Introduction to Scientific and Engineering Computing, BIL108E

Karaman

INTRODUCTION TO SCIENTIFIC & ENGINEERING COMPUTING BIL 108E, CRN24023

Dr. S. Gökhan Karaman

Technical University of Istanbul

March 08, 2010

Tentative Course Schedule, CRN 24023

Week	Date	Topics
1	Feb. 08	Introduction to Scientific and Engineering Computing
2	Feb. 15	Introduction to Program Computing Environment
3	Feb. 22	Variables, Operations and Simple Plot
4	Mar. 01	Algorithms and Logic Operators
5	Mar. 08	Flow Control, Errors and Source of Errors
6	Mar. 15	Functions
6	Mar. 20	Exam 1
7	Mar. 22	Arrays
8	Mar. 29	Solving of Simple Equations
9	Apr. 05	Polynomials Examples
10	Apr. 12	Applications of Curve Fitting
11	Apr. 19	Applications of Interpolation
11	Apr. 24	Exam 2
12	Apr. 26	Applications of Numerical Integration
13	May 03	Symbolic Mathematics
14	May 10	Ordinary Differential Equation (ODE) Solutions with Built-in Functions

LECTURE # 5

LECTURE # 5

1. INLINE FUNCTIONS
2. M-FILE
   - SCRIPT M–FILES
   - FUNCTION M–FILES
3. EXAMPLES
4. RECURSIVE FUNCTIONS
5. STRUCTURES
6. EXAMPLES

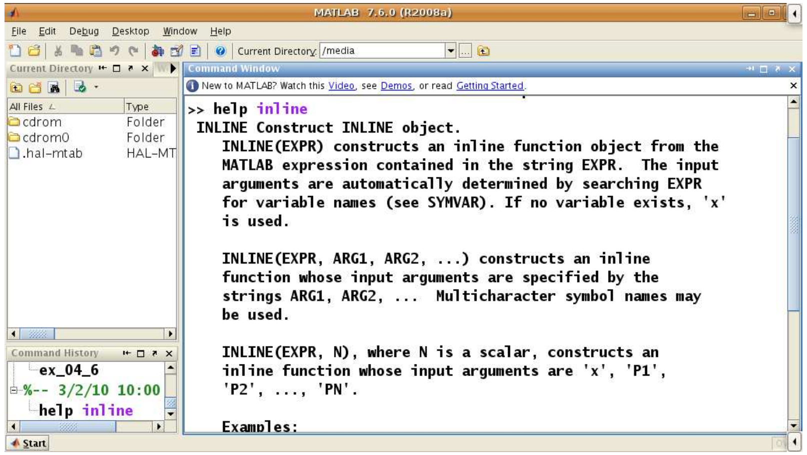
INLINE FUNCTIONS

INLINE FUNCTIONS

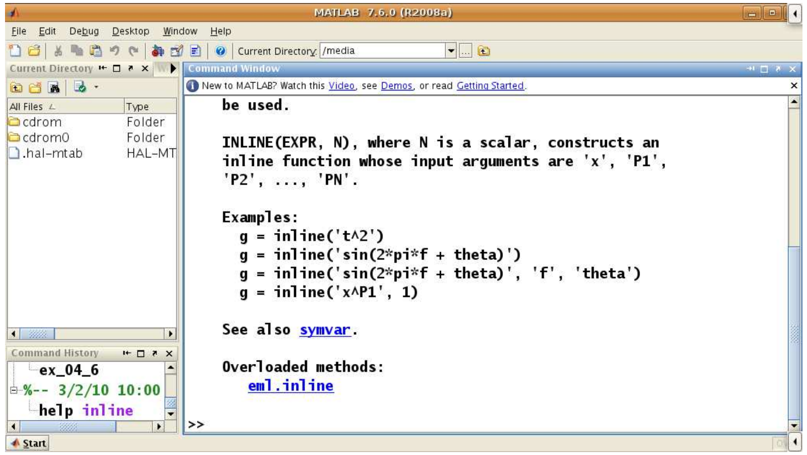
- Short mathematical functions may be written as one=line inline objects.
- Usage:

function_ name = `inline`('function_ definition', 'argument1', 'argument2',...)

INLINE FUNCTIONS



INLINE FUNCTIONS



INLINE FUNCTIONS

EXAMPLE:

1. Convert degrees to radians

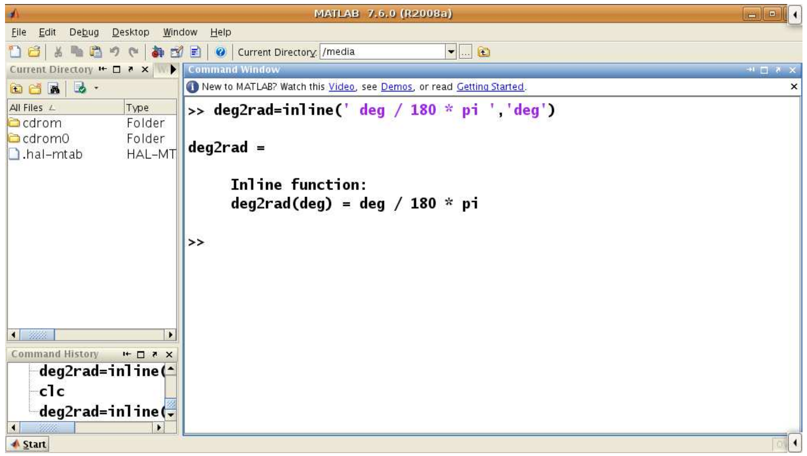
   `deg2rad=inline(' deg / 180 * pi ', 'deg')`

2. Calculate hypothenus

   `hyp = inline('sqrt(a^2 + b^2 )', 'a', 'b')`

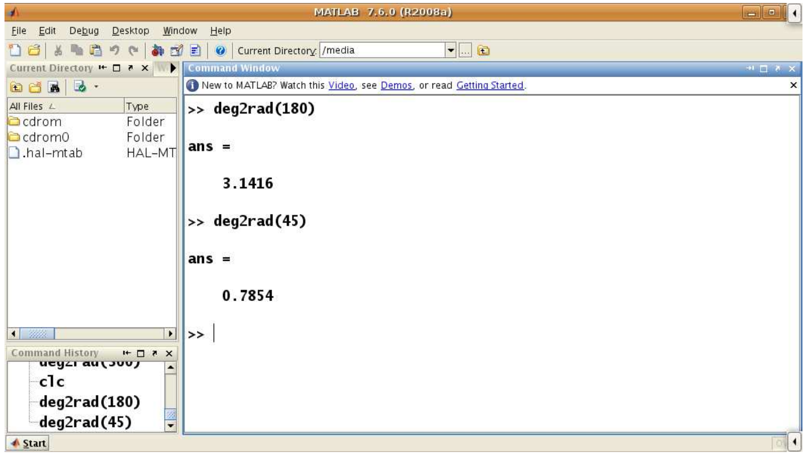
INLINE FUNCTIONS

EXAMPLES;



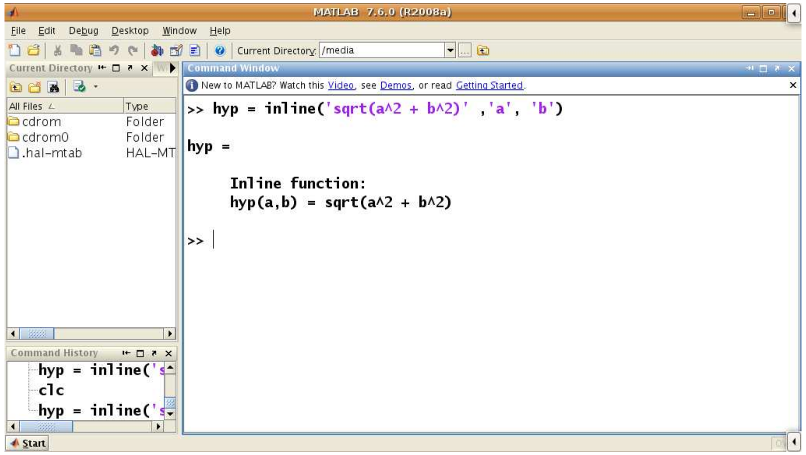
INLINE FUNCTIONS

EXAMPLES;



