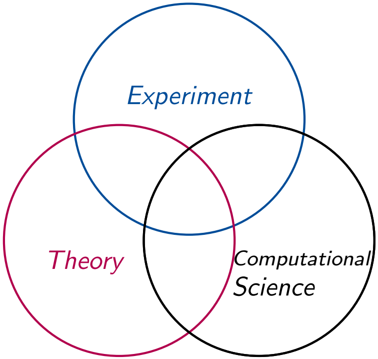
Scientific Computing Lecture Series
Introduction to MATLAB Programming

Hamdullah Yücel*

*Scientific Computing, Institute of Applied Mathematics

Lecture I
Basic Commands, Arrays and Matrices, Functions

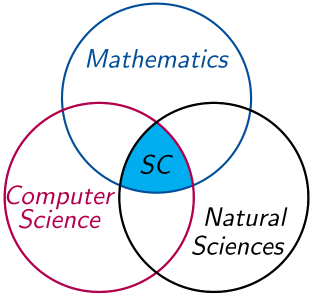
Computational Science



Computational Science now constitutes what many call the **third pillar** of the scientific enterprise, a peer alongside theory and physical experimentation.

Report to the President:"Computational Science : Ensuring America's Competitiveness", June 2005.

Scientific Computing



Scientific Computing

= Computational Science

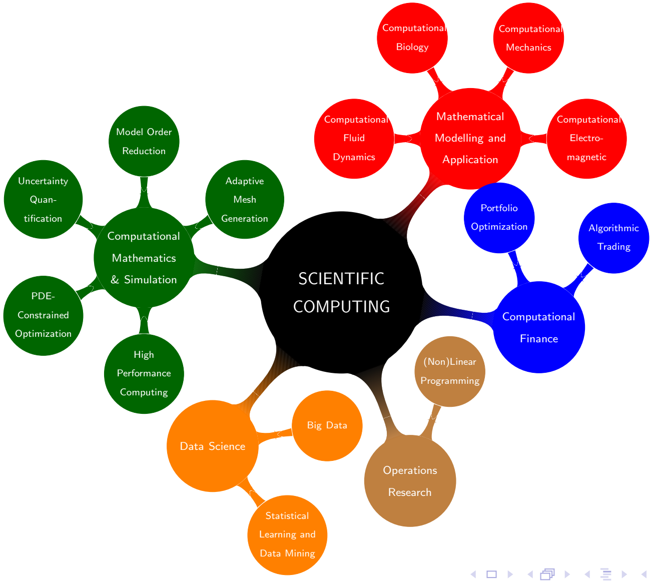
= Computational Science and Engineering

= Scientific Computation

= Computational Mathematics

[0] Quoted by https://en.wikipedia.org/wiki/Computational_science

Scientific Computing Program


SCIENTIFIC COMPUTING
Mathematical Modelling and Application
Computational Biology
Computational Mechanics
Computational Fluid Dynamics
Computational Electro-magnetic
Computational Mathematics & Simulation
Model Order Reduction
Uncertainty Quan-tification
Adaptive Mesh Generation
PDE-Constrained Optimization
High Performance Computing
Computational Finance
Portfolio Optimization
Algorithmic Trading
Operations Research
(Non)Linear Programming
Data Science
Big Data
Statistical Learning and Data Mining

For More Information

- http://iam.metu.edu.tr/scientific-computing
- https://www.facebook.com/SCiamMETU/

Lecture Information

MATLAB Lecture Series is organized by members of **Scientific Computing Program** of IAM:

- **October 14**: Hamdullah Yücel
  - Basic Commands, Arrays and Matrices, Functions,
- **October 15**: M. Alp Üreten
  - Control Loops, Advanced Data Structures, Graphics and Visualizations
- **October 21**: Mustafa Kütük
  - Introduction to Deep Learning for Applied Mathematicians with MATLAB

Lecture I–Outline

What is MATLAB ?

- Matlab is a high–level language and interactive environment that enables you to perform computationally intensive tasks. It was originally designed for solving linear algebra type problems using matrices. It's name is derived from MATrix LABoratory.

