Basics of Matlab

Adriana Hera

ahera @wpi.edu

- 2013-

Reference: Matlab documentation

MATLAB Course

1. Getting started; Matlab Help;
2. Variables;
3. Operators;
4. Matlab functions;
5. Matrices;
6. Scripts;
7. Basic plotting.
8. User Defined Functions
9. Importing Data
10. Simulink
11. Flow Control Commands

Where to find Matlab?

1. Matlab is installed on all computers in the public labs (i.e. HL230, library, etc)

2. To install Matlab on your personal computer go to:
http://www.wpi.edu/academics/CCC/Help/Software/Installs/matlab.html

or google for WPI Matlab

3. Run Matlab on the terminal server windows.wpi.edu with Remote Desktop or using Virtual Lab Web Access

http://www.wpi.edu/academics/CCC/Help/Software/termserv.html

Additional Matlab Resources and Training

Matlab Lecture Notes at blackboard.wpi.edu → SESA Training

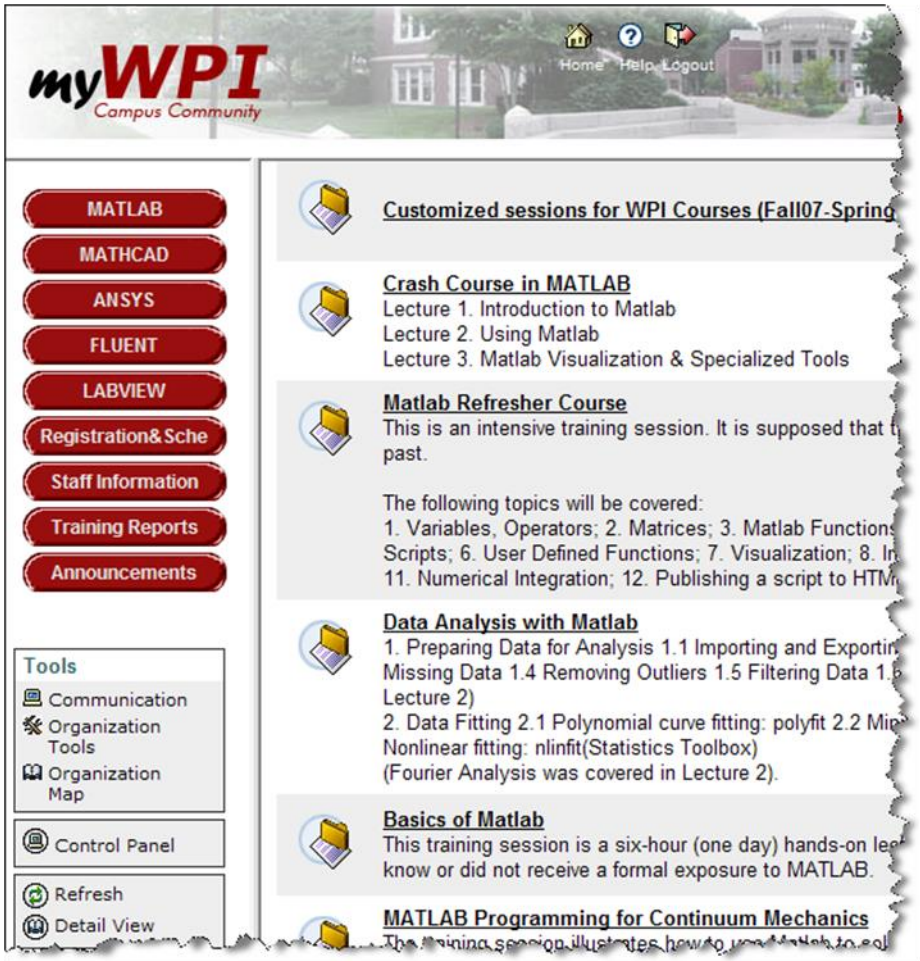
To access SESA Training web site:

1. Log in to the blackboard.wpi.edu
2. Once you have logged into Blackboard, click on the **Community** tab at the top of the page
3. In the **'Organization Search'** text box type **SESA** and click **"Go"**.
4. The organization information will be displayed in the Search Results. Click the down arrows located to the right of the Organization ID and then click on **Enroll**
5. An "Action Successful" enrollment message will appear on the screen. Click the **OK** button to access the organization.

- Registration at:

http://www.wpi.edu/Regi/CCC/sesa.html

- Customized Matlab training sessions, if more than 5-7 students are interested in.



Matlab Resources

MATWORKS web site

Tutorials on Specific Topics and Features

Each video shows a specific feature or application example. Topics range from basic to advanced.
http://www.mathworks.com/products/matlab/examples.html (MATLAB Overview, Getting Started, Mathematics, Graphics and Visualization, Programming)

Interactive MATLAB & Simulink Based Tutorials

An interactive video-based tutorial that introduces MATLAB capabilities and programming
Getting Started (Tutorials for Beginners)
http://www.mathworks.com/academia/student_version/start.html

MATLAB recorded webinars

http://www.mathworks.com/company/events/webinars/index.html

Introduction to MATLAB (for beginners)

Short Simulink examples

http://www.mathworks.com/products/simulink/examples.html

MATLAB Central

You can learn more about how MathWorks products are used by visiting MATLAB Central.
http://www.mathworks.com/matlabcentral/

Experiments with MATLAB, by Cleve Moler

Online textbook that introduces MATLAB through the use of interesting puzzles and problems

http://www.mathworks.com/moler/exm/chapters.html?s_cid=edu_cr_1532

What is Matlab ?

- **MATLAB® is a high-performance language for technical computing.**

- It integrates **computation**, **visualization**, and **programming** in an *easy-to-use environment* where problems and solutions are expressed in familiar mathematical notation.

- **MATLAB** stands for **matrix laboratory.**

- MATLAB is an interactive system whose **basic data element** is an **matrix (array)** that does not require dimensioning.

- This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

MATLAB System

1. **The MATLAB Language**

   (matrix language with loops, functions, data structures..)

2. **Development Environment** (desktop, windows, browsers,..)

3. **The MATLAB Mathematical Function Library**; **Toolboxes**

4. **Graphics** (data visualization, image processing, animation)

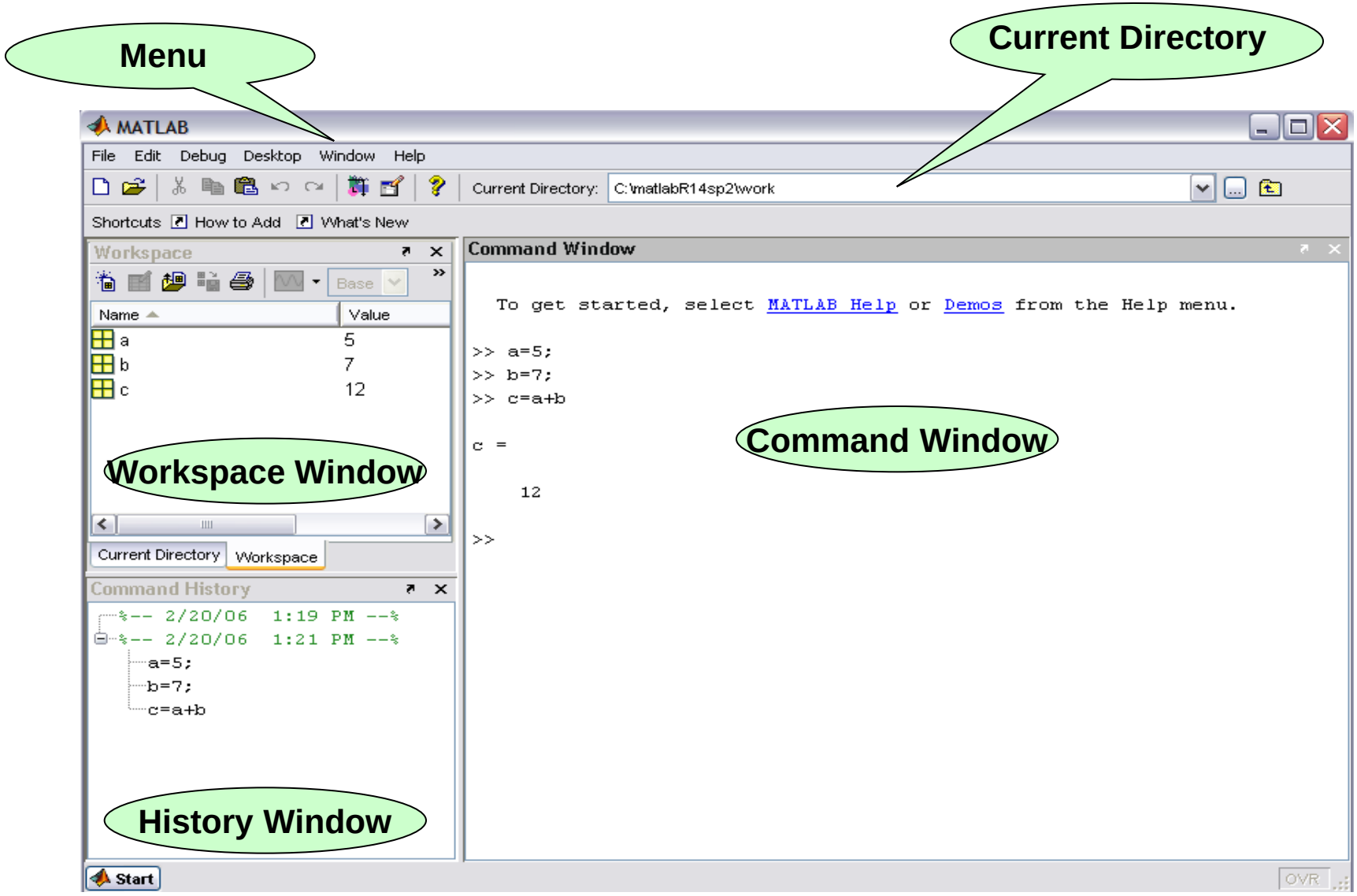
5. **The MATLAB External Interfaces (API)**

   (a library that allows you to write C and Fortran programs that interact with MATLAB)

Starting Matlab

- ***Windows:*** Start menu → Matlab → Matlab
- ***Unix:*** Terminal window→ type **`matlab`**

Matlab Window



Matlab Help

1. **Using HELP menu → MATLAB Help**

   HELP → Using Help Browser

2. **>> helpdesk Opens the Help browser.**

3. **>> help commandname/toolboxname/functionname**

   Ex: >> help sin

4. **>> doc commandname/toolboxname/functionname**

   displays the detailed info in the Help browser.

