# Lab 1: Introduction to MATLAB

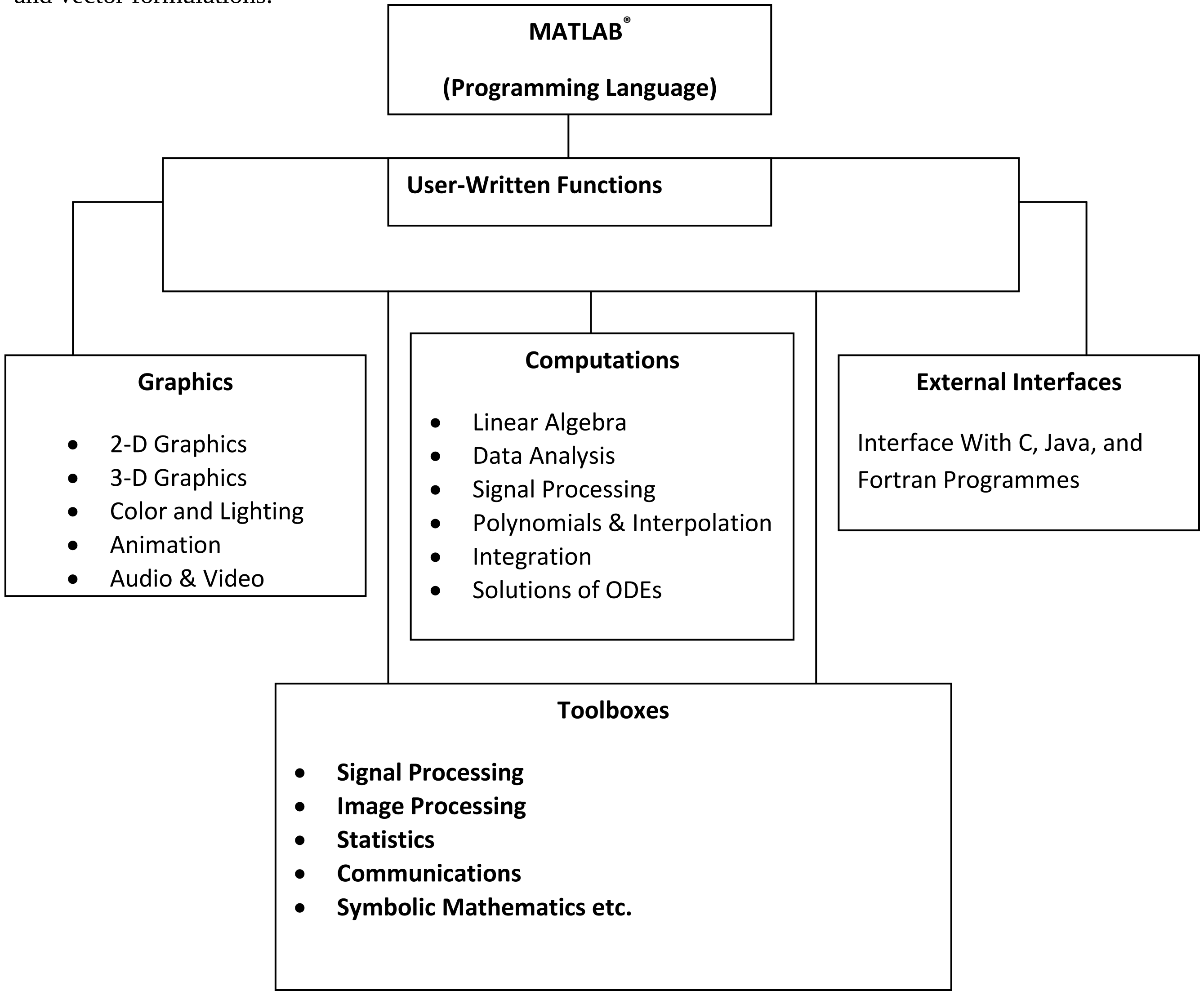
**Aim:** Introduction to MATLAB and its various applications.