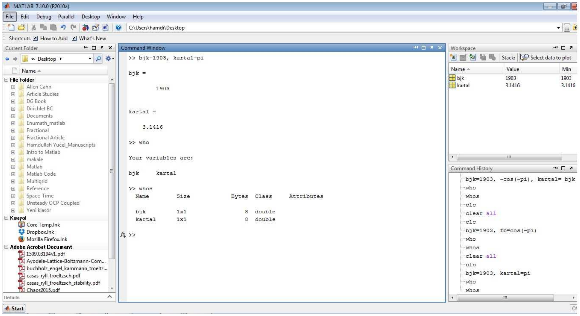
MATLAB System

- Desktop Tools and Development Environment
  - Includes the MATLAB desktop and Command Window, an editor and debugger, a code analyzer, browsers for viewing help, the workspace, files, and other tools.
- Mathematical Function Library
  - Vast collection of computational algorithms ranging from elementary functions, like sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.
- Language
  - The MATLAB language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features.

Continue...

- Graphics
  - MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as editing and printing these graphs. It also includes functions that allow you to customize the appearance of graphics as well as build complete graphical user interfaces on your MATLAB applications.
- External Interfaces
  - The external interfaces library allows you to write C and Fortran programs that interact with MATLAB.

MATLAB Interface



```
>> bjk=1903, kartal=pi

bjk =

        1903


kartal =

    3.1416

>> who

Your variables are:

bjk     kartal

>> whos
  Name        Size            Bytes  Class     Attributes

  bjk         1x1                 8  double
  kartal      1x1                 8  double

>>
```

Name	Value	Min
bjk	1903	1903
kartal	3.1416	3.1416

```
bjk=1903, -cos(-pi), kartal= bjk
who
whos
clc
clear all
clc
bjk=1903, fb=cos(-pi)
who
whos
clear all
clc
bjk=1903, kartal=pi
who
whos
```

- Command Window: Here you can give MATLAB commands typed at the prompt, >>.
- Current Directory: Directory where MATLAB looks for files.
- Workspace: Shows what variable names are currently defined and some info about their contents.
- Command History: History of your commands.

Helps/Docs

- MATLAB is huge! - there is no way to remember everything you will need to know.
  - help command - shows in the Commmand Window all the ways in which you can use the command.
  - doc command - brings up more extensive help in a separate window.
  - lookfor command- searches for the keyword.

```
>> help sin
   SIN    Sine of argument in radians.
   SIN(X) is the sine of the elements of X.
   See also asin, sind.
   Overloaded methods:
                  codistributed/sin
   Reference page in Help browser
   doc sin
```

Basic Commands

- MATLAB records in the workspace and command history everything you write in the command window, so:
  - clear variable
    - deletes variable from memory (and workspace)
  - clear all
    - deletes all variables from memory (and workspace)
  - clc
    - cleans command window
  - save
    - save variables to a file (.mat format)
  - load
    - load variable bindings into the environment (look at workspace, the variables a is back)
- MATLAB's command window works like a Linux terminal
  - cd, mkdir, rmdir, ls, . . .

Basic Commands

- Some commands used to interact with MATLAB
  - what
    - returns the MATLAB files (.m , .mat) in the current directory
  - who
    - returns the variables in your workspace
  - whos
    - returns the variables in the workspace with additional info (size, dimensions)
  - Try typing why in the command window. You will see that MATLAB is also a Philosopher!

Variables

- MATLAB is a weakly typed language
  - No need to initialize variables!
- Just assign some value to a variable name, and MATLAB will automagically understand its type
  - $x = 3$ double
  - $x =$ 'hello' char
- MATLAB supports various types, the most often used are
  - 64-bit double (default)
  - 16-bit char
- Most variables you will deal with will be vectors or matrices of doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8-bit integers, etc.

Variables

- Naming Conventions
  - Have not to be previously declared
  - Variable names can contain up to 63 characters
  - To create a variable, simply assign a value to a name

```
>>  var1 = 1903;
>>  myStrings = 'merhaba';
```

  - Variable names
    - first character must be LETTER
    - after that, any combination of letters, numbers and _
    - allowable: NetCost, Left2Pay, X3, BJK1903
    - not allowable: Net-Cost, 1903BJK, %x, @sign
  - Variable names are case sensitive (var1 is different from Var1)

Variables

- Avoid to use built–in variables such as

ans	Default variable name for results
eps	Smallest incremental number
pi	Value of π
inf	Infinity
NaN	Not a number e.g. 0/0
i and j	represent complex numbers