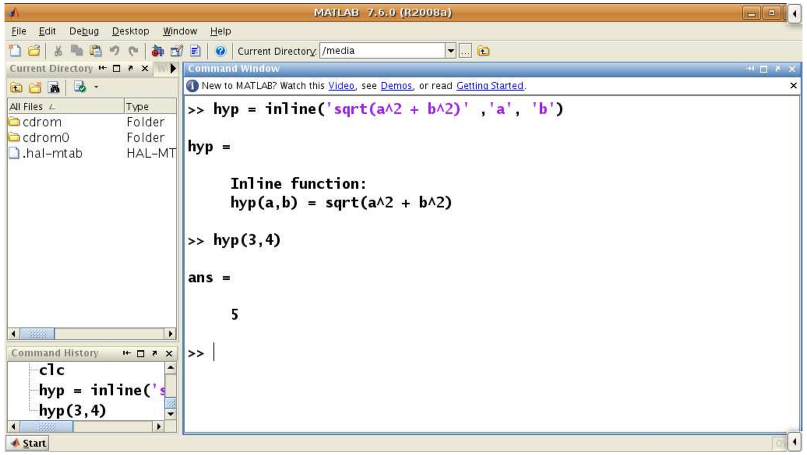
INLINE FUNCTIONS

EXAMPLES;



INLINE FUNCTIONS

EXAMPLES;

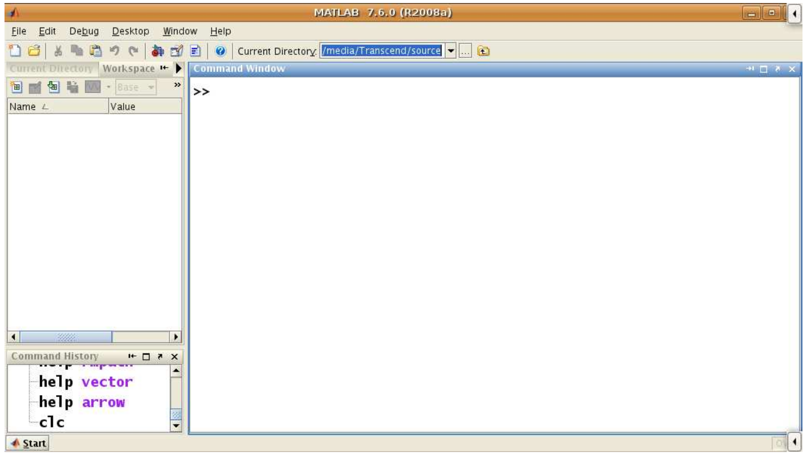


SCRIPT M–FILES

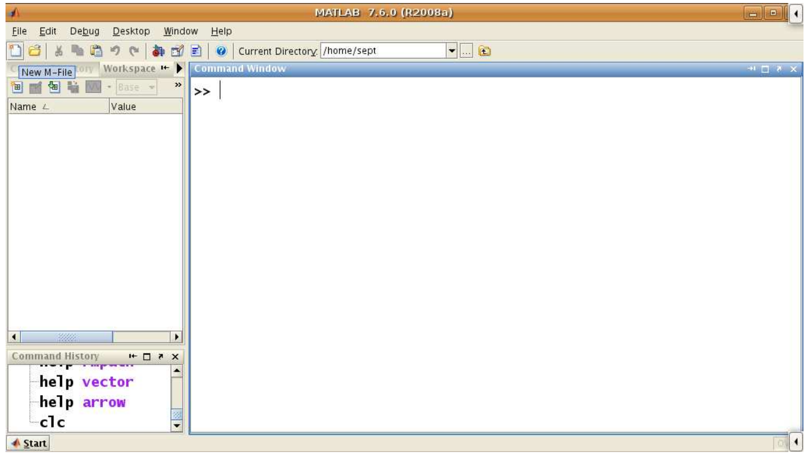
SCRIPT M–FILES

- Programs are contained in **m–files**. **m–files** are plain text files. Not binary files produced by word processors.
- File must have `".m"` extension
- m–file must be in the path Matlab maintains its own internal path
- The path is the list of directories that Matlab will search when looking for an m–file to execute.
- Manually modify the path with the `path`, `addpath`, and `rmpath` built-in functions.

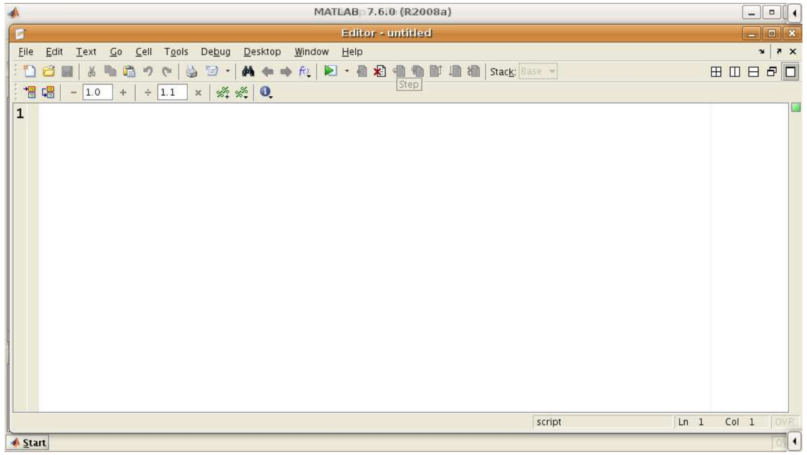
CURRENT DIRECTORY



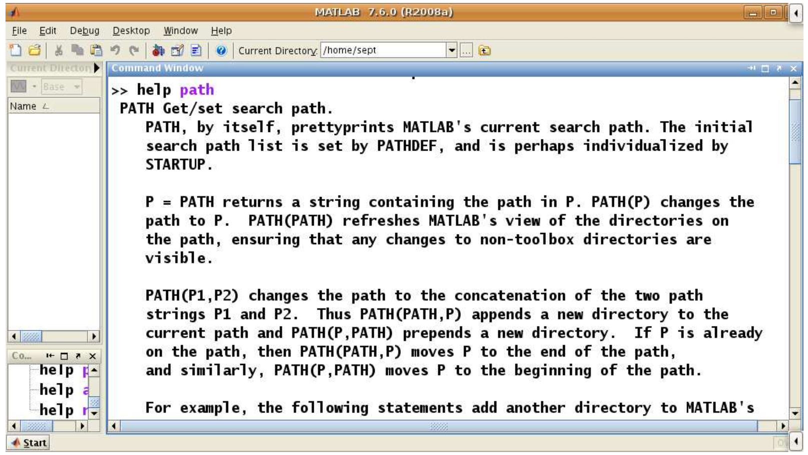
CREATE M–FILE



CREATE M–FILE



path



path

```
visible.

PATH(P1,P2) changes the path to the concatenation of the two path
strings P1 and P2.  Thus PATH(PATH,P) appends a new directory to the
current path and PATH(P,PATH) prepends a new directory.  If P is already
on the path, then PATH(PATH,P) moves P to the end of the path,
and similarly, PATH(P,PATH) moves P to the beginning of the path.

For example, the following statements add another directory to MATLAB's
search path on various operating systems:

  Unix:     path(path,'/home/myfriend/goodstuff')
  Windows:  path(path,'c:\tools\goodstuff')

See also what, cd, dir, addpath, rmpath, genpath, pathtool, savepath, rehas

Reference page in Help browser
   doc path
```


addpath

```
>> help addpath
 ADDPATH Add directory to search path.
    ADDPATH DIRNAME prepends the specified directory to the current
    matlabpath.  Surround the DIRNAME in quotes if the name contains a
    space.  If DIRNAME is a set of multiple directories separated by path
    separators, then each of the specified directories will be added.

    ADDPATH DIR1 DIR2 DIR3 ...  prepends all the specified directories to
    the path.

    ADDPATH ... -END    appends the specified directories.
    ADDPATH ... -BEGIN  prepends the specified directories.
    ADDPATH ... -FROZEN disables directory change detection for directories
                        being added and thereby conserves Windows change
                        notification resources (Windows only).

    Use the functional form of ADDPATH, such as ADDPATH('dir1','dir2',...),
    when the directory specification is stored in a string.
```