   Ex: >> doc sin

Other commands:

5. **>> lookfor = helpdesk -> search**

I. Matlab Programming

- Matlab Variables
- Numbers
- Operators
- Functions

.....

Matlab Variables

- A MATLAB variable is essentially a tag that you assign to a value in memory.
- MATLAB does not require any type declarations or dimension statements.
- When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage.
- If the variable already exists, MATLAB changes its contents.
- Variable names consist of a letter, followed by any number of letters, digits, or underscores.
- MATLAB uses only the first 31 characters of a variable name.
- MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters.
- MATLAB stores variables in a part of memory called workspace.
- To view what is stored in a variable type its name.

Types of Variables: MATLAB provides three basic types of variables:

Local Variables
Global Variables
Persistent Variables

Matlab Variables

Rules for variable names:

- Make Sure Variable Names Are Valid
- Don't Use Function Names for Variables
- Check for Reserved Keywords
- Avoid Using i and j for Variables

Syntax:

```
varialeName=Value;
```

Example:

```
>> a=5;
>> b=7;
>> c=a+b
>> method='linear'
```

How to remove a variable from workspace:

```
>> clear variableName
>> clear
```

- removes all variables from the workspace (!!!!)

ans = default variable, when the result is not assign to a variable

Exercise: 1. Define a1=8 and b2=8, c1=a1+b2

2. Other commands: variable= input('prompt') `(>>help input)`

```
>> a3=input('a3=')
```

Numbers

1. **Integers**

Matlab has 8,16,32, 64 bit , signed and unsigned integer data types.

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

2. **floating-point numbers**: in either **double-precision** (64 bits) or **single-precision** (32 bits) format.

(`double, single`)

3. **Imaginary numbers** use either i or j as a suffix

Ex: 5+2i 6 -3.14159j

4. **Infinity** and **NaN**

Useful related functions: **`realmax , realmin, intmin , intmax, eps`**

isnan, **`format`**

Numbers

eps – function that returns the distance from 1.0 to the next largest double-precision number

d = eps(X) is the positive distance from abs(X) to the next larger in magnitude floating point number of the same precision as X.

inf Infinity : division by zero and overflow, which lead to results too large to represent as conventional floating-point values.

```
ex: 1/0, 1.e1000
```

NaN Not-a-Number: a result of mathematically undefined operations like 0.0/0.0 and inf-inf.

Related commands: `edit`

Numbers

1. - decimal notation (with an optional decimal point and leading plus or minus sign)

```
Ex:    3          -99          0.0001
```

2. . Scientific notation (uses the letter **e** to specify a **power-of-ten scale factor**.)

```
Ex: 1.60210e-5    6.02252e10
```

All numbers are stored internally using the long format specified by the IEEE floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly 10-308 to 10+308.

Useful related functions: **realmax** , **realmin**, **intmin** , **intmax**, **eps**

isnan, **format**

Strings

```
>> name ='John'    % create a string variable
>> errorMessage= 'negative number'
```

Useful related functions: `num2str`, `str2num`, `strcat`

Operators

Expressions use familiar **arithmetic operators** and precedence rules.

+	Addition
-	Subtraction
*	Multiplication
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

Functions

1. Standard elementary mathematical functions

```
>> help elfun
```

Trigonometric (`sin, cos`)

Exponential (`exp, log`)

Complex (`abs, angle`)

Rounding and remainder (`round`)

2. Elementary matrices and matrix manipulation.

```
>> help elmat
```

3. Specialized math functions.

```
>>  help specfun
```

pi	3.14159265...
i	Imaginary unit, $\sqrt{-1}$
j	Same as i
eps	Floating-point relative precision, $\varepsilon = 2^{-52}$
Inf	Infinity
NaN	Not-a-number

Functions

1. Built-in functions (Ex. sqrt, sin)

Some of the functions, like `sqrt` and `sin`, are built in.

Built-in functions are part of the MATLAB core

They are very efficient

The computational details are not readily accessible.

(you cannot see the code)

2. Function implemented in M-files (ex. `factorial`, `mean`, `det`)

You can see the code and even modify it, if you want.

Syntax:

```
>> outputArgs = functionName(inputArgs)
```

Related commands: `edit`

We talked about:

- ✓ Matlab Variables
- ✓ Numbers
- ✓ Operators
- ✓ Functions

III. Matlab Programming

....

- Matrices
- Operators
- Functions

.....

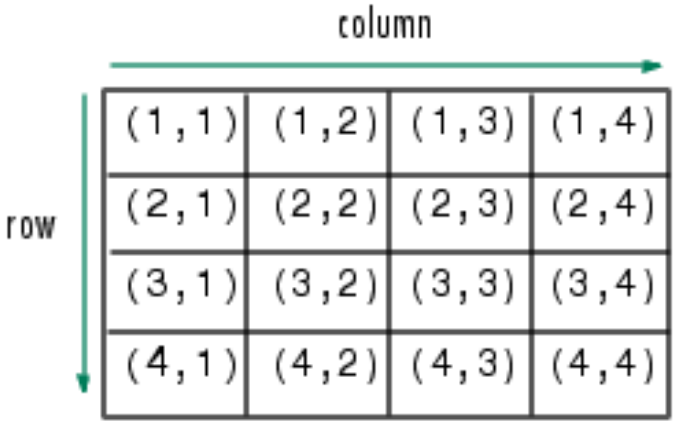
Matrix & basic matrix functions

Define a matrix:

1. Type the matrix
2. Use Specialized Matrix Functions

Matrix Manipulation

Matrix Functions



Matrix: Define a matrix

1. Type the matrix

- Separate the elements of a row with **blanks** or **commas**.
- Use a **semicolon**, **;** , to indicate the end of each row.
- Surround the entire list of elements with **square brackets**, [ ].

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$$

```
1. >> A=[1 2 3; 21 22 23; 31 32 33];

2. >> A=[1, 2, 3; 21, 22, 23; 31, 32,33];

3. >> A(1,1)=1; A(1,2)=12; A(1,3)=13; A(2,1)=21; A(2,2)=22;
A(2,3)=23;
```

Basic matrix information: **size** (size of a matrix)

```
>> [m,n] = size(X)
```

Matrix: Define a matrix

2. Use Specialized Matrix Functions

Function	Description
ones	Create a matrix or array of all ones.
zeros	Create a matrix or array of all zeros.
eye	Create a matrix with ones on the diagonal and zeros elsewhere.
diag	Create a diagonal matrix from a vector.
rand	Create a matrix or array of uniformly distributed random numbers.
randn	Create a matrix or array of normally distributed random numbers and arrays.

```
>> B=eye(3)
```

```
B =
     1     0     0
     0     1     0
     0     0     1
```

```
>> C=ones(2,3)
```

```
C =
     1     1     1
     1     1     1
```

```
>> D=rand(4,1)
```

```
D =
    0.4289
    0.3046
    0.1897
    0.1934
```

Matrix: Accessing Matrix Elements

- **individual element**

```
>> A(2,2)
```

```
ans =

    22
```

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$$

- **column**

```
>>  A(:,2)
```

```
ans =

     2
    22
    32
```

- **row**

```
>> A(2,:)
```

```
ans =

    21    22    23
```

- **group of elements**

```
>> A(2,1:2)
```

```
ans =

    21    22
```

: (colon) → all elements

first element : step: last element

Matrix: Operations

+	Addition	**A+B**
-	Subtraction	**A-B**
*	Multiplication	**A*B**
/	Division	**A/B**
\	Left division (described in "Mat	**A\B**
^	Power	**A^B**
'	Complex conjugate transpose	**A'**
( )	Specify evaluation order	

.*	Element-by-element multiplication	**A.*B**
./	Element-by-element division	**A./B**
.\	Element-by-element left division	**A.\B**
.^	Element-by-element power	**A.^B**
.'	Unconjugated array transpose	**A.'**

Matrix: Operations

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 2 & 3 & 1\\ 2 & 2 & 2\end{bmatrix},\qquad \mathbf{B}=\begin{bmatrix}10 & 20 & 30\\ 11 & 21 & 31\\ 1 & 2 & 3\end{bmatrix},$$

```
>> A=[1 2 3; 2 3 1; 2 2 2];
>> B= [10 20 30; 11 21 31; 1 2 3];
```

```
>> A*B
```

```
ans =
    35    68   101
    54   105   156
    44    86   128
```

$$\mathbf{A}=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix},\qquad \mathbf{B}=\begin{bmatrix}b_{11} & b_{12} & b_{13}\\ b_{21} & b_{22} & b_{23}\\ b_{31} & b_{32} & b_{33}\end{bmatrix},$$

$$\mathbf{A}\cdot*\mathbf{B}=\begin{bmatrix}a_{11}\cdot b_{11} & a_{12}\cdot b_{12} & a_{13}\cdot b_{13}\\ a_{21}\cdot b_{21} & a_{22}\cdot b_{22} & a_{23}\cdot b_{23}\\ a_{31}\cdot b_{31} & a_{32}\cdot b_{32} & a_{33}\cdot b_{33}\end{bmatrix}$$

```
>> A.*B
```

```
ans =
    10    40    90
    22    63    31
     2     4     6
```

Element by element multiplication

Matrix: Functions

Few matrix functions :

det -Determinant

trace -Sum of diagonal elements

linsolve-Solve linear systems of equations (using LU factorization)

eig - Find eigenvalues and eigenvectors

eigs -Find largest eigenvalues and eigenvectors of a sparse matrix

sdv - Singular value decomposition

balance -Improve accuracy of computed eigenvalues

**** - Linear equation solution

(**X = A\B** is the solution to the equation AX = B

computed by Gaussian elimination)

cond - condition number (the ratio of the largest singular value to the smallest)

>> **doc elmat**

Matrix: Solution of a linear system

X = A\B is the solution to the equation AX = B

computed by Gaussian elimination

A=square matrix

$$\begin{aligned} x_1 + x_2 + x_3 &= 2 \\ x_1 + 2x_2 + 3x_3 &= 5 \\ x_1 + 3x_2 + 6x_3 &= 7 \end{aligned} \qquad \begin{aligned} x_1 &= ? \\ x_2 &= ? \\ x_3 &= ? \end{aligned}$$

\ = ldivide

```
>>A=[1,1,1;1,2,3;1,3,6];
>>b=[2;5;7];
1. >>x=A\b
2. >> x=linsolve(A,b)
3. >> x =inv(A)*b
```

→

```
X =
    -2
     5
    -1
```

(**X = A\B** is the solution to the equation Ax = b computed by Gaussian elimination)
linsolve-Solve linear systems of equations (using LU factorization)

We talked about:

….