## <u>What is MATLAB?</u>

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.

Typical uses include Math and computation Algorithm development, Data acquisition Modeling, simulation, and prototyping Data analysis, exploration, and visualization Scientific and engineering graphics Application development, including graphical user interface building.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations.



The name MATLAB stands for ***Matrix Laboratory***.

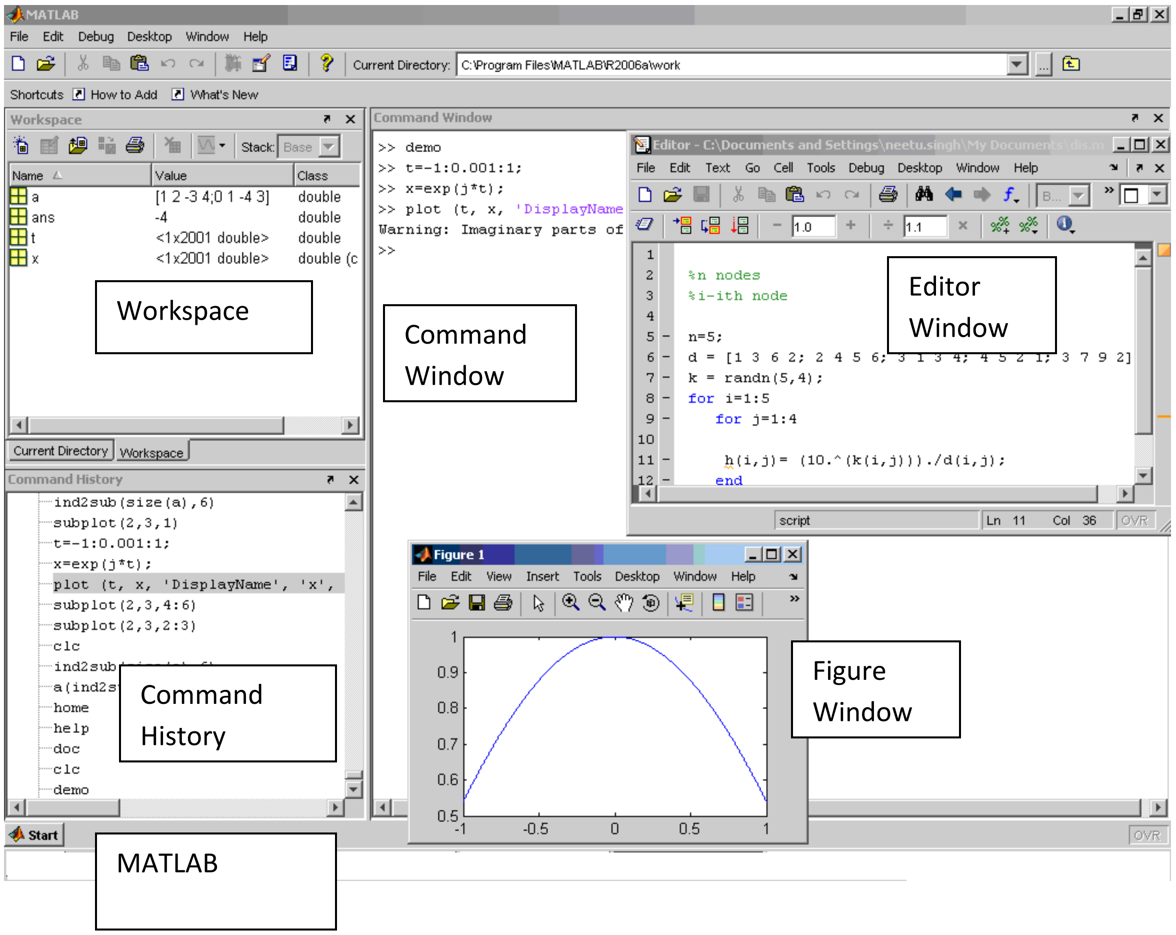
## Theory:

### A. MATLAB Basics

#### 1. MATLAB Windows

- **MATLAB Desktop –** Double Click on the MATLAB program icon to launch MATLAB. This is where MATLAB puts you when you launch it. It consists of the following subwindows.
    - **a. Command Window –** This is main window. It is characterized by the MATLAB command prompt (>>).Use the Command Window to enter variables and to run functions and M-file scripts. All commands, including those for running user-written programs, are typed in this window at the MATLAB command prompt. To execute a command, you must press enter or return at end.
    - b. **Current Directory –** All the files from the current directory are listed here. You have several options of what you can do with file once you select it using mouse. You can run, M-files, rename them, delete them, etc., using right click of mouse.
    - c. **Workspace –** The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces.
    - d. **Command History -** Statements you enter in the Command Window are logged in the Command History. From the Command History, you can view previously run statements, as well as copy and execute selected statements. You can also create an M-file from selected statements.
- **Figure Window -** MATLAB directs graphics output to a window that is separate from the Command Window. In MATLAB this window is referred to as a figure. The characteristics of this window are controlled by your computer's windowing system and MATLAB figure properties. Graphics functions automatically create new figure windows if none currently exist. If a figure already exists, MATLAB uses that window. If multiple figures exist, one is designated as the current figure and is used by MATLAB (this is generally the last figure used or the last figure you clicked the mouse in).
- **Editor Window -** Use the Editor/Debugger to create and debug M-files, which are programs you write to run MATLAB functions. The Editor/Debugger provides a graphical user interface for

text editing, as well as for M-file debugging. To create or edit an M-file use File -> New or File -> Open, or use the edit function.


MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\Program Files\MATLAB\R2006a\work
Shortcuts How to Add What's New
Workspace
Stack: Base
Workspace
Command Window
Editor Window
Command History
Current Directory Workspace
Editor - C:\Documents and Settings\neetu.singh\My Documents\dis.m
File Edit Text Go Cell Tools Debug Desktop Window Help
script Ln 11 Col 36 OVR
Figure 1
File Edit View Insert Tools Desktop Window Help
Figure Window
Start
OVR
MATLAB

| Name | Value | Class |
|---|---|---|
| a | [1 2 -3 4;0 1 -4 3] | double |
| ans | -4 | double |
| t | <1x2001 double> | double |
| x | <1x2001 double> | double (c |

```
>> demo
>> t=-1:0.001:1;
>> x=exp(j*t);
>> plot (t, x, 'DisplayName
Warning: Imaginary parts of
>>
```

```
%n nodes
%i-ith node

n=5;
d = [1 3 6 2; 2 4 5 6; 3 1 3 4; 4 5 2 1; 3 7 9 2]
k = randn(5,4);
for i=1:5
   for j=1:4

    h(i,j)= (10.^(k(i,j)))./d(i,j);
   end
```

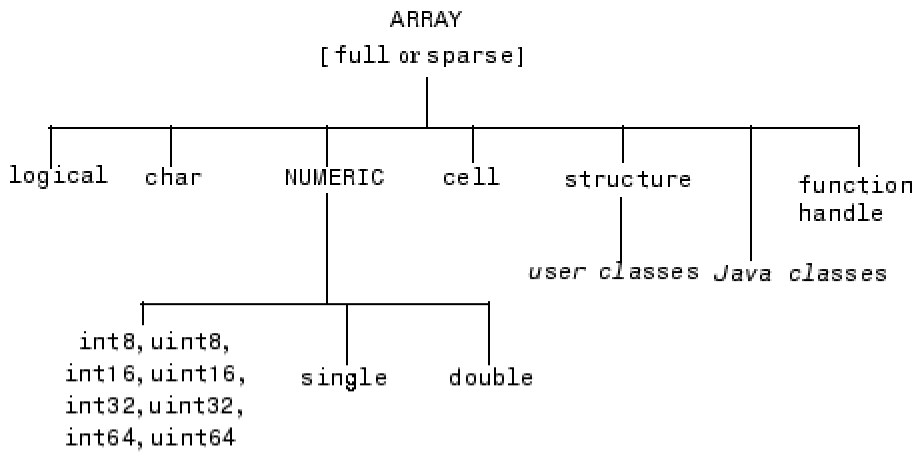
```
ind2sub(size(a),6)
subplot(2,3,1)
t=-1:0.001:1;
x=exp(j*t);
plot (t, x, 'DisplayName', 'x',
subplot(2,3,4:6)
subplot(2,3,2:3)
clc
ind2sub
a(ind2s
home
help
doc
clc
demo
```