addpath

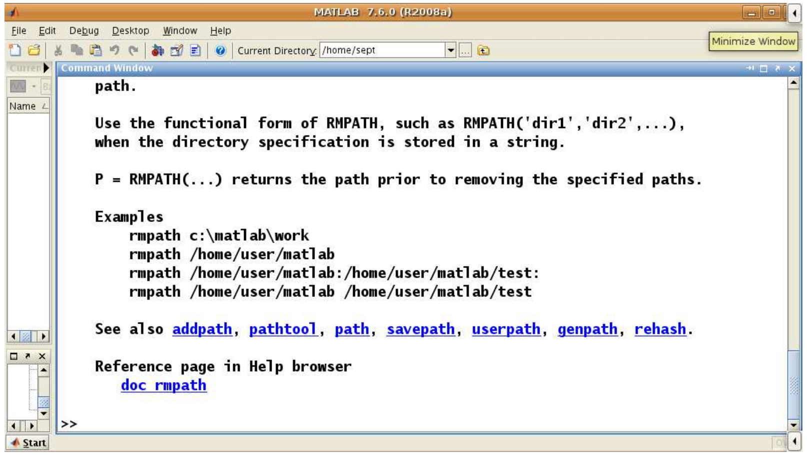
```
                        being added and thereby conserves Windows change
                        notification resources (Windows only).

    Use the functional form of ADDPATH, such as ADDPATH('dir1','dir2',...),
    when the directory specification is stored in a string.

    P = ADDPATH(...) returns the path prior to adding the specified paths.

    Examples
        addpath c:\matlab\work
        addpath /home/user/matlab
        addpath /home/user/matlab:/home/user/matlab/test:
        addpath /home/user/matlab /home/user/matlab/test

    See also rmpath, pathtool, path, savepath, userpath, genpath, rehash.

    Reference page in Help browser
       doc addpath
```


rmpath

```
>> help rmpath
 RMPATH Remove directory from search path.
    RMPATH DIRNAME  removes the specified directory from the current
    matlabpath.  Surround the DIRNAME in quotes if the name contains a
    space.  If DIRNAME is a set of multiple directories separated by path
    separators, then each of the specified directories will be removed.

    RMPATH DIR1 DIR2 DIR3  removes all the specified directories from the
    path.

    Use the functional form of RMPATH, such as RMPATH('dir1','dir2',...),
    when the directory specification is stored in a string.

    P = RMPATH(...) returns the path prior to removing the specified paths.

    Examples
        rmpath c:\matlab\work
        rmpath /home/user/matlab
        rmpath /home/user/matlab:/home/user/matlab/test:
```

SCRIPT M–FILES

rmpath



Introduction to Scientific and Engineering Computing, BIL108E
Karaman

SCRIPT M–FILES

SCRIPT M–FILES

- Not really programs
- No input / output parameters
- Script variables are part of workspace
- Useful for tasks that never change
- Use a script to run function for specific parameters required by the assignment

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

SCRIPT M–FILES

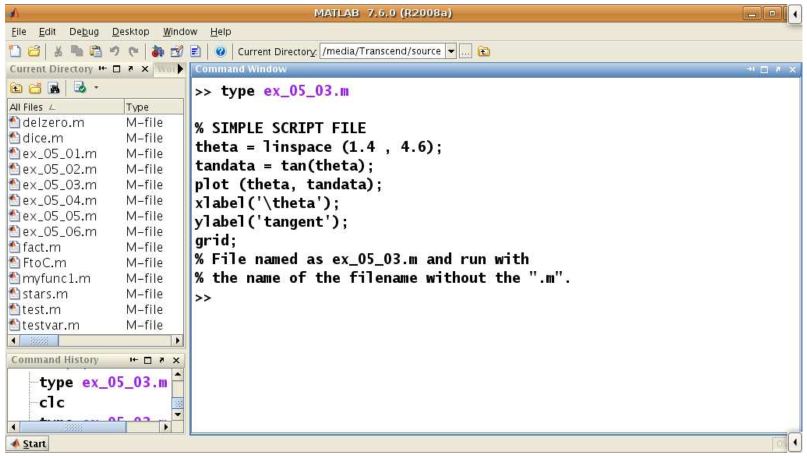
EXAMPLE:

```
% SIMPLE SCRIPT FILE
theta = linspace (1.4 , 4.6);
tandata = tan(theta);
plot (theta, tandata);
xlabel('\theta');
ylabel('tangent');
grid;
% File named as ex_05_03.m and run with
% the name of the filename without the ".m".
```

Introduction to Scientific and Engineering Computing, BIL108E
Karaman

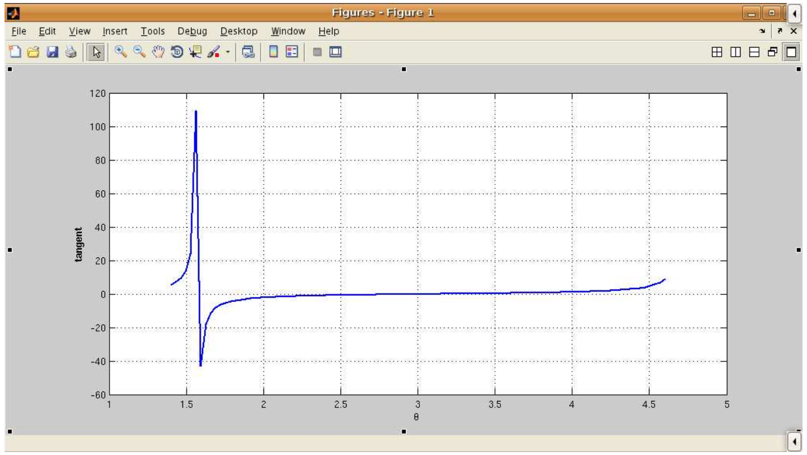
SCRIPT M–FILES

EXAMPLE:



SCRIPT M–FILES

EXAMPLE:



SCRIPT M–FILES

SCRIPT M–FILES

- All variables created in a script file are added to the workplace.
- This may have undesirable effects because variables already existing in the workspace may be overwritten

FUNCTION M–FILES

FUNCTION M–FILES

- Matlab has many built-in functions.
- Use `type` *functionname* to verify the functions.
- Function m–files differ from a script file in that it communicates with the MATLAB workspace only through specially designated **input** and **output** arguments.

FUNCTION M–FILES

FUNCTION M–FILES

- Functions use input and output parameters to communicate with other functions and the command window
- Functions use local variables that exist only while the function is executing. Local variables are distinct from variables of the same name in the workspace or in other functions.
- Input parameters allow the same calculation procedure (same algorithm) to be applied to different data. Thus, function m-files are reusable.
- Functions can call other functions.

FUNCTION M–FILES

BASIC RULES

General form of a function *filename.m* file

`function` [outarg1, outarg2, . . .] =
name(inarg1, inarg2, . . .)

% Comments to be displayed with `help name`

. . .

outarg1;

. . .

outarg2;

FUNCTION M–FILES

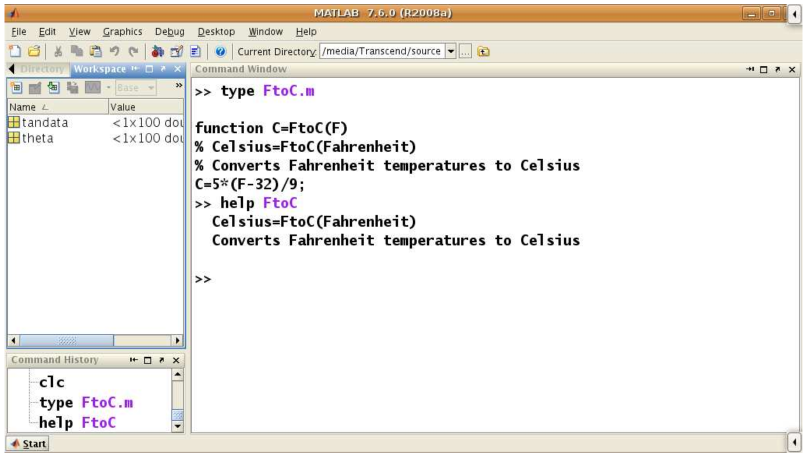
EXAMPLE:

1. Write a function called FtoC (`FtoC.m`) to convert Fahrenheit temperatures into Celsius.

```
function C=FtoC(F)
% Celsius=FtoC(Fahrenheit)
% Converts Fahrenheit temperatures to Celsius
C=5*(F-32)/9;
```

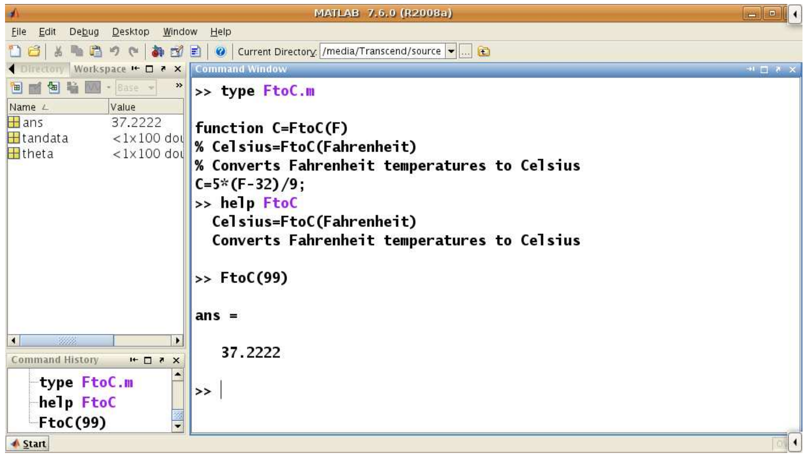
FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

SUMMARY OF INPUT AND OUTPUT ARGUMENTS

- Values are communicated through input arguments and output arguments.
- Variables defined inside a function are local to that function. Local variables are invisible to other functions and to the command environment.
- The number of return variables should match the number of output variables provided by the function.