- ✓ Matrices
- ✓ Operators
- ✓ Functions

…..

III. How to write a program (M-files)

IV. How to plot data

M-files

- **Files that contain code in the MATLAB** language are called *M-files*.
- You create M-files using a text editor.
- Use a M-file as any other MATLAB function or command.
- A M-file is a plain text file.

Two kinds of **M-files**:

Scripts

do not accept input arguments or *return output arguments*

operate on data in the workspace.

Functions

can accept input arguments and *return output arguments*

internal variables are local to the function.

M-files: Scripts

- do not accept input arguments or *return output arguments*
- operate on data in the workspace.

>> edit myScript

type the code

File/Save

>> myScript

(to run the script type its name)

Example →

```
%myScript


% this is my script file
% calculates the determinant of A
clear
N=10;
theta=pi/N;
r=sin(theta);
```

% - indicates a comment

plot

Sintax:

```
plot(y);      plot(x,y);     plot(x,y,s)
```

The plot function has different forms, depending on the input arguments.

If y is a vector, `plot(y)` produces a piecewise linear graph of *the elements of y* versus the *index of the elements of y*.

If you specify two vectors as arguments, `plot(x,y)` produces a *graph of y versus x*.

Matlab - Plotting

```
plot(x,y, s);
```

s allows to plot : colors, symbols, different lines

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
		….			

```
plot (x,y,'c+:')
```
plots a cyan dotted line with a plus at each data point;

Matlab - Plotting

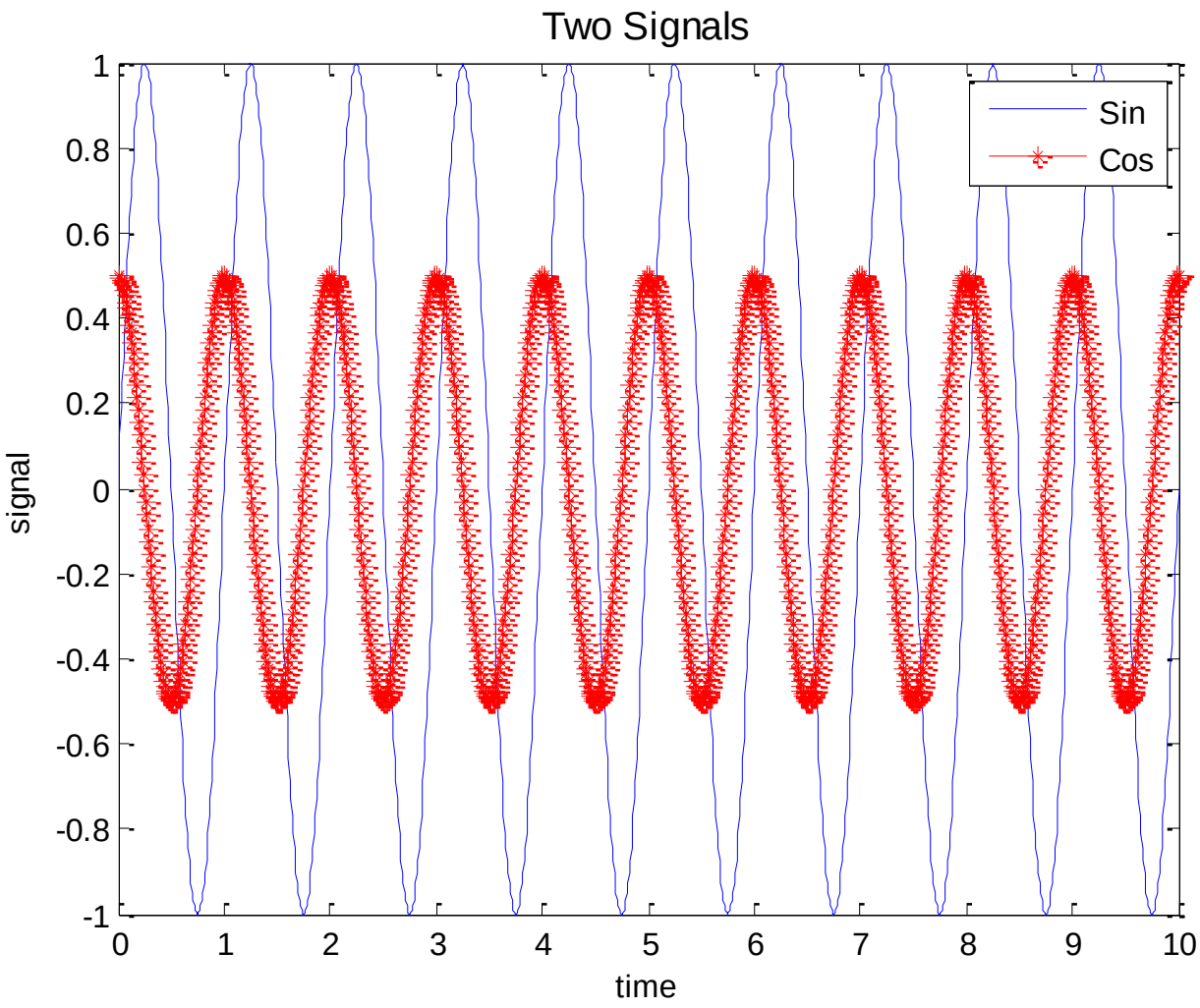
```
clear
t=0:0.01:10; % time seconds
signalSin=sin(2*pi*t); % signal1 - frequency =1 Hz
signalCos=0.5*cos(2*pi*t); % signal2 - frequency =1 Hz

figure
plot(t,signalSin);
```

plot2signals.m

```
hold on
plot(t,signalCos, '-*r');

xlabel('time');  ylabel('signal');
legend('Sin', 'Cos');
title('Two Signals','FontSize',12)
```

Other commands: figure xlabel ylabel legend, title

Matlab - Plotting



line

Create line object

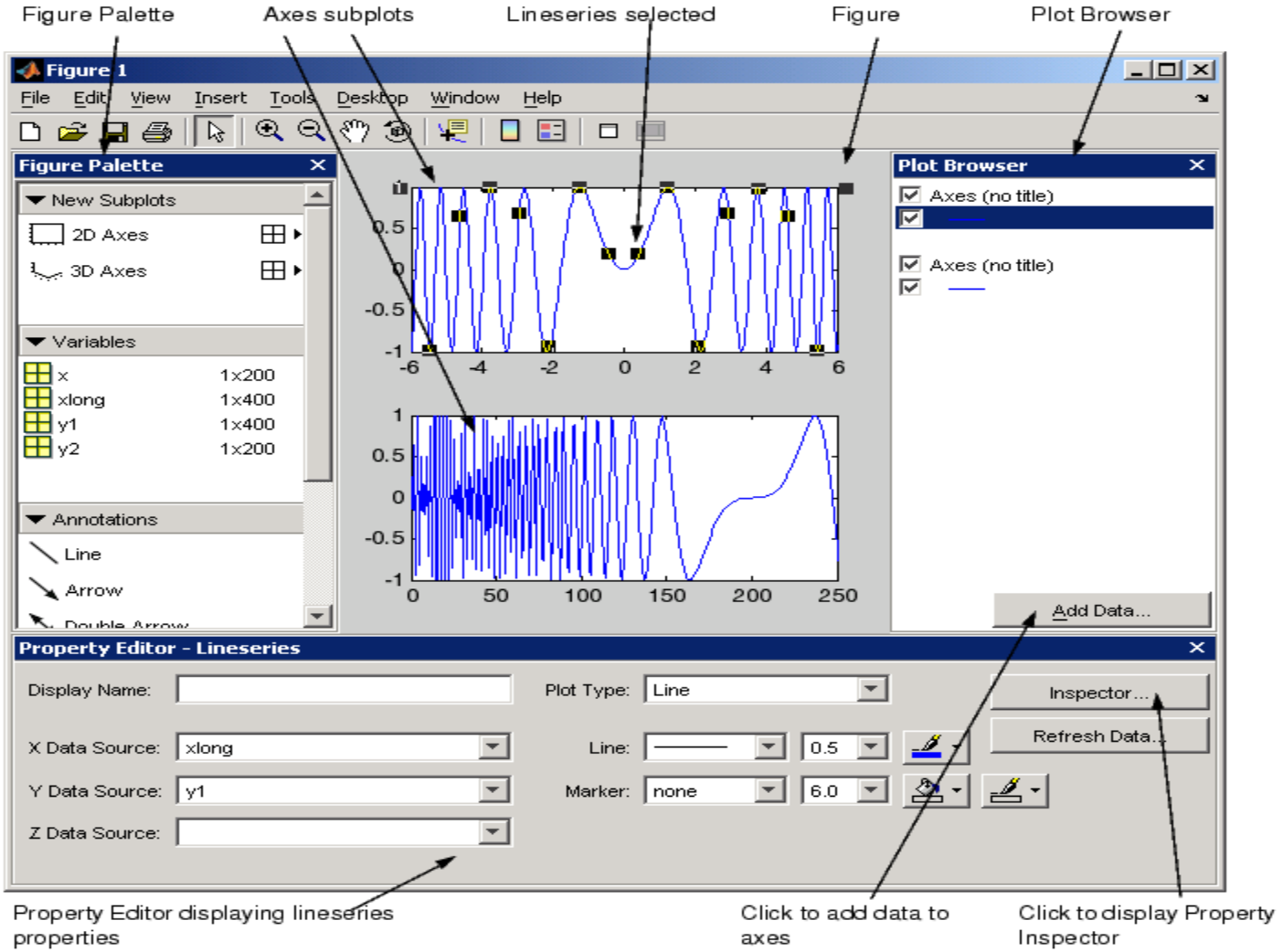
Syntax

```
line(X,Y)
line(X,Y,Z)
line(X,Y,Z,'PropertyName',propertyvalue,...)
line('XData',x,'YData',y,'ZData',z,...)
h = line(...)
```

```
>> P1=[1,2];
>> P2=[3,4];
>> P3=[3,6];
>> line([P1(1), P2(1), P3(1)], [P1(2), P2(2), P3(2)])
```