Scalars

- A variable can be given a value explicitly (shows up in workspace!)

```
>>  a = 1903
```

- Or as a function of explicit values and existing variables

```
>>  c = 2.4*24-4*a
```

- To suppress output, and the line with a semicolon ;

```
>>  h = 22/7;
```

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays
  - matrix of numbers (either double or complex)
  - cell array of objects (more advanced data structure)
- Row vector: comma or space separated values between brackets

```
>> row = [1 4 6 7]
>> row = [1,4,6,7]
```

- Column vector: semicolon separated values between brackets

```
>> column = [1.4;2;pi]
```

- Size of a vector: length

```
>> l = length(column)
```

Special Vector Constructors

- linspace()

```
>> a = linspace(0,10,5)

a =

    0    2.5000    5.0000    7.5000   10.0000
```

- Colon operator (:). The basic syntax is

$$\boxed{\text{inital:stepsize:final}}$$

```
>> m = 3:8, r = 0:0.25:1, s=1:-1

m =

    3     4     5     6     7     8

r =

    0    0.2500    0.5000    0.7500    1.0000

s =

    Empty matrix: 1-by-0
```

- logspace (to initialize logarithmically spaced values)

Matrices

- Make matrices like vectors

```
>> A = [5 7 9; 1 -3 -7];
```

- Concatenation of vectors

```
>> r1 = [2 4];
>> r2 = [3 6];
>> M  = [r1; r2];
```

- Concatenation of vectors and matrices. Dimensions and Type must coincide!

```
>> r1 = [2 4];
>> m1 = [3 6; 8 12];
>> M  = [r1; m1];
```

- Getting size of the matrix

```
>> [r,c] = size(M);     % size in each dimension
>> r  = size(M,1);      c = size(M,2);
>> nd = ndims(M);       % number of dimensions
```

Special Matrices

- zeros(m,n) $m \times n$ matrix of zeros
- ones(m,n) $m \times n$ matrix of ones
- eye(n) $n \times n$ identity matrix
- rand(m,n)
  - $m \times n$ matrix of uniformly distributed random numbers in range [0,1]

```
>> M = rand(2,3)
   M = 0.8147    0.1270    0.6324
       0.9058    0.9134    0.0975
```

- randn(m,n)
  - $m \times n$ matrix of normally distributed random numbers (mean 0, std. dev. 1))

```
>> M = randn(2,3)
   M =   -0.4336    3.5784   -1.3499
          0.3426    2.7694    3.0349
```

Replicating and Concatenating Matrices

- repmat

```
>> X = [1 2;3 4];
>> Y = repmat(X,2,3)
   Y =  1     2     1     2     1     2
        3     4     3     4     3     4
        1     2     1     2     1     2
        3     4     3     4     3     4
```

- vertcat

```
>> v1 = [2 3 4];   v2 = [1 2 3];
>> X = vertcat(v1,v2)
   X =  2     3     4
        1     2     3
```

- horzcat

```
>> v1 = [2; 3; 4];   v2 = [1; 2; 3];
>> X = horzcat(v1,v2)
   X = 2  1
       3  2
       4  3
```

Reshaping Matrices

- Using the : operator

```
>> x = round(10*rand(2,4));
>> y = x(:);  (The elements of x are stacked in a
               column vector, column after column)
```

- reshape()

```
>> x2 = reshape(y,2,4);
>> M = reshape(linspace(11,18,8),[2,2,2])
   M(:,:,1) =
             11    13
             12    14
   M(:,:,2) =
             15    17
             16    18
```

Vector Indexing

- MATLAB indexing starts with 1, not 0
- a(n) returns nth element
- The index argument can be vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.

```
>> x = [4 6 7 -1  0];
>> a = x(2:4); --------> a=[6 7 -1];
>> b = x(1:end-2); ----> b=[4 6  7];
```

Matrix Indexing

- using subscripts (row and column)

```
>> A = [1:3;4:6;7:9];
>> A(1:2,:)
ans =
     1     2     3
     4     5     6
>> A([3 1], [2 3])
ans =
     8     9
     2     3
>> A([1:2],:) = []    % delete row 1 and 2
A =
   7     8     9
```

- using linear indices (as if matrix is vector)

```
>> [A(2), A(4), A(9)]
ans =
     4     2     9
```