FUNCTION M–FILES

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

SCOPE

LOCAL VARIABLES

- Any variable defined inside a function is inaccessible outside it.
- Such variables are referred to as local. They exist only inside the function, which has its own workspace separate from the base workspace of variables defined in the Command Window.

FUNCTION M–FILES

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

SCOPE

GLOBAL VARIABLES

- Variables defined in the base workspace are not normally accessible inside functions.
- Their scope is restricted to the workspace itself unless they have been declared global: `global VARIABLENAME`
- If several functions, and possibly the base workspace, declare particular variables as global, then they all share single copies of them.

FUNCTION M–FILES

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

SCOPE

GLOBAL VARIABLES cont'd.

- Matlab recommends that global variables be typed in capital letters to remind you that they are global.
- The function `isglobal(VARNAME)` returns 1 if A is global, and 0 otherwise.
- The command `who global` gives a list of global variables.
- The command `clear global` makes all variables nonglobal. Example: `clear VARNAME` makes `VARNAME` nonglobal.

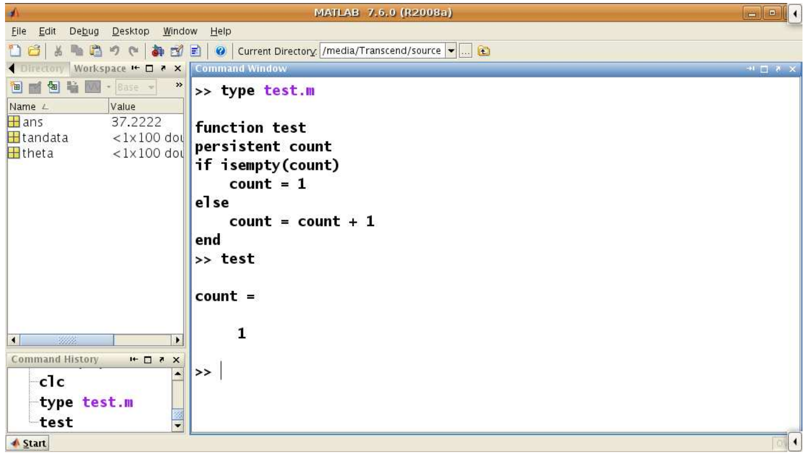
PERSISTENT VARIABLES

Persistent variables remain in existence between function calls.

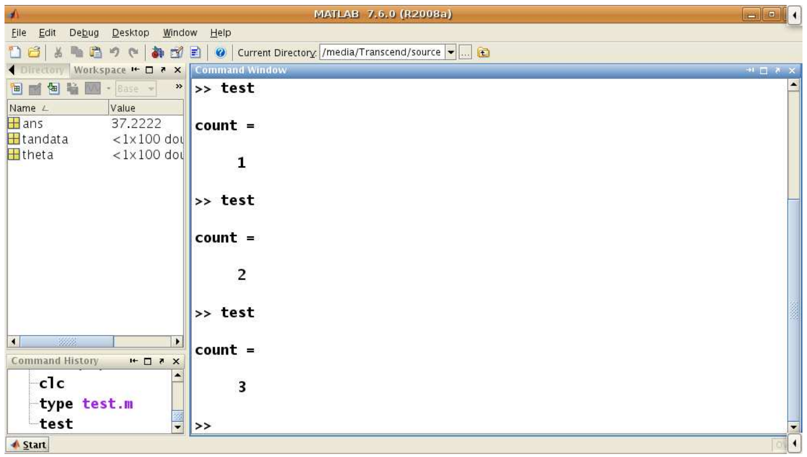
Example:

```
function test
persistent count
if isempty(count)
count = 1
else
count = count + 1
end
```


EXAMPLE:



EXAMPLE:



FUNCTIONS WITHOUT RETURN VALUE

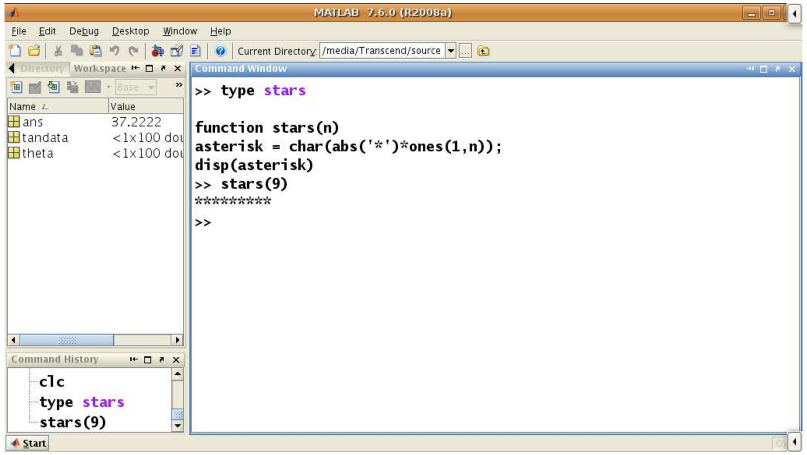
Omit the equal sign and output arguments in the function definition line.

EXAMPLE:

```
function stars(n)
asterisk = char(abs('*')*ones(1,n));
disp(asterisk)
```

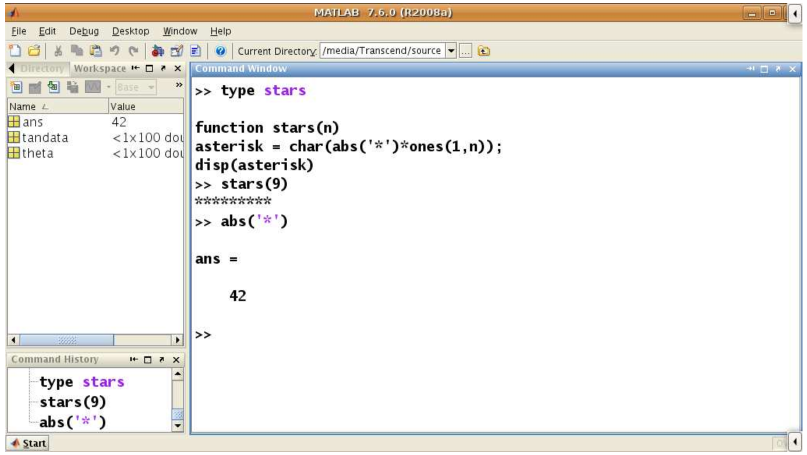
FUNCTION M–FILES

EXAMPLE:



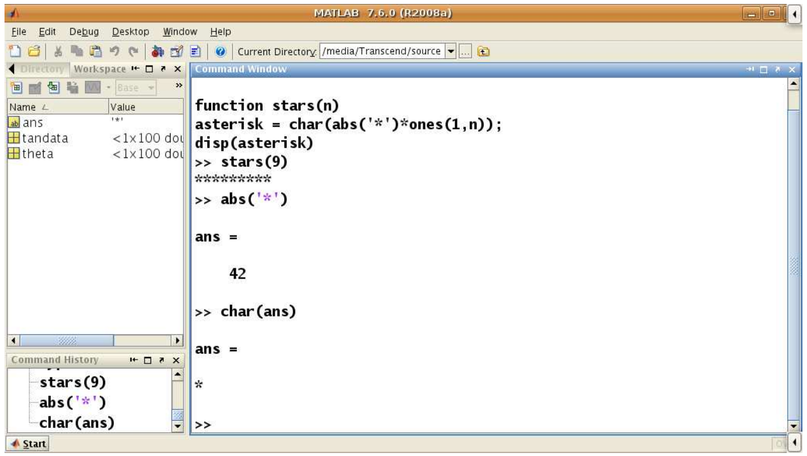
FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

VECTOR ARGUMENTS

- Input and output arguments can be defined as vectors.
- A vector is initialized each time the function is called.