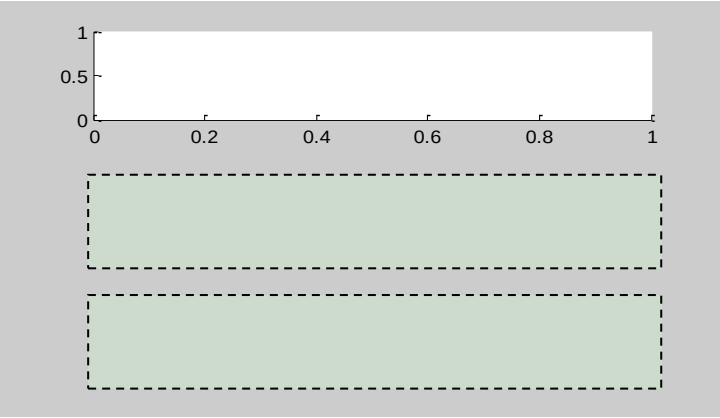
Visualization - Interactive editing

show plot tools

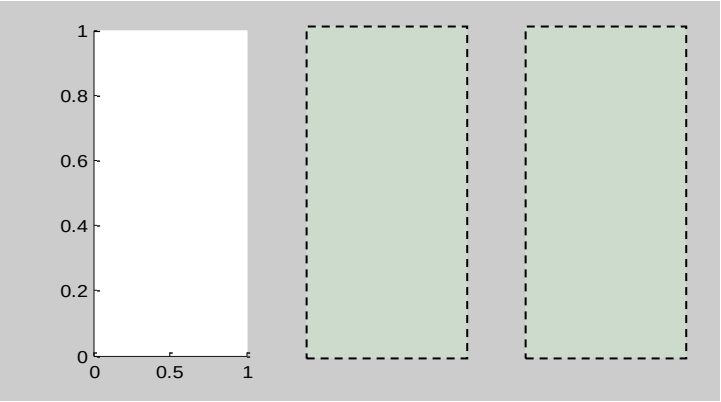


Visualization - subplot

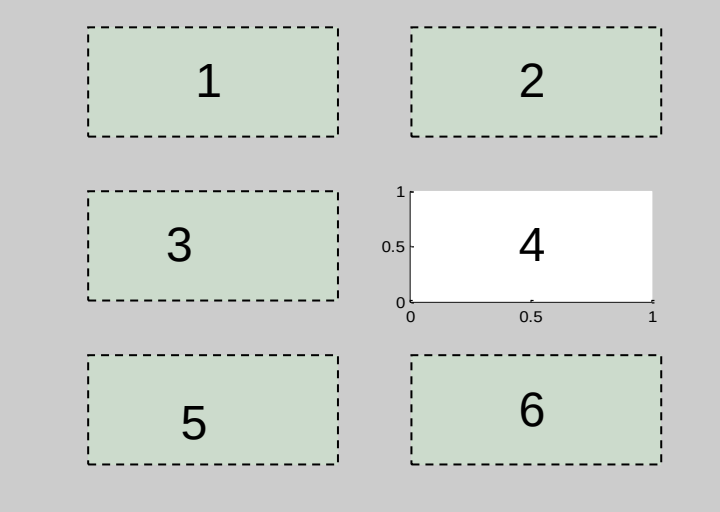
```
>> subplot(3,1,1)
```




```
>> subplot(1,3,1)
```




```
>> subplot(3,2,4)
```



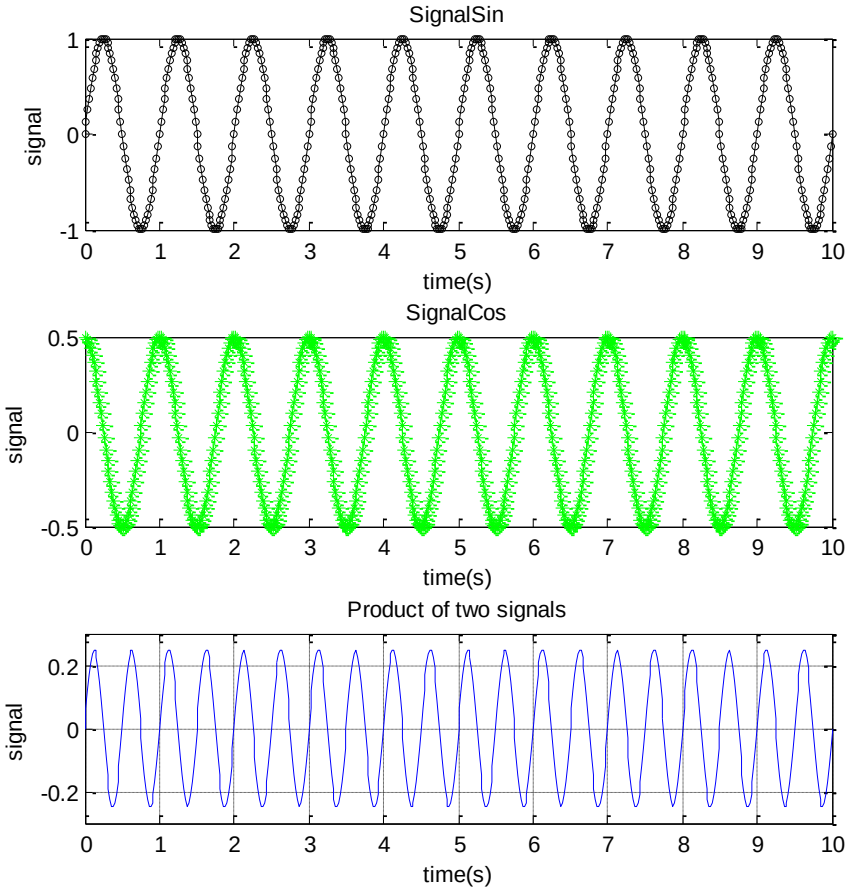
`subplot(m,n,q)`

* breaks the figure into a `m x n` matrix of windows.

`q` = the current window

`plot3Windows.m`

Visualization - subplot



plot3Windows.m

```
clear; % clear the workspace
close; % close previous figures

% Generate the signals
t=0:0.02:10; % time seconds
signalSin=sin(2*pi*t); % signal1 - frequency =1 Hz
signalCos=0.5*cos(2*pi*t); % signal2 - frequency =1 Hz
signal3=signalSin.*signalCos;

% Plot the signals
figure; subplot(3,1,1)
plot(t,signalSin,'-ok', 'MarkerSize',2);
title('SignalSin')
xlabel('time(s)');  ylabel('signal');

subplot(3,1,2)
plot(t,signalCos, '-*g');
title('SignalCos')
xlabel('time(s)');  ylabel('signal');

subplot(3,1,3)
plot(t,signal3);
grid
title('Product of two signals')
xlabel('time(s)');  ylabel('signal');
axis([0 10 -0.3 0.3]); % sintax: axis([XMIN XMAX YMIN YMAX])
```

SUMMARY

1. Getting started; Matlab Help;
2. Variables;
3. Operators;
4. Matlab functions;
5. Matrices;
6. Scripts;
7. Basic plotting.
8. User Defined Functions
9. Importing Data
10. Simulink

FUNCTIONS

✓ **Matlab functions & Toolboxes**

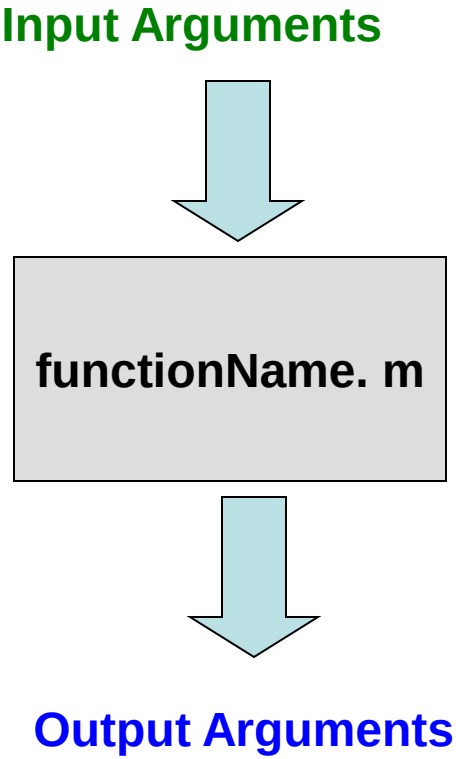
- **User Defined Functions**

There are essentially two ways to create a new function in MATLAB:

1. **in a file saved to permanent storage. (.m file)**
2. **in a command entered at run-time (inline, anonymous)**