Matrix Indexing

- To select rows and columns of a matrix

```
>> c = [1 4; 0 2];
>> d = c(1,:)
   d =
       1     4
```

- To get the min. (or max.) value and its index

```
>> a = [ 1 -1 0 -4, 21];
>> [minVal,minInd] = min(a)
minVal =  -4  minInd =   4
```

- To find any indices of specific values or ranges

```
>> ind = find(a==0);
>> ind = find(a > 0 & a < 4);
```

- To convert between subscripts and indices, use ind2sub and sub2ind

Operations

- Arithmetic operations (+,-,*,/)

```
>> 7/45
>> (2+i)*4/5
```

- Exponentiation (^)

```
>> (3+2*j)^2
```

- Complicated expressions, use parentheses

```
>> ((2+3)*3)^0.5
```

- Multiplication is NOT implicit given parenthesis

```
>> 3(1+0.7)
??? 3(1+0.7)
     |
Error: Unbalanced or unexpected parenthesis or bracket.
```

- MATLAB has an enormous library of built-in-functions

```
>> sqrt(2), log(2), log(10)(0.23), cos(pi), atan(2.5)
>> exp(1903), round(1.4),  floor(3.3), ceil(4.23)
```

Transpose

- The transpose operators turns a column vector into a row vector and vice versa
- The $'$ gives the Hermitian-transpose, i.e., transposes and conjugates all complex numbers
- For vectors of real numbers $.'$ and $'$ give same result

```
>> a = [ 1;5; 3i+2]
>> a'
ans =
   1.0000    5.0000   2.0000 - 3.0000i
>> transpose(a)
ans =
   1.0000    5.0000   2.0000 + 3.0000i
>> a.'
ans =
   1.0000    5.0000   2.0000 + 3.0000i
```

Element-Wise Functions

- All functions that work on scalars also works on vectors

```
>> t = [1, pi, 0];
>> f = exp(t);
>> f = [exp(1) exp(pi) exp(0)];
```

- To do element-wise operations, use the dot: $.*, ./, .\hat{}$. Both dimensions must match (unless one is scalar)

```
>> u=1:2:8, v=u.^2, w=u./v
   u =
      1      3      5      7
   v =
      1      9     25     49
   w =
      1.0000     0.3333     0.2000     0.1429
>> A = [ 5 7 9; 1 -3 -7];  B = [-1 2 5; 9  0  5];
>> A.*B
   ans =
       -5     14     45
        9      0    -35
```

Rational and Logical Operators

- Boolean values: zero is false, nonzero is true
- Some of the logical operators:

Operator	Meaning
$<, <=, >, >=$	less than, less than or equal to, etc.
$==, \sim=$	equal to, not equal to
&	logical AND
$\mid$	logical OR
$\sim$	logical NOT
all	all true
any	any true

Logical Indexing

- Construct a matrix R

```
>> R = rand(5)
   R =
       0.8147    0.0975    0.1576    0.1419    0.6557
       0.9058    0.2785    0.9706    0.4218    0.0357
       0.1270    0.5469    0.9572    0.9157    0.8491
       0.9134    0.9575    0.4854    0.7922    0.9340
       0.6324    0.9649    0.8003    0.9595    0.6787
```

- Test for some logical cases

```
>> R(R<0.15)'
  ans =
        0.1270    0.0975    0.1419    0.0357
>> isequal(R(R<0.15), R(find(R<0.15)))
  ans =
        1
```

Find

- find returns indices of nonzero values. It can simplify code and help avoid loops
- basic syntax: index = find(condition)

```
>> x = rand(1,10)
   x =
      Columns 1 through 5
      0.4505    0.0838    0.2290    0.9133    0.1524
      Columns 6 through 10
      0.8258    0.5383    0.9961    0.0782    0.4427
>> inds = find(x>0.4 & x<0.7)
   inds =
          1     7    10
>> x(inds)
   ans =
         0.4505    0.5383    0.4427
```

Dense Matrices

- Dense matrix is a matrix in which most of its elements are nonzero.
- Any classical approach to create a matrix results a dense matrix in MATLAB.

