =  
    0     1     2     3     4     5
```

```
>> x = [0 1 2 3 4 5]'         % column vector, (0:5)'
```

```
x =  
    0  
    1  
    2  
    3  
    4  
    5
```

the prime operator, `'`, or transpose, turns row vectors into column vectors, and vice versa

caveat: `'` is actually conjugate transpose, use dot-prime, `.'`, for transpose w/o conjugation

```
>> z = [i; 1+2i; 1-i]           % column vector
```

```
z =
```

```
    0 + 1.0000i  
    1.0000 + 2.0000i  
    1.0000 - 1.0000i
```

```
>> z.'                           % transpose without conjugation
```

```
ans =
```

```
    0 + 1.0000i    1.0000 + 2.0000i    1.0000 - 1.0000i
```

```
>> z'                             % transpose with conjugation
```

```
ans =
```

```
    0 - 1.0000i    1.0000 - 2.0000i    1.0000 + 1.0000i
```

```
>> (z.')'                         % same as (z')' , or, conj(z)
```

```
ans =
```

```
    0 - 1.0000i  
    1.0000 - 2.0000i  
    1.0000 + 1.0000i
```

about linspace:

```
x = linspace(a,b,N+1);
```

is equivalent to:

```
x = a : (b-a)/N : b;
```

i.e., $N+1$ equally-spaced points in the interval $[a,b]$
or, dividing $[a,b]$ into N equal sub-intervals

$$x(n) = a + \left(\frac{b-a}{N} \right) (n-1), \quad n = 1, 2, \dots, N+1$$

step
increment

```
>> x = 0 : 0.2 : 1
```

% in general, $x = a:s:b$

```
>> x = linspace(0,1,6)
```

% see also logspace

```
x =
```

```
0    0.2000    0.4000    0.6000    0.8000    1.0000
```

```
└──┘
```

6 points, 5 subintervals

step increment

```
>> x = 0 : 0.3 : 1
x =
    0    0.3    0.6    0.9
```

```
>> x = 0 : 0.4 : 1
x =
    0    0.4    0.8
```

```
>> x = 0 : 0.7 : 1
x =
    0    0.7
```

```
% before rounding, (b-a)/s was in the three cases:
% 1/0.3 = 3.3333, 1/0.4 = 2.5, 1/0.7 = 1.4286
```

x = a : s : b;

the number of subintervals within [a,b] is obtained by rounding $(b-a)/s$, down to the nearest integer,

N = floor((b-a)/s);

length(x) is equal to **N+1**

x(n) = a + s*(n-1),
n = 1,2,...,N+1

Note: MATLAB array indices always start with 1 and may not be 0 or negative

exception:
logical indexing,
discussed later

```
>> x = [ 2,    5,   -6,   10,    3,    4 ];  
        ↑    ↑    ↑    ↑    ↑    ↑  
      x(1) , x(2) , x(3) , x(4) , x(5) , x(6)
```

Other languages, such as C/C++ and Fortran, allow indices to start at 0. For example, the same array would be declared/defined in C as follows:

```
double x[6] = { 2,    5,   -6,   10,    3,    4 };  
              ↑    ↑    ↑    ↑    ↑    ↑  
            x[0] , x[1] , x[2] , x[3] , x[4] , x[5]
```

rule of thumb: $M = C + 1$

accessing array entries:

```
>> x = [2, 5, -6, 10, 3, 4]
```

```
x =
```

```
     2     5    -6    10     3     4
```

```
>> length(x)      % length of x, see also size(x)
```

```
ans =
```

```
     6
```

```
>> x(1)           % first entry
```

```
ans =
```

```
     2
```

```
>> x(3)           % third entry
```

```
ans =
```

```
    -6
```

```
>> x(end)         % last entry - need not know length
```

```
ans =
```

```
     4
```

accessing array entries:

```
>> x(end-3:end)           % x = [2, 5, -6, 10, 3, 4]
ans =
    -6    10     3     4      % last four

>> x(3:5)                 % list third-to-fifth entries
ans =
    -6    10     3

>> x(1:3:end)             % every third entry
ans =
     2    10

>> x(1:2:end)             % every second entry
ans =
     2    -6     3
```

accessing array entries:

```
>> x = [2, 5, -6, 10, 3, 4];
```

```
>> x(end:-1:1)      % list backwards, same as flip1r(x)
ans =
     4     3    10    -6     5     2
```

```
>> x([3,1,5])        % list [x(3),x(1),x(5)]
ans =
    -6     2     3
```

```
>> x(end+3) = 8
x =
     2     5    -6    10     3     4     0     0     8
```



automatic memory re-allocation

automatic memory allocation and de-allocation:

```
>> clear x
```

```
>> x(3) = -6
```

```
x =  
    0    0   -6
```

```
>> x(6) = 4
```

```
x =  
    0    0   -6    0    0    4
```

```
>> x(end) = [] % delete last entry
```

```
x =  
    0    0   -6    0    0
```

```
>> x = [2, 5, -6, 10, 3, 4];
```

```
>> x(3) = [] % delete third entry
```

```
x =  
    2    5   10    3    4
```

pre-allocation

```
>> clear x
>> x = zeros(1,6)           % 1x6 array of zeros
x =
    0    0    0    0    0    0

>> x = zeros(6,1)          % 6x1 array of zeros
x =
    0
    0
    0
    0
    0
    0
```

Pre-allocation is useful for very large arrays, e.g., **length** > 10^4 , for example, in dealing with audio or image files, or finite-element methods.

See, for example, the program **echoes.m**, which reads an audio file and adds reverberation effects to it, as described in **echoes.pdf**, and discussed also in Lecture-2 lectures.

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