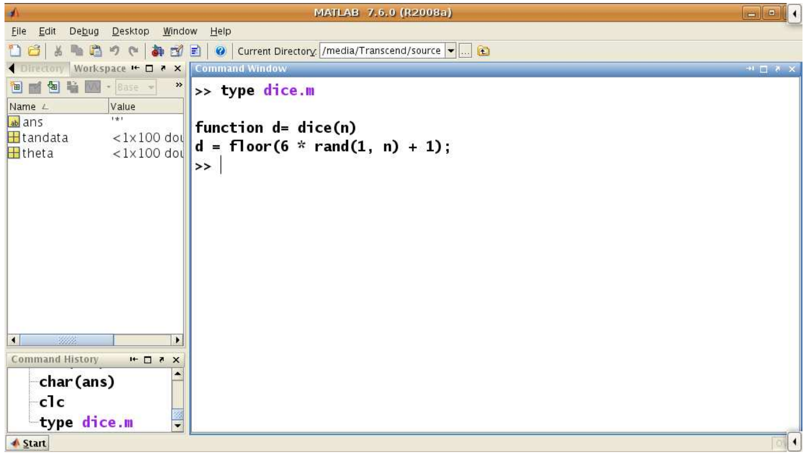
EXAMPLE:

`dice` function generates a vector of `n` random rolls of a die.

```
function d= dice(n)
d = floor(6 * rand(1, n) + 1);
```

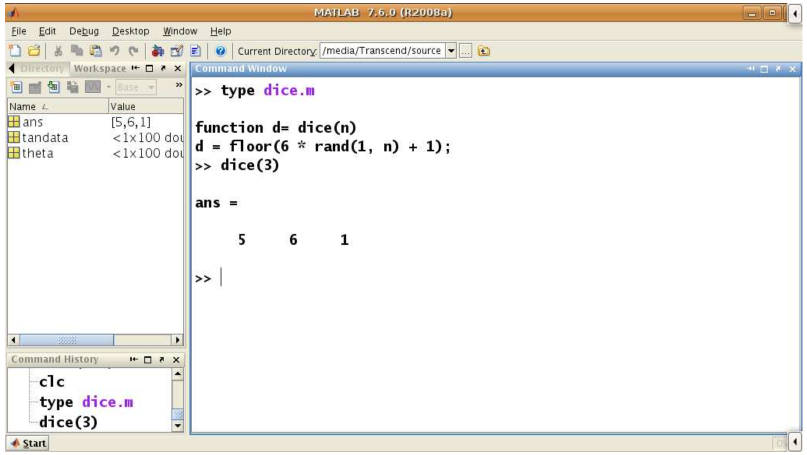
FUNCTION M–FILES

EXAMPLE:



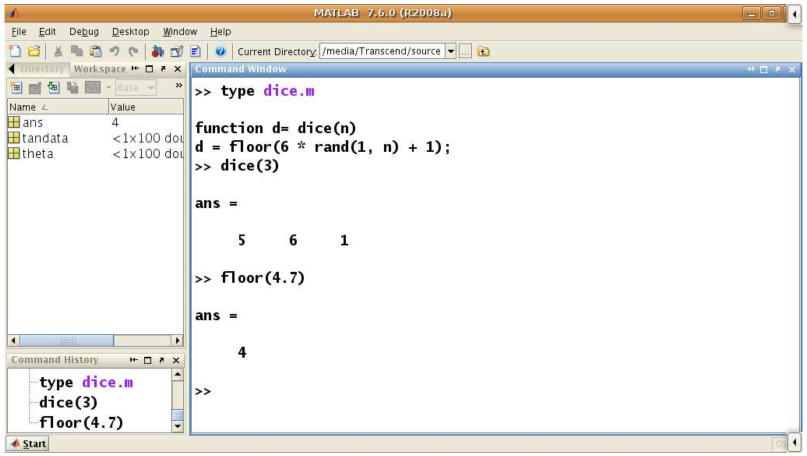
FUNCTION M–FILES

EXAMPLE:



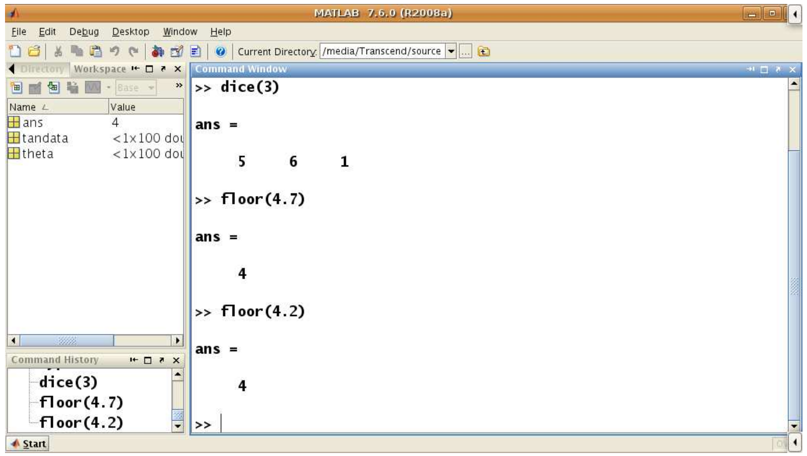
FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

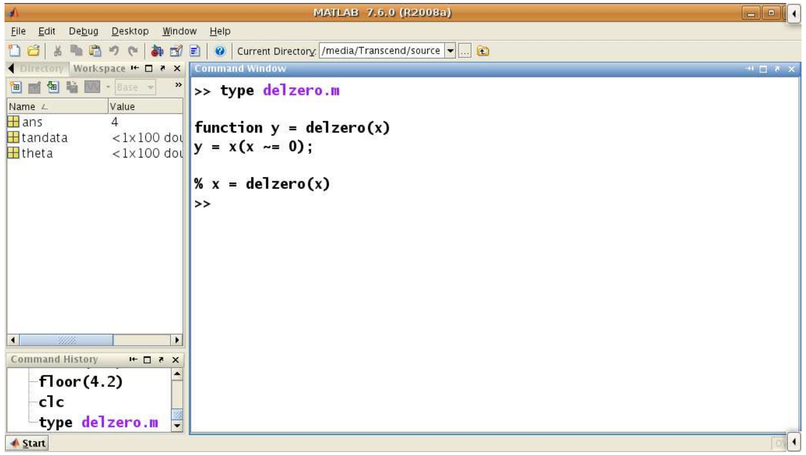
PASSING AN ARGUMENT

- Passing an argument by value:
  An input argument is passed by value only if a function modifies it.
- Passing an argument by reference

```
function y = delzero(x)
y = x(x ~= 0);

% x = delzero(x)
```

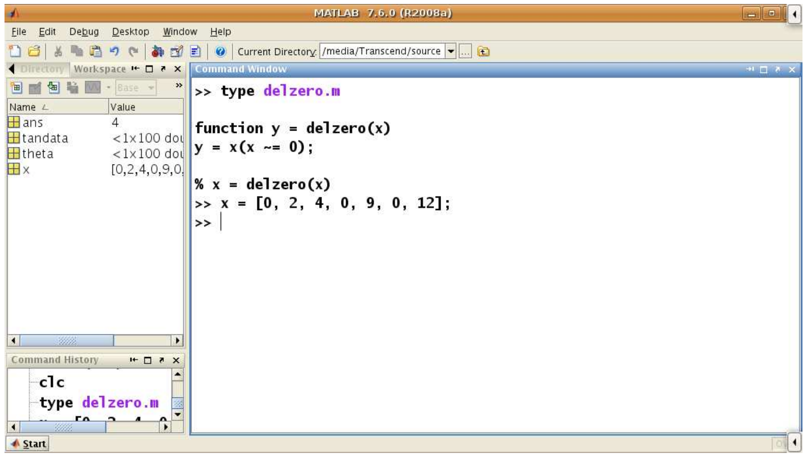

FUNCTION M–FILES

EXAMPLE:



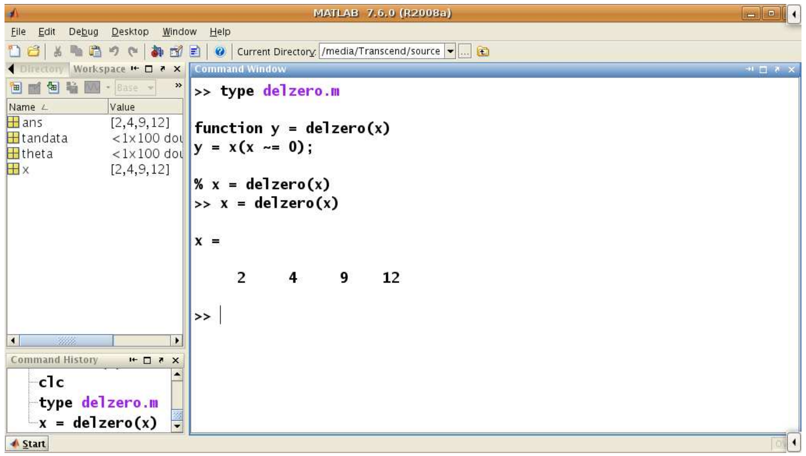
FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