`[,]`	creates a single row matrix
`[;]`	creates a singe column matrix
`zeros(n)`	returns an $n \times n$ matrix of zeros
`ones(n)`	returns an $n \times n$ matrix of 1s
`diag()`	creates diagonal matrix of given vector

- Create a 1000×1000 matrix A

$$\begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix}$$

```
M = 1000;
A = diag(ones(M-1,1),-1) + diag(-2*ones(M,1),0) + diag(ones(M-1,1),1);
```

- Compute how much storage this dense matrix need

```
s = whos('A');
by = s.bytes;
>> by = 8000000 bytes
```

Sparse Matrices

- A sparse matrix is a matrix which has relatively small number of nonzero elements.
- Triplet Format in MATLAB stores values and their corresponding row and column values.

```
row = [1 2 3 1 5 4 1 5];
col = [1 1 2 3 3 4 5 5];
val = [2 8 9 2 4 5 7 3];

S = sparse(row,col,val);
```

$$S = \begin{bmatrix} 2 & 0 & 2 & 0 & 7 \\ 8 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 4 & 0 & 3 \end{bmatrix}$$

- spalloc() creates an all zero allocation for a sparse matrix.

```
m = 10;        % number of rows
n = 10;        % number of columns
nz = 21;       % number of nonzero entries
S = spalloc(m,n,nz);
```

- spones() generates a matrix of 1s with same sparsity structure as matrix S

```
M = spones(S);
```

- speye() constructs a sparse identity matrix of size $m \times n$

```
I = speye(m,n);
```

- spdiags() extracts or constructs sparse diagonal matrices.
  - Extracts nonzero diagonal entries from matrix S

    ```
    B = spdiags(S);
    ```

  - Extracts diagonals of S specified by d

    ```
    B = spdiags(S,d);
    ```

  - Replaces the diagonals of S specified by d with columns of B

    ```
    S = spdiags(B,d,S);
    ```

  - Create $m \times n$ sparse matrix from the columns of B and place them along the diagonals specified by d

    ```
    S = spdiags(B,d,m,n)
    ```

- Create a 1000×1000 matrix S

$$\begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix}$$

```
M = 1000;
S = spdiags([ones(M,1) -2*ones(M,1) ones(M,1)], [-1 0  1] , M, M);
```

- Compute how much storage this dense matrix need

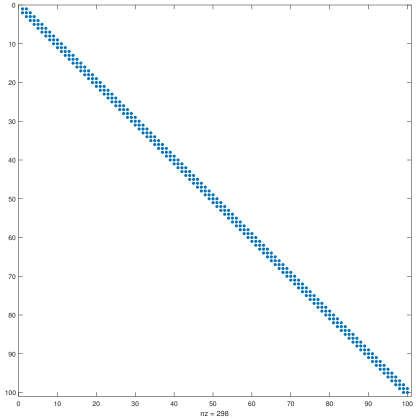
```
s = whos('S');
by = s.bytes;
>> by = 55976 bytes
```

- full() converts a sparse matrix to a dense matrix

```
A = full(S)
```

- spy() plots sparsity structure of a matrix.

```
spy(S)
```



- Do not change sparsity structure
- Indexing in a sparse structures is a expensive procedure
  - Accessing the row and column indexes i, j and changing previous value $S(i, j) = c$ is required
- Accessing values is slow in sparse matrices
  - When an element $S(i, j)$ is requested, a search trough row and column values is needed

M-files

- Text files containing MATLAB programs can be called from
  - the command line
  - the M-files
- Two kind of M-files:
  - Scripts
  - Functions

A Precaution

- Be careful naming files!

  It's easy to get unexpected results:

  - if you give the same name to different functions
  - if you give a name that is already used by MATLAB

- Check new names with the command which.

- It is also useful to include some error checking in your functions.

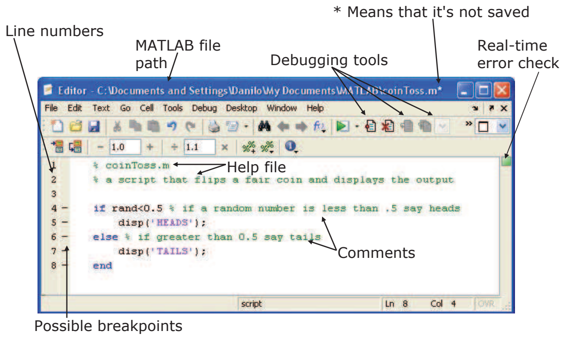
M-Files: Scripts

- Scripts are
  - collection of commands executed in sequence
  - written in the MATLAB editor
  - saved as MATLAB files (.m extension)
- To create an MATLAB file from command-line

```
>> edit helloWorld.m
```