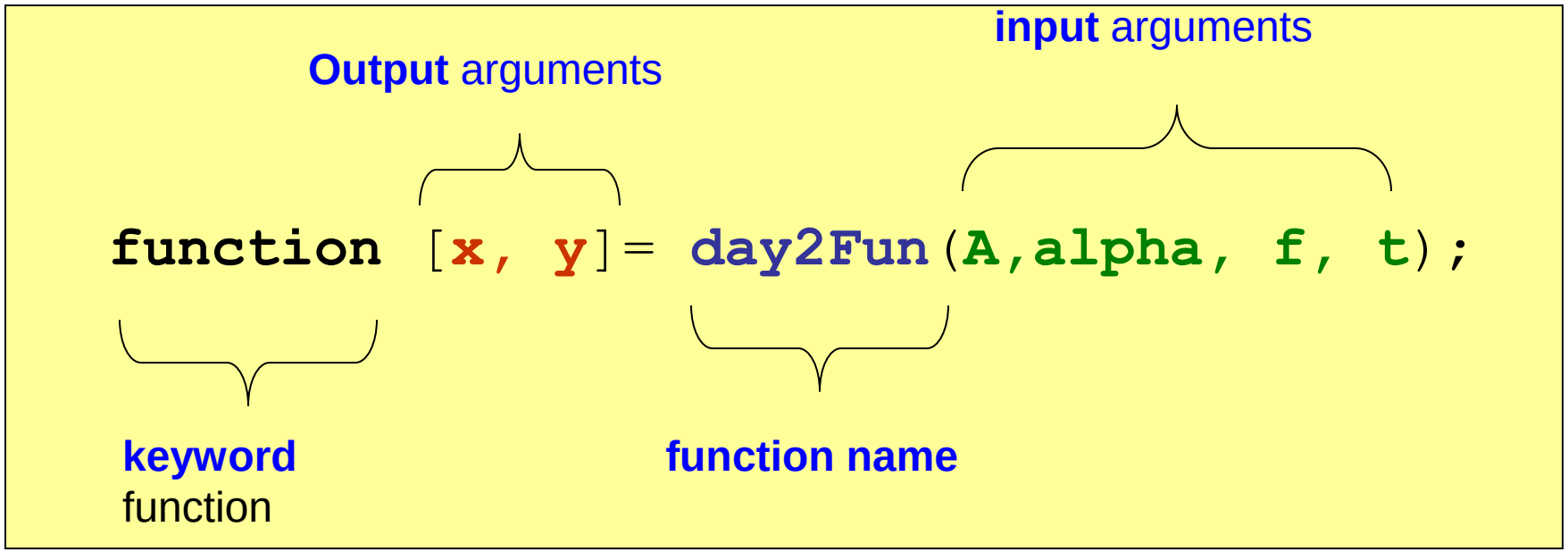
FUNCTIONS

- Functions are M-files that can accept input arguments and return output arguments.
- The M-file and function should have the same name.
- Each M-file function has an area of memory, called the function workspace, separate from the MATLAB base workspace, in which it operates.



FUNCTIONS

Function definition:



Calling the function:

```
>> [x1, y1]= day2Fun (A1,alpha1, f1,
t1);
```

FUNCTIONS

Function definition:

day2Fun.m

exampleFun.m

```
function [x, y]=day2Fun(A,alpha, f, t);

%% function [x, y]=day2Function(A,alfa, f, t);
%% Lecture2: user defined functions in Matlab

amplitude=A*exp(-alpha*t);
x=amplitude.*sin(2*pi*f*t);
y=amplitude.*cos(2*pi*f*t);
```

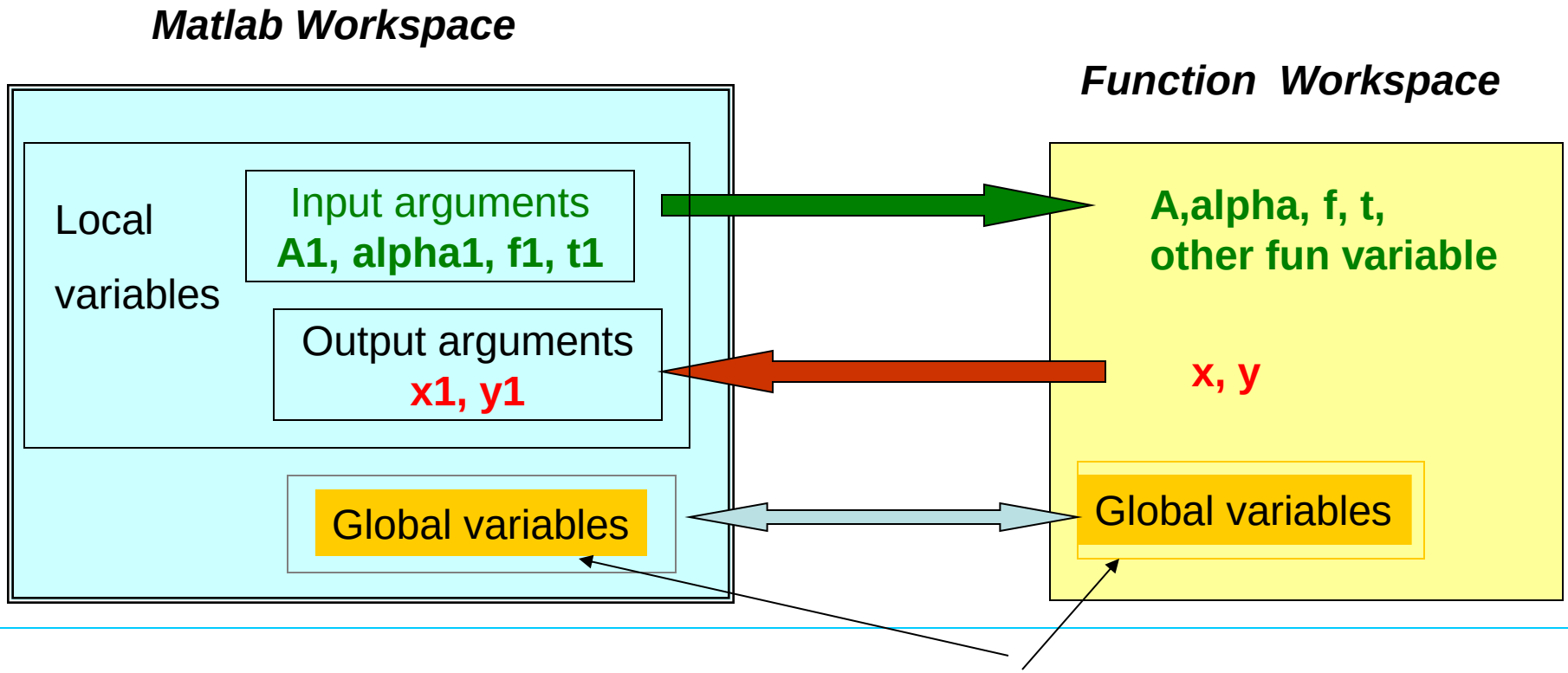
Main code

```
>> clear all
>> A1=5;
>> alpha1=0.5;
>> f1=1;
>> t1=0:0.01:10;
>> [x1, y1]=day2Fun(A1,alpha1, f1, t1);
```

```
>> figure
>> plot(t1,x1, t1,y1);
>> legend('x1', 'y1')
```

Function: Input and output arguments

```
function [x, y]= day2Fun(A,alpha, f, t);
>> [x1, y1]= day2Fun (A1,alpha1, f1, t1);
```



Global variables:

day2Fun2.m

exampleFun2.m

FUNCTIONS: How to document a function

the function-declaration line

keyword **function**

function name `trace`

order of arguments.

```
function t = trace(a)
%TRACE  Sum of diagonal elements.
%   TRACE(A) is the sum of the diagonal elements of A,
which is
%   also the sum of the eigenvalues of A.
%
%   Class support for input A:
%      float: double, single
%   Copyright 1984-2004 The MathWorks, Inc.
%   $Revision: 5.8.4.1 $  $Date: 2004/04/10 23:30:11 $

t = sum(diag(a));
```

executable code

The help text

```
>> trace(A)
>> results= trace(A);
```

Function: Inline functions

There are essentially two ways to create a new function in MATLAB:

1. in a command entered at run-time (inline and anonymous)
2. or in a file saved to permanent storage.

inline function, feval

```
>> f=inline('x^2+y^2', 'x', 'y')
f =
     Inline function:
     f(x,y) = x^2+y^2
>> m=f(1,2)
m =5
```

1.1 Importing and Exporting Data

1.1.2 Supported File Formats

File Format	File Content	Extension	Functions
MATLAB formatted	Saved MATLAB workspace	.mat	load, save
Text	Text	any	textscan
	Text	any	textread
	Delimited text	any	dlmread, dlmwrite
	Comma-separated numbers	.csv	csvread, csvwrite
Extended Markup Language	XML-formatted text	.xml	xmlread, xmlwrite
Audio	NeXT/SUN sound	.au	auread, auwrite
	Microsoft WAVE sound	.wav	wavread, wavwrite
Movie	Audio/video	.avi	aviread
Scientific data	Data in Common Data Format	.cdf	cdfread, cdfwrite
	Flexible Image Transport System data	.fits	fitsread
	Data in Hierarchical Data Format	.hdf	hdfread
Spreadsheet	Excel worksheet	.xls	xlsread, xlswrite
	Lotus 123 worksheet	.wk1	wk1read, wk1write
Graphics	TIFF image	.tiff	imread, imwrite
	PNG image	.png	same
	HDF image	.hdf	same
	BMP image	.bmp	same
	JPEG image	.jpeg	same

1. Importing and Exporting Data

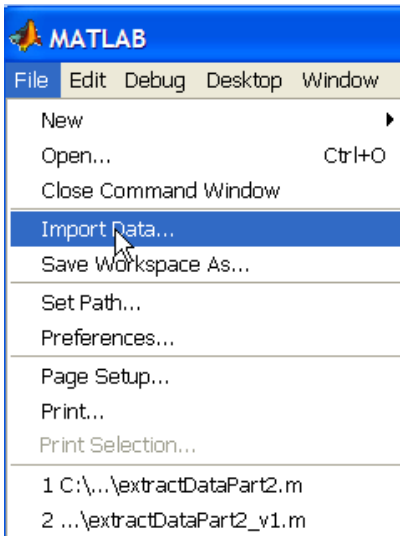
- **using the Import Wizard**
- **save , load**
- **dlmread , dlmwrite**
- **xlsread, xlswrite**
- **fopen, , fscanf, fprintf**