NUMBER OF ARGUMENTS OF A FUNCTION

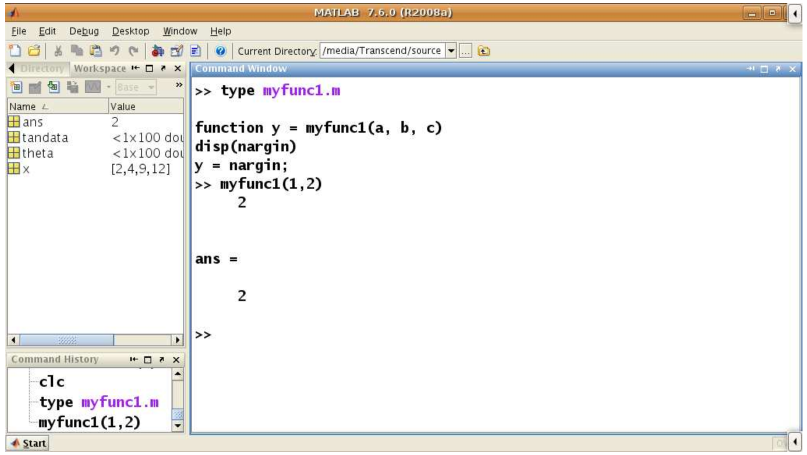
Use `nargin` and `nargout` functions to display the number of input and output arguments.

EXAMPLE:

```
function y = myfunc1(a, b, c)
disp(nargin)
y = nargin;
```

FUNCTION M–FILES

EXAMPLE:



FUNCTION M–FILES

SUBFUNCTIONS

- An M-file may contain the code for more than one function.
- The first one in the file is the **primary function** and is invoked with the M-file name.
- Additional functions are called **subfunctions** and are visible only to the primary function and to other subfunctions.
- Each subfunction begins with its own function definition line.
- Subfunctions follow each other in any order after the primary function.

RECURSIVE FUNCTIONS

RECURSIVE FUNCTIONS

MATLAB allows functions to call themselves in a process called recursion.

Recursive function can cost too much time and memory.

Example:

Factorial function can be written in a recursive ".m" file.

$$n! = n \times (n-1)!$$

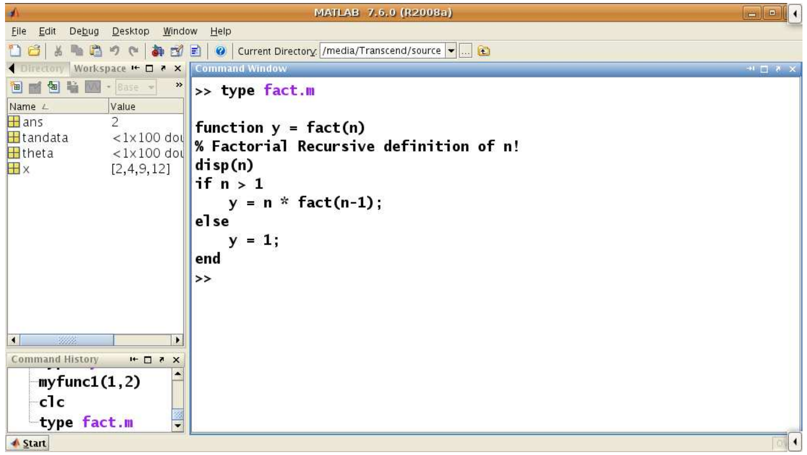
RECURSIVE FUNCTIONS

EXAMPLE:

```
function y = fact(n)
% Factorial Recursive definition of n!
disp(n)
if n > 1
    y = n * fact(n-1);
else
    y = 1;
end
```

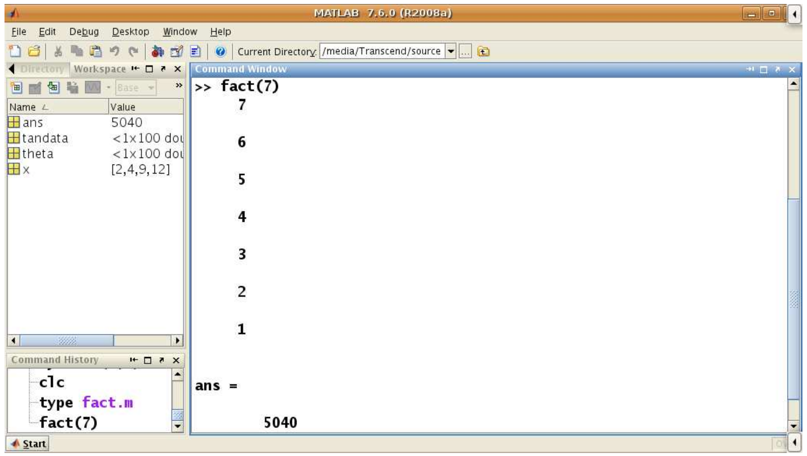
RECURSIVE FUNCTIONS

EXAMPLE:



RECURSIVE FUNCTIONS

EXAMPLE:



FUNCTION HANDLES

FUNCTION HANDLES
Function handles points the defined function.

- A handle for a function is created with `@`. A function may be represented by its handle. In particular, the handle may be passed as an argument to another function.
- `feval` evaluates a function whose handle is passed to it as an argument.

FUNCTION HANDLES

EXAMPLE:

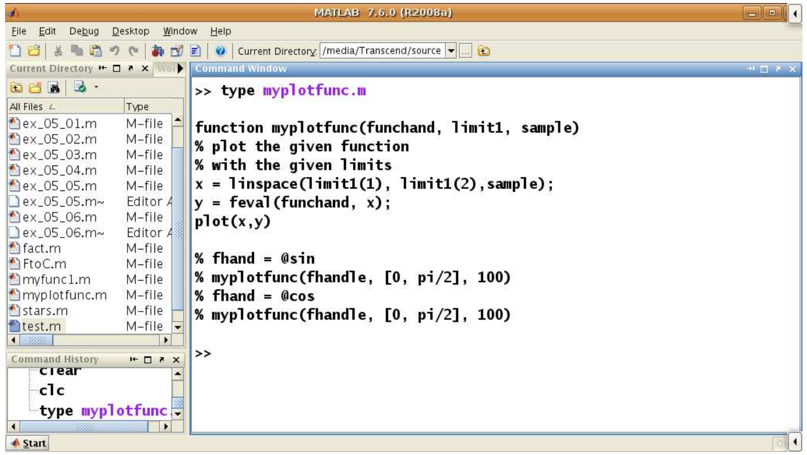
```
function myplotfunc(funchand, limit1, sample)
% plot the given function
% with the given limits
x = linspace(limit1(1), limit1(2),sample);
y = feval(funchand, x);
plot(x,y)

% fhand = @sin
% myplotfunc(fhandle, [0, pi/2], 100)
% fhand = @cos
% myplotfunc(fhandle, [0, pi/2], 100)
```

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

FUNCTION HANDLES

EXAMPLE:

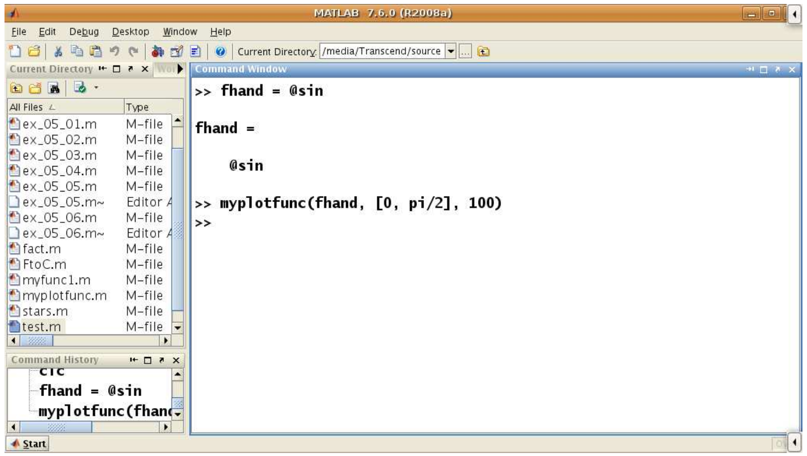


Introduction to Scientific and Engineering Computing, BIL108E

Karaman

FUNCTION HANDLES

EXAMPLE:

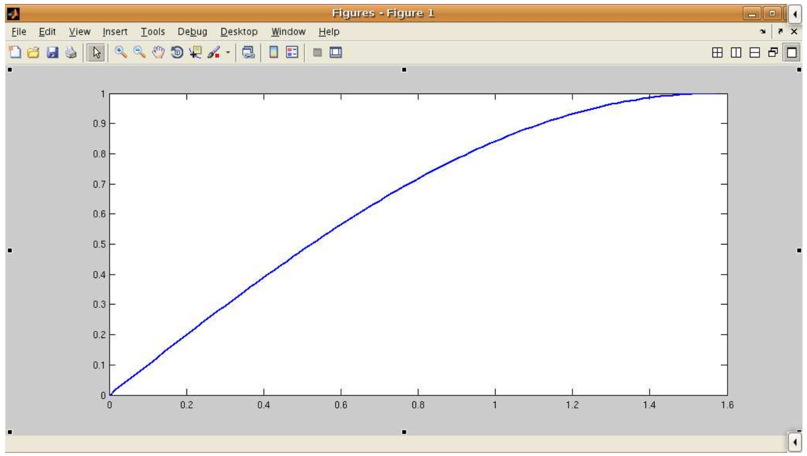


Introduction to Scientific and Engineering Computing, BIL108E

Karaman

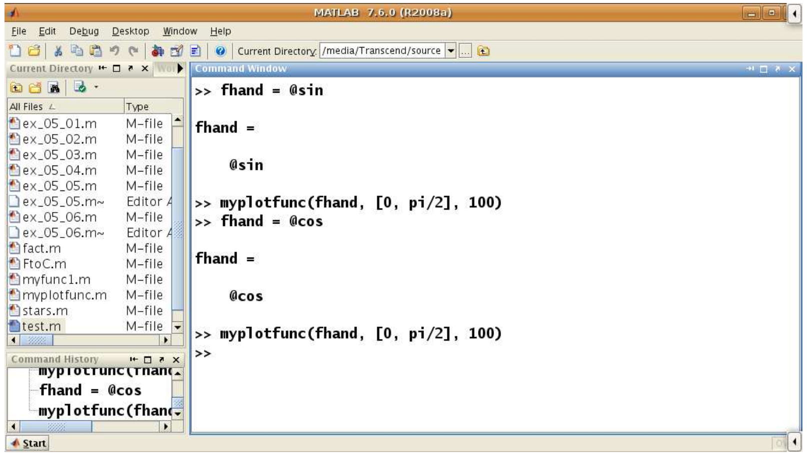
FUNCTION HANDLES

EXAMPLE:



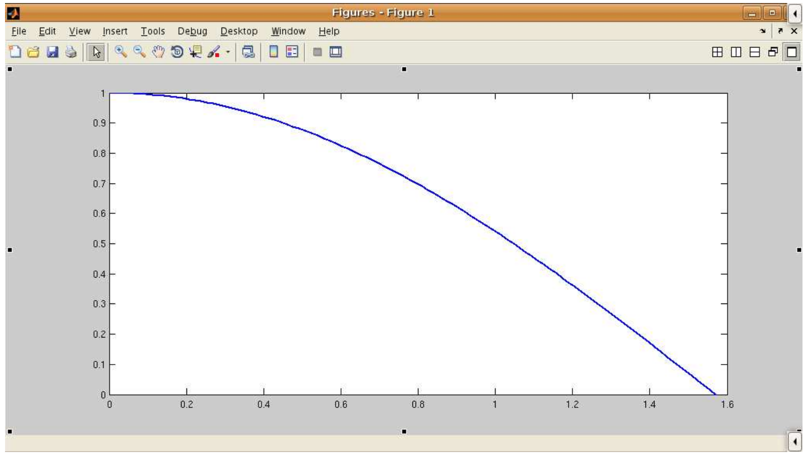
FUNCTION HANDLES

EXAMPLE:



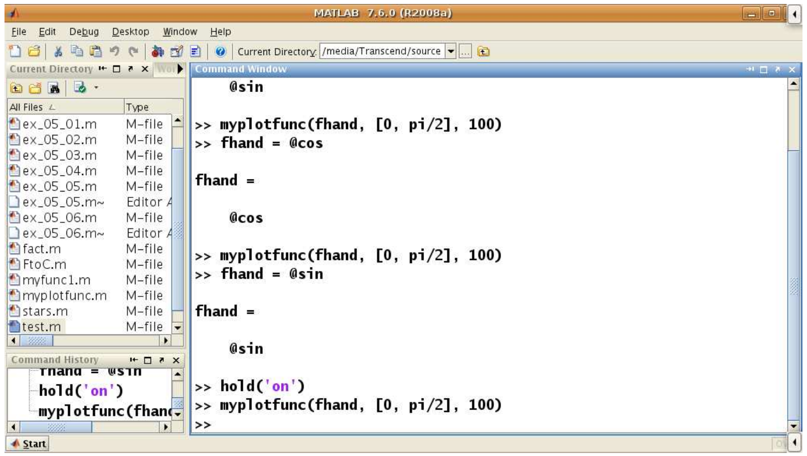
FUNCTION HANDLES

EXAMPLE:



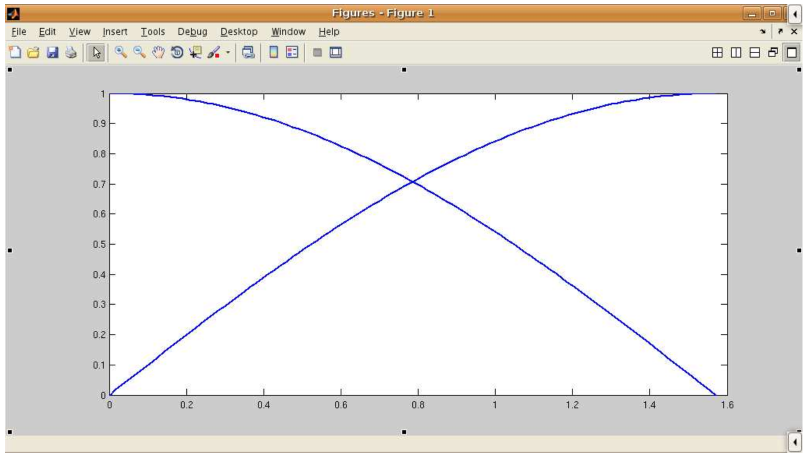
FUNCTION HANDLES

EXAMPLE:



FUNCTION HANDLES

EXAMPLE:



FUNCTION M–FILES

EXAMPLE:

The function $sin(x)$ can be written as a Taylor series by

$$sin(x) = \sum_{k=1}^{\infty} \frac{(-1)^k \, x^{2k+1}}{(2k+1)!}$$

Write a user-defined function file that calculates $sin(x)$ by using Taylor's series. For the function name and arguments use $y = Tsin(x, n)$. The input arguments are the angle x in degrees, and n the number of terms in the series. Use the function to calculate $sin(150^\circ)$ using 3 and 7 terms.

STRUCTURES

STRUCTURES

Arrays can store variables that may be all numeric or character. With structure different data types can be stored as one variable within a structure.