- To open scripts from command window

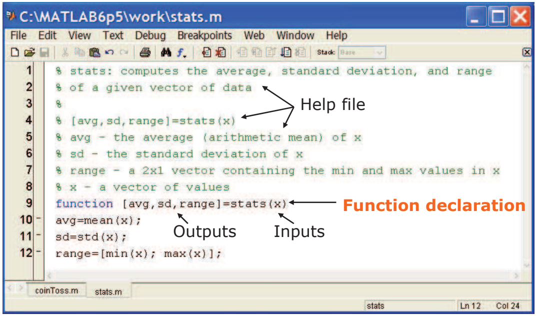
```
>> open helloWorld.m
```


* Means that it's not saved
Line numbers
MATLAB file path
Debugging tools
Real-time error check
Editor - C:\Documents and Settings\Danilo\My Documents\MATLAB\coinToss.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
Help file
Comments
script
Ln 8
Col 4
OVR
Possible breakpoints

```
1    % coinToss.m
2    % a script that flips a fair coin and displays the output
3
4 -  if rand<0.5 % if a random number is less than .5 say heads
5 -      disp('HEADS');
6 -  else % if greater than 0.5 say tails
7 -      disp('TAILS');
8 -  end
```

- COMMENT!
  - Anything following a % is seen as a comment.
  - The first contiguous comment becomes the script's help file.
  - Comment thoroughly to avoid wasting time later.
- Note : Scripts are somewhat static, since there is no input and no explicit output.
- All variables created and modified in a script exist in the workspace even after it has stopped running.

M-Files: Functions



```
% stats: computes the average, standard deviation, and range
% of a given vector of data
%
% [avg,sd,range]=stats(x)
% avg - the average (arithmetic mean) of x
% sd - the standard deviation of x
% range - a 2x1 vector containing the min and max values in x
% x - a vector of values
function [avg,sd,range]=stats(x)
avg=mean(x);
sd=std(x);
range=[min(x); max(x)];
```

- Functions look exactly like scripts, but for ONE difference: Functions must have a function declaration:

function outArguments = NameOfFunAsYouLike(inArguments)

- Variable scope: Any variables created within the function but not returned disappear after the function stops running.

Input

- input prompt the user to input a number or string

```
>> input('Enter a number:', 's')

Enter a number: 5

ans = 5
```

- If a character or string input is desired, 's' must be added after the prompt.

```
>> name = input('Enter a name:  ')

Enter your name:  Mehmet

Error using input

Undefined function or variable 'Mehmet'.


>> name = input('Enter a name:  ','s')

Enter your name:  Mehmet

name = Mehmet
```

Anonymous Functions

- Functions without a file
  - Stored directly in function handle
  - Store expression and required variables
  - Zero or more arguments allowed
  - Nested anonymous functions permitted
- Array of functions handle not allowed; function handle may return array

```
>> f = @(x,y) x^2 + y^2;
>> f(1,2)
ans = 5

>> ezplot(@(x,y) x.^4 + y.^4 -1,[-1,1])
>> ezsurf(@(x,y) exp(-x.^2 -2*y.^2))
```

Local Functions

- A given MATLAB file can contain multiple functions:
- The first function is the `main` function
  - Callable from anywhere, provided it is in the search path
- Other functions in file are local functions
  - Only callable from main function or other local functions in same file
  - Enables modularity (large number of small functions) without creating a large number of files
  - Unfavorable from code reusability standpoint

Local Function Example

- Contents of loc_func_ex.m

```
function main out = loc_func_ex()
    main out = ['I can call the ',loc func()];
end

function loc_out = loc_func()
    loc_out = 'local function';
end
```

- Command-line

```
>> loc_func_ex()
ans =
I can call the local function

>> ['I can''t call the ',loc_func()]
??? Undefined function or variable 'loc_fun
```

For More Information

- http://iam.metu.edu.tr/scientific-computing
- https://iam.metu.edu.tr/scientific-computing-lecture-series
- https://www.facebook.com/SCiamMETU/

For More Information

- http://iam.metu.edu.tr/scientific-computing
- https://iam.metu.edu.tr/scientific-computing-lecture-series
- https://www.facebook.com/SCiamMETU/

...thank you for your attention !