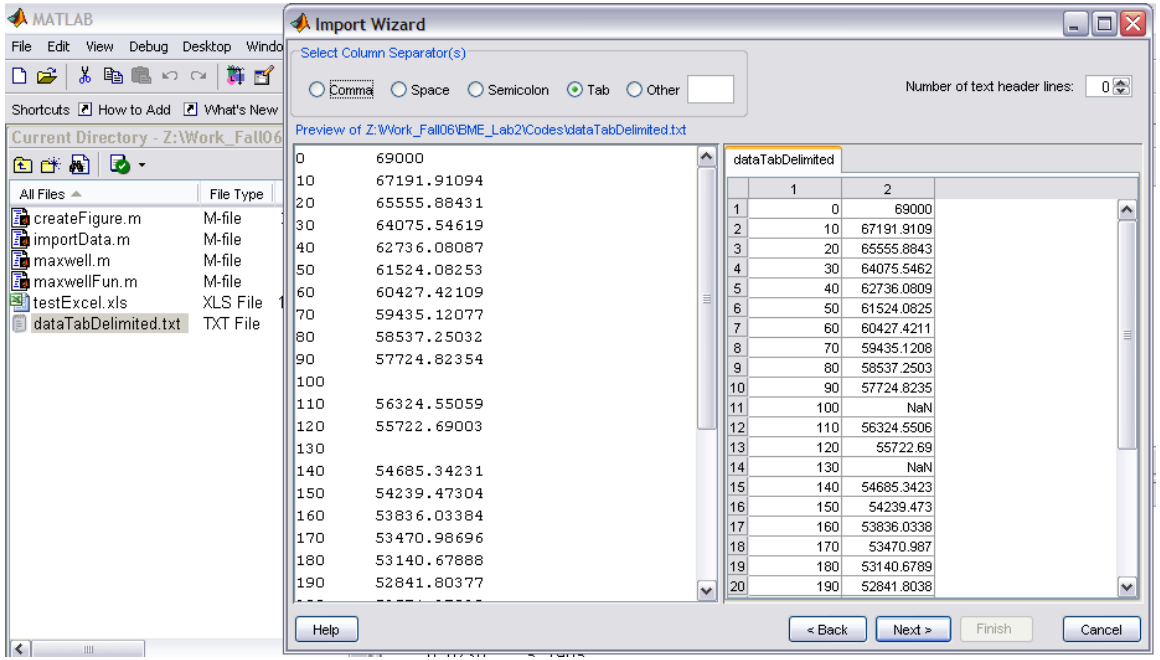
1.1 Importing and Exporting Data

1.1.1 Using the Import Wizard with Text Data

File → Import Data

or >> **uiimport**





1.1.2 Supported File Formats

Wizard: missing data: NaN (Not-a-Number.)

1.1 Importing & exporting data: dlmread & dlmwrite

dlmread , **dlmwrite** - Read/Write ASCII delimited file.

Syntax

```
M = dlmread(filename)
M = dlmread(filename, delimiter)
M = dlmread(filename, delimiter, R, C)
M = dlmread(filename, delimiter, range)
```

prepareData.m

fileName

matrix to be saved

add at the end of existing file

```
data=dlmread('myFile');
dlmwrite(fileName, data,'delimiter', '\t', '-append');
dlmwrite(fileName, data, '\t')
```

Delimiter = tab

```
data=dlmread('dataCSV.csv', ',',2,0);
```

dlmread reads from the ASCII-delimited numeric data file filename to output matrix M. The delimiter separating data elements is inferred from the formatting of the file.

Comma (,) is the default delimiter.

1.1 Importing & exporting data: dlmread & dlmwrite

dlmread , **dlmwrite** - Read/Write ASCII delimited file.

Syntax

```
M = dlmread(filename)
M = dlmread(filename, delimiter)
M = dlmread(filename, delimiter, R, C)
M = dlmread(filename, delimiter, range)
```

```
for i=1:4
    fileName=['results', num2str(i), '.csv']
    data(i,:,:)=dlmread(fileName, ',',2,0);
end
```

1.1 Importing & exporting data: xlswrite, xlsread

xlswrite, xlsread - Write/Read Excel file.

```
xlswrite('filename', data)
xlsread('filename')
```

To import Excel data by wizard (uiimport) the file should have the extension **.xls**

SIMULINK

SIMULINK

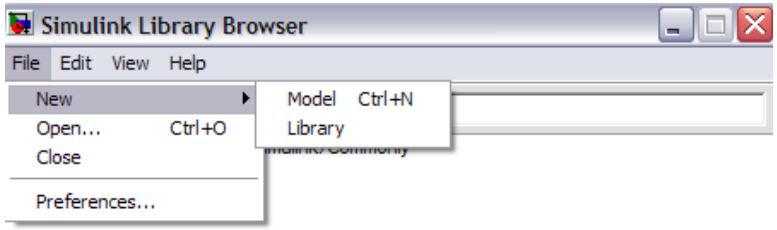
Simulink® is software for modeling, simulating, and analyzing dynamic systems.

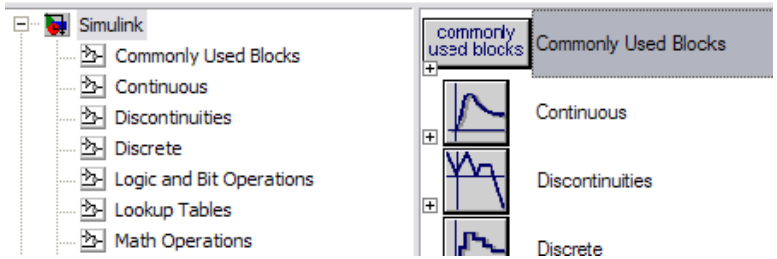
It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two.

Systems can also be multirate, i.e., have different parts that are sampled or updated at different rates.

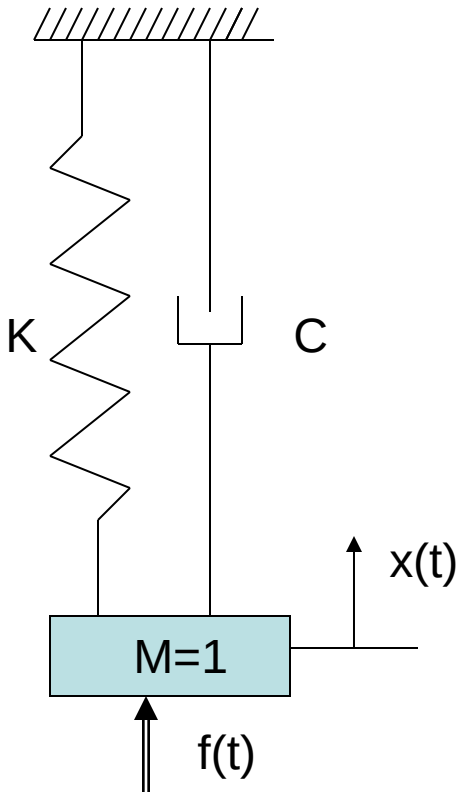
For modeling, Simulink provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations.

```
>> simulink
```





Response of a SDOF (nDOFs) system



Linear systems: **time and laplace domains**

Time Domain

$$\ddot{x} + C_d \dot{x} + Kx = f(t)$$

$$x(t_0), \dot{x}(t_0)$$

$$x_1 = x$$

$$x_2 = \dot{x} = \dot{x}_1$$

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -C_d x_2 - K x_1 + f(t)$$

$$y = x$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -K & -C_d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [0] f(t)$$

Laplace Domain

F(s) → $H(s) = \dfrac{1}{\left(s^2 + C_d s + K\right)}$ → X(s)

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}f$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}f$$

parameters.m

Linear systems: **time and laplace domains**

Time Domain

$$\ddot{x} + C_d \dot{x} + Kx = f(t)$$

$$x(t_0), \dot{x}(t_0)$$

A B

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -K & -C_d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [0] f(t)$$