EXAMPLE:

Create a structure called student with one field for a student's name, a second for his/her student ID number, and a third for all her marks to date.

```
student.name = 'Can Ozgur';
student.id = 'N010080090';
student.marks=[80, 60, 40];
% student
% student.marks(2)
```

STRUCTURES

STRUCTURES

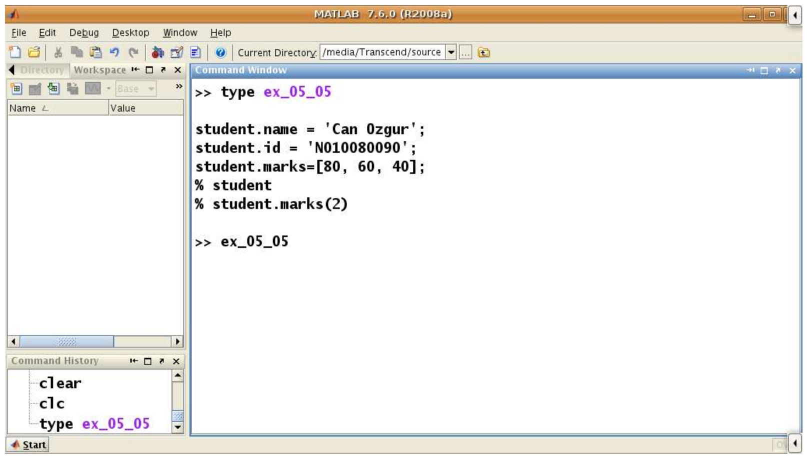
Use subscripts to add more elements

EXAMPLE:

```
student(2).name = 'Ergun Yilmaz';
student(2).id = 'N010080091';
student(2).marks=[70, 30, 90];
% student
% student(2).marks(2)
```

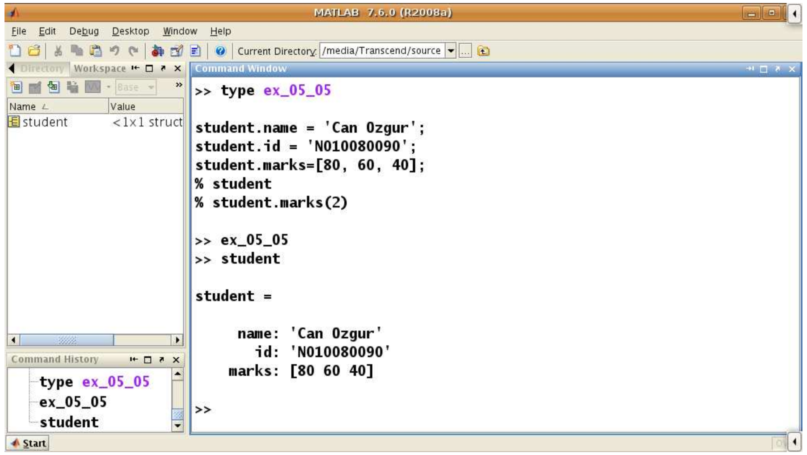
STRUCTURES

EXAMPLE:



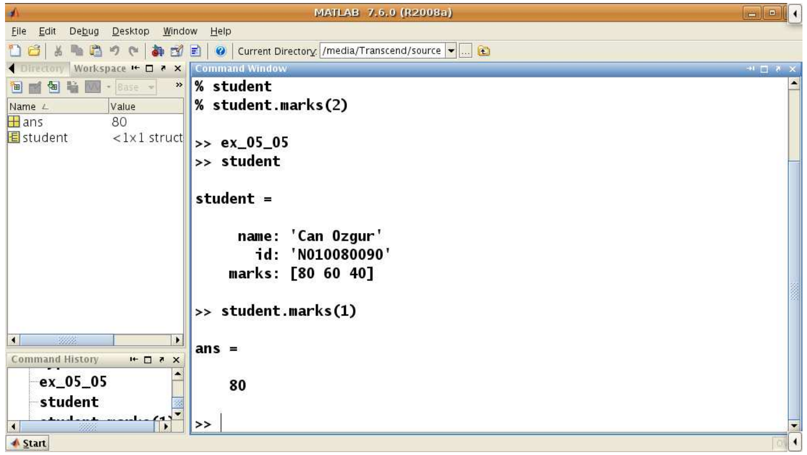
STRUCTURES

EXAMPLE:



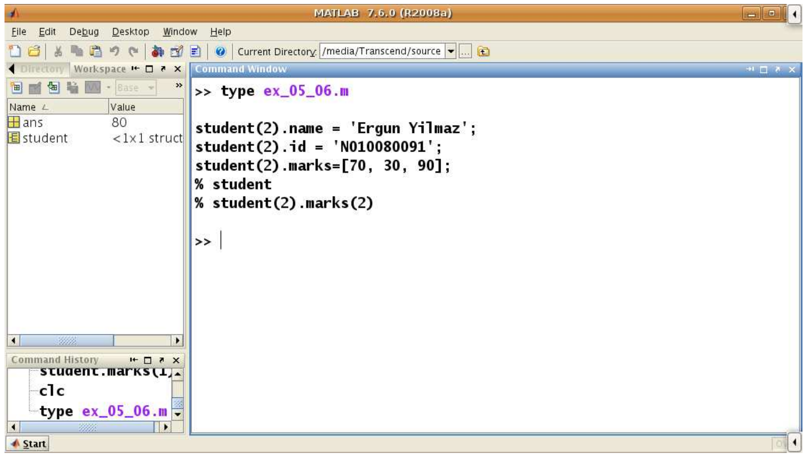
STRUCTURES

EXAMPLE:



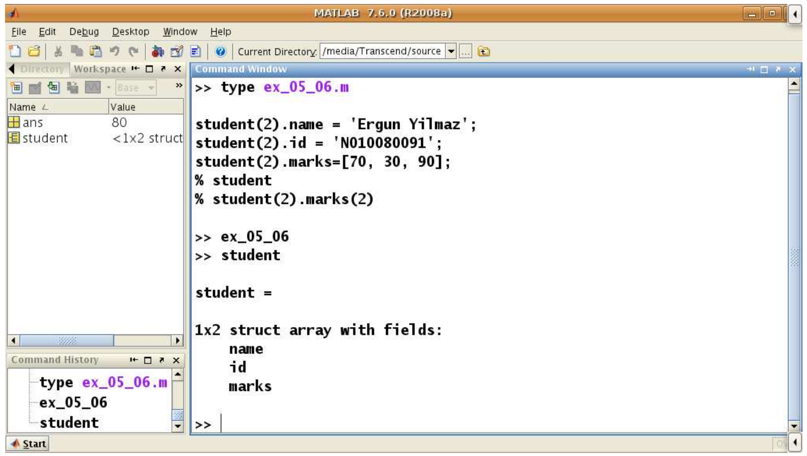
STRUCTURES

EXAMPLE:



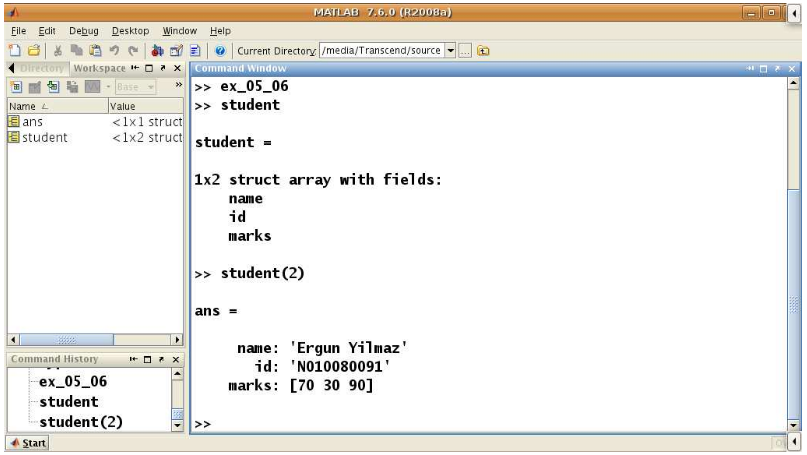
STRUCTURES

EXAMPLE:



STRUCTURES

EXAMPLE:

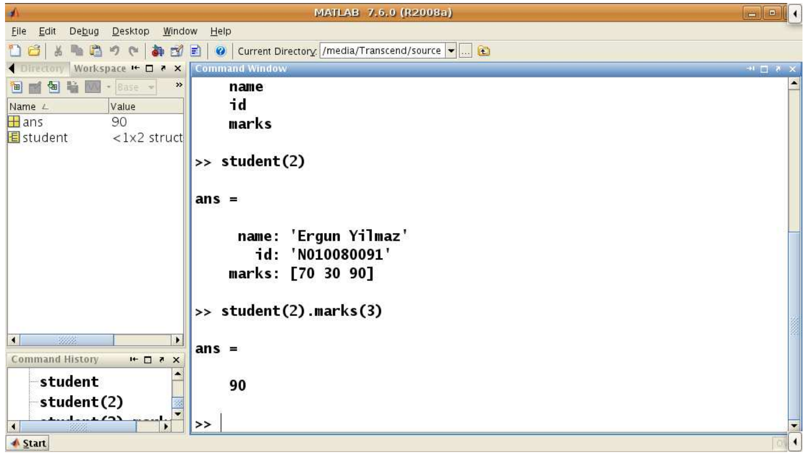


STRUCTURES

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

EXAMPLE:



M–FILE DEBUGGING

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

M–FILE DEBUGGING

The Editor / Debugger enables you to get inside a function, while it is running.

References

Introduction to Scientific and Engineering Computing, BIL108E

Karaman

References for Week 5

1. Brian Hahn, Daniel T.Valentine, Essential Matlab for Engineers and Scientists, Elsevier, 2010.