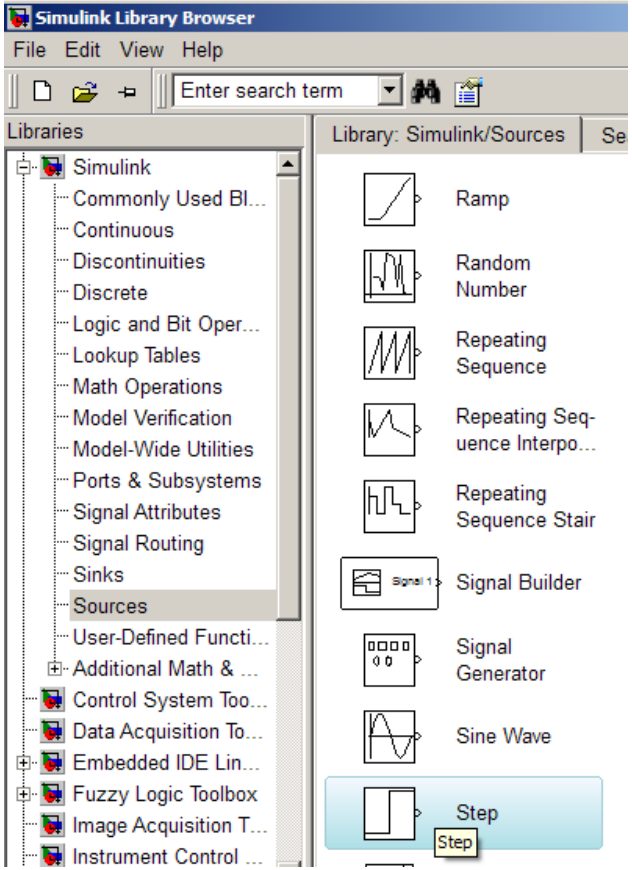
C D

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}f$$

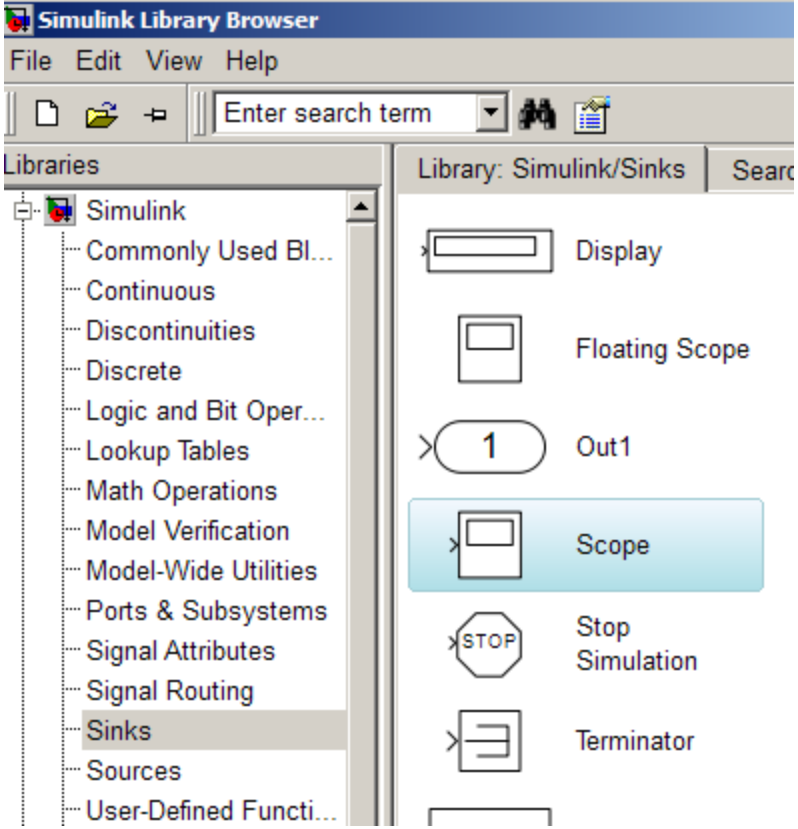
$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}f$$

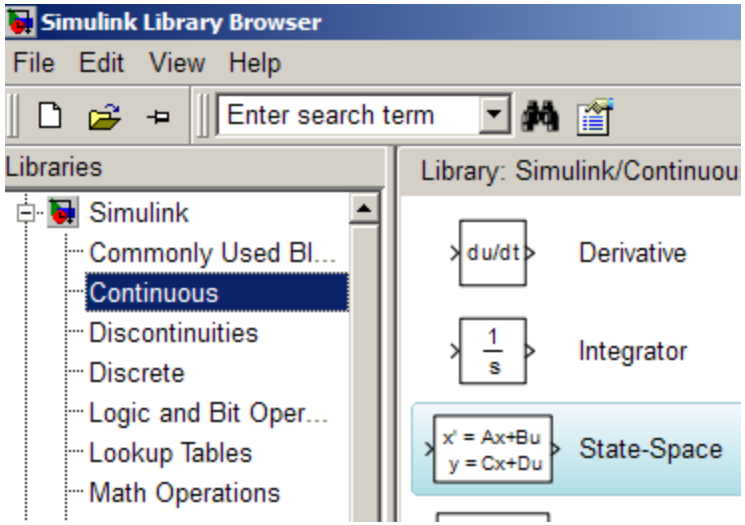
➔ y= x

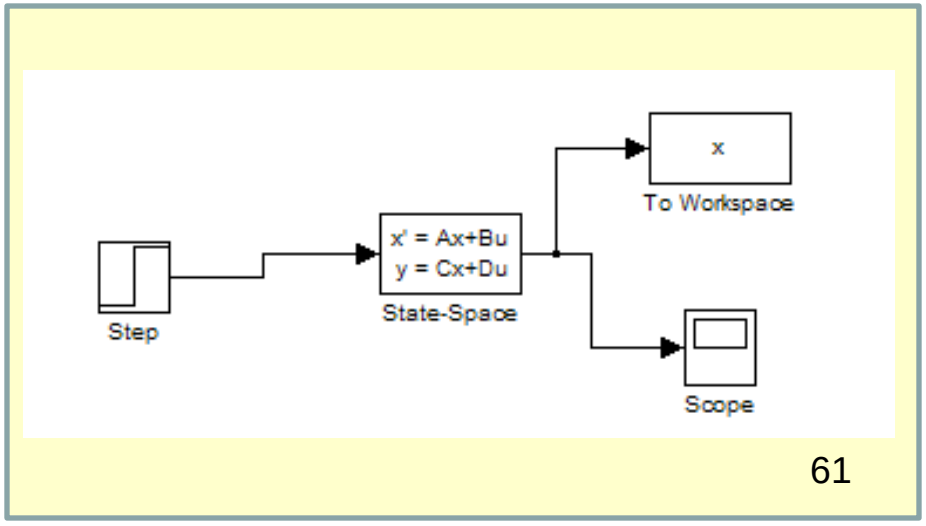
parameters.m


Simulink Library Browser
File Edit View Help
Enter search term
Libraries
Library: Simulink/Sources
Simulink
Commonly Used Bl...
Continuous
Discontinuities
Discrete
Logic and Bit Oper...
Lookup Tables
Math Operations
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
User-Defined Functi...
Additional Math & ...
Control System Too...
Data Acquisition To...
Embedded IDE Lin...
Fuzzy Logic Toolbox
Image Acquisition T...
Instrument Control ...
Ramp
Random Number
Repeating Sequence
Repeating Seq-uence Interpo...
Repeating Sequence Stair
Signal Builder
Signal Generator
Sine Wave
Step
Step

Example 1:

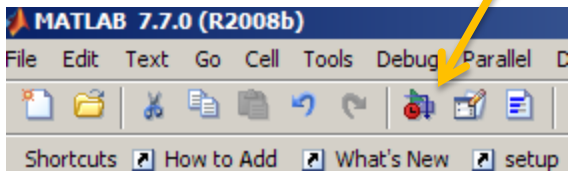

Simulink Library Browser
File Edit View Help
Enter search term
Libraries
Library: Simulink/Sinks
Simulink
Commonly Used Bl...
Continuous
Discontinuities
Discrete
Logic and Bit Oper...
Lookup Tables
Math Operations
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
User-Defined Functi...
Display
Floating Scope
1
Out1
Scope
STOP
Stop Simulation
Terminator


Simulink Library Browser
File Edit View Help
Enter search term
Libraries
Library: Simulink/Continuou
Simulink
Commonly Used Bl...
Continuous
Discontinuities
Discrete
Logic and Bit Oper...
Lookup Tables
Math Operations
du/dt
Derivative
1
s
Integrator
x' = Ax+Bu
y = Cx+Du
State-Space

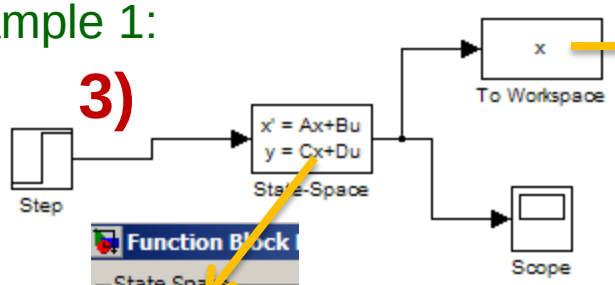

x
To Workspace
x' = Ax+Bu
y = Cx+Du
State-Space
Step
Scope

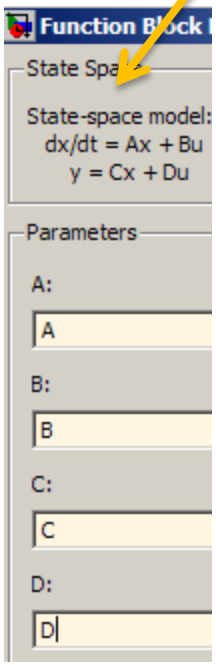
0) Start simulink

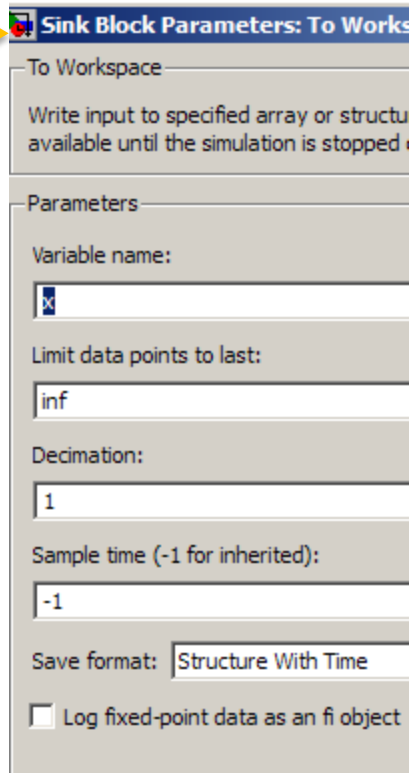
Example 1:



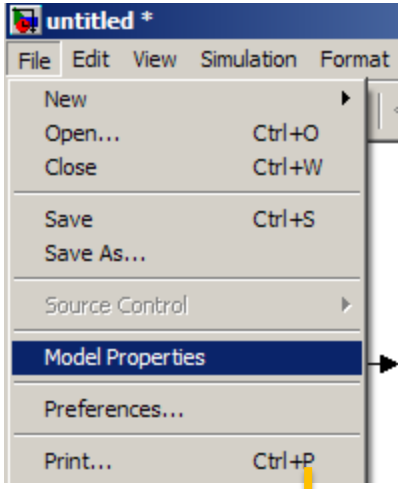
3)



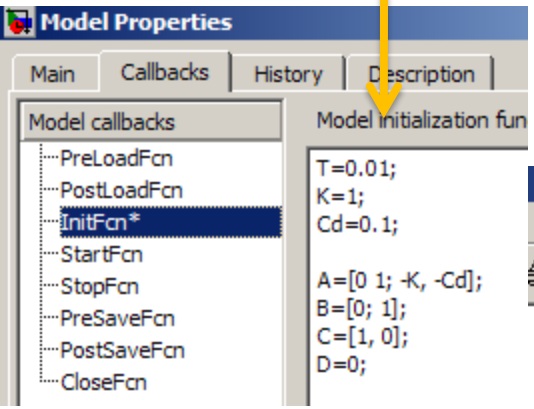


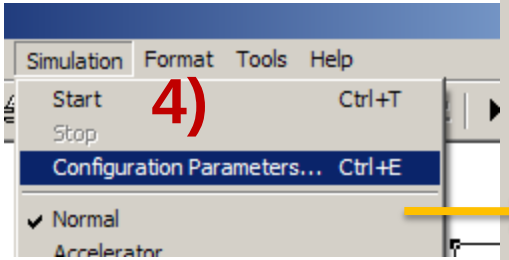


1)

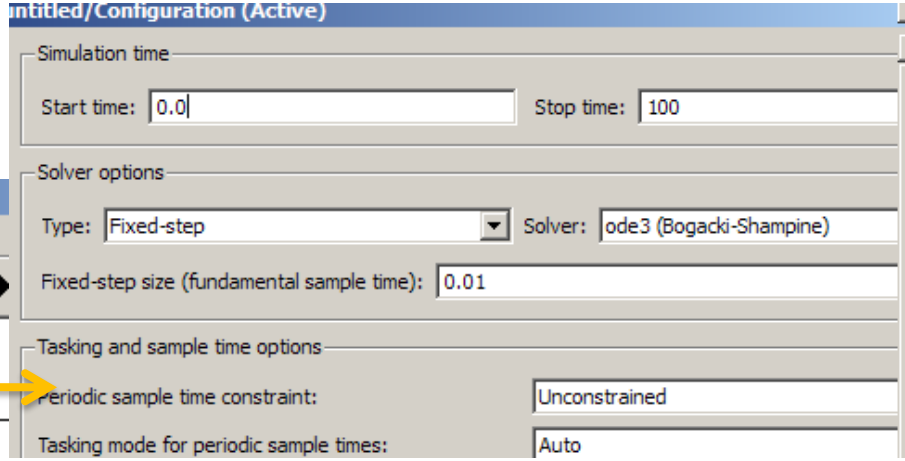


2)

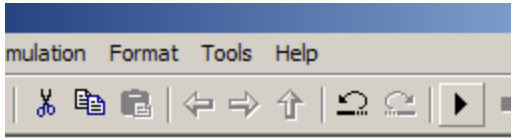




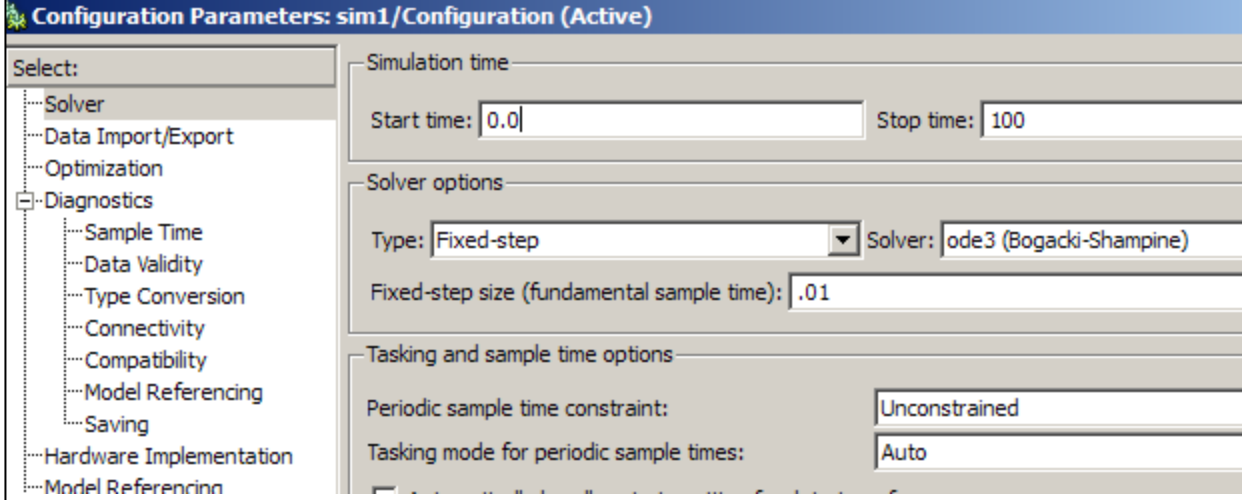
4)



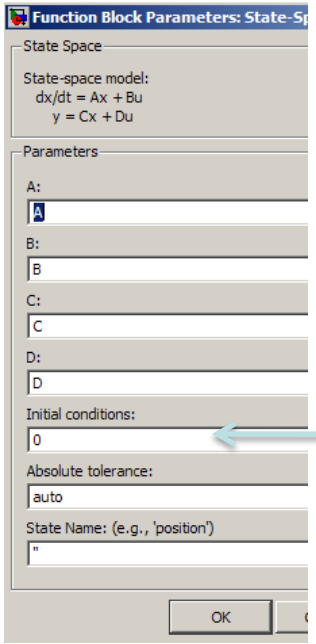
5) Run the model

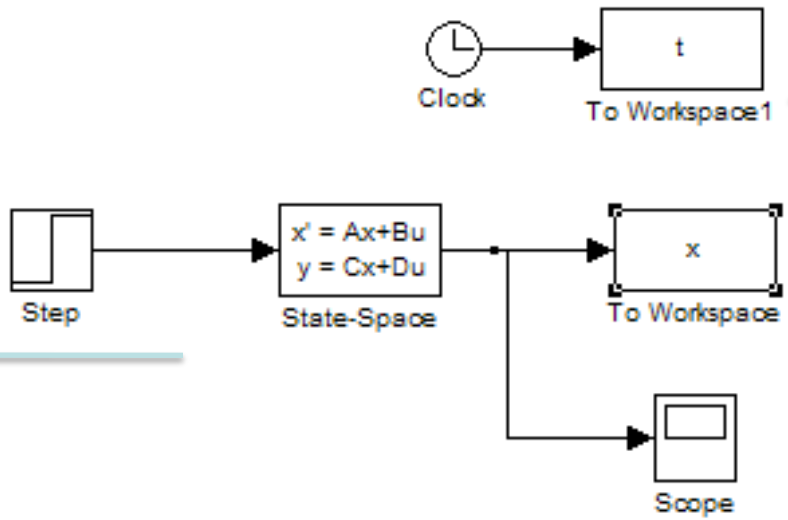


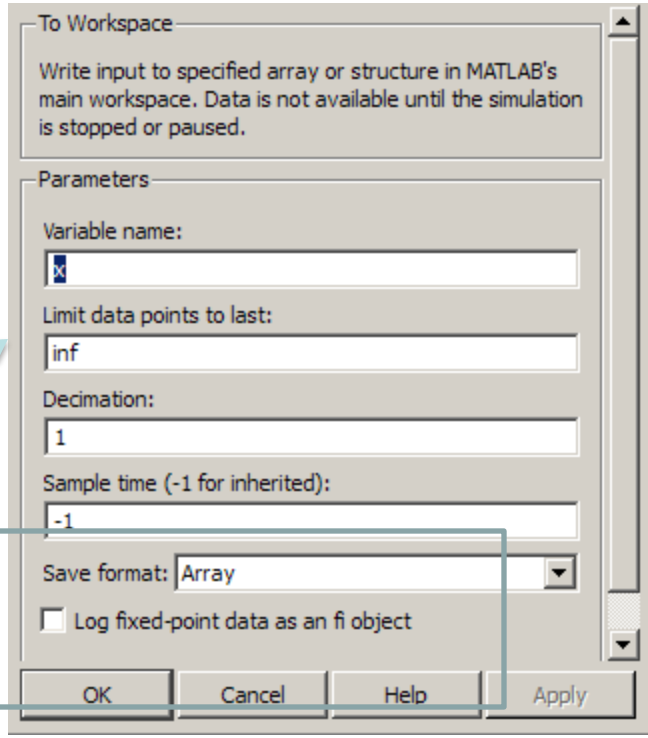
```
clear all; close all;

% set the sample perion in sec.
Ts = 0.01;

% Set the parameter values
K=1;
Cd=.1;

% Fill in the state space configuration
A = [0,1;-K,-Cd];
B = [0;1];
C = [1,0];
D = 0;

% Run the simulink simulation 'sim1'
sim sim1

% Plot the results
plot(t,x)
```



Another approach: Example 2:







Loops (Flow Control)

MATLAB has several flow control commands:

- **if**, **else**, and **elseif**
- **switch** and **case**
- **for**
- **while**

- **continue**
- **break**
- **return**

Flow Control: if … else

if : conditionally executes statements

```
if relation
        statements 1
else
        statements 2
end
```

Example:

```
a=5; b=7;
if a>b
   disp('a greater than b');
else
   disp('b greater than a');
end
```

```
if expression1
                statements1

elseif  expression2
                statements2
else
                statements3
end
```

Flow Control: for

The for loop executes a group of statements a number of times.

```
for  variable = expression

          statements
end
```

Example:

```
for n = 1:5
   r(n) =n^2;
end
r
```

expression:

first value: last value

first value: step: last value

Change of increment:

```
for n = 1:2:10; statement; end;
for n=10:-1:1; statement; end;
```

Flow Control: while

The while loop executes a group of statements until a logical test is false.

```
while expression
      statements
end
```

Example:

```
a=1;
while a>0.1
      a=rand
end
```

Other commands:

rand

Flow Control: switch, case

Ex: Find the structure of the command.

```
>> help switch
>> doc switch
```

My Example:

```
clear
a=6; b=2;
method=input(' method=');
switch method
    case 1
        c=a+b;
    case 2
        c=a*b;
    case 3
        c=a/b;
    otherwise
        disp('no valid method')
end
```

files: exSwitch.m

exSwitch2.m

Other commands: **input**, **disp**

Loops: Exit commands

`break`

Lets you exit early from a for loop or while loop.

In nested loops, break exits from the innermost loop only.

`return`

Terminates the current sequence of commands.

Returns control to the invoking function or to the keyboard.

`Ctrl + C`
Emergency exit

We talked about:

- ✓ **Relational operators (>, <. <=, >=…)**
- ✓ **Loops (Flow Control)**
  - ✓ **`if`, `else`,** and **`elseif`**
  - ✓ **`switch`** and **`case`**
  - ✓ **`for`**
  - ✓ **`while`**
  - ✓ **`continue`**
  - ✓ **`break`**
  - ✓ **`return`**