**Figure 1. The MATLAB environment consists of MATLAB Desktop, a figure window, and an editor window. The figure window and the editor window appear only when invoked with appropriate commands.**

## 2. Input-Output

- **Data Type -** There are many different types of data that you can work with in MATLAB. You can build matrices and arrays of floating-point and integer data, characters and strings, logical true and false states, etc. You can also develop your own data types using MATLAB classes. Two of the MATLAB data types, structures and cell arrays, provide a way to store dissimilar types of data in the same array.

There are 15 fundamental data types in MATLAB. Each of these data types is in the form of a ***matrix or array.*** This matrix or array is a minimum of 0-by-0 in size and can grow to an n-dimensional array of any size.



- **Dimensioning –** automatic in MATLAB.
- **Case sensitivity – Case Sensitive,** MATLAB requires an exact match for variable names. For example, if you have a variable a, you cannot refer to that variable as A. With respect to functions, filenames, objects, and classes on the search path or in the current directory, MATLAB prefers an exact match with regard to case. MATLAB runs a function if you do not enter the function name using the exact case, but displays a warning the first time you do this.
- **Output Display –** The output of every command is displayed on the command window unless MATLAB is directed otherwise. A semicolon at the end of a command suppresses output.

  Output Format - The number of digits displayed is not related to the accuracy. The display format is set by typing **format type** on the command line, type **format short e** for scientific notation with 5 decimal places, **format long e** for scientific notation with 15 significant decimal places and **format bank** for placing two significant digits to the right of the decimal, default is **format short**.

- **Command History -** The Command History window presents a log of the statements most recently run in the Command Window. Use the arrow, tab, and control keys on your keyboard to recall, edit, and reuse functions you typed earlier. Instead of retyping the entire line, press the up arrow key. The previously typed line is redisplayed. Use the left arrow key to move the cursor, edit if required, and press Enter or Return to run the line. Repeated use of the up arrow key recalls earlier lines, from the current and previous sessions. Using the up arrow key, you can recall any line maintained in the Command History window. Similarly, specify the first few characters of a line you entered previously and press the up arrow key to recall the previous line.

3. **File Types**

- **M-files -** MATLAB provides a full programming language that enables you to write a series of MATLAB statements into a file and then execute them with a single command. You write your program in an ordinary ASCII text file, giving the file a name of filename.m. **The term you use for filename becomes the new command that MATLAB associates with the program**. The file extension of .m makes this a MATLAB M-file. M-files can be scripts that simply execute a series of MATLAB statements, or they can be functions that also accept input arguments and produce output.
- **Mat – files -** The ***save*** command in MATLAB saves the MATLAB arrays currently in memory to a binary disk file called a MAT-file. The term MAT-file is used because these files have the extension .mat. The ***load*** command performs the reverse operation. It reads the MATLAB arrays from a MAT-file on disk back into MATLAB workspace. MAT-files provide a convenient mechanism for moving your MATLAB data between different platforms and for importing and exporting your data to other stand-alone MATLAB applications.
- **Fig – files –** Binary files with a .fig extension that can be opened again in MATLAB as figures. Such files are created by saving a figure in this format using save or save as options from File menu of Figure window. This contains all the information required to recreate the figure. To open use ***open filename. fig***.
- **P – files -** You can save a preparsed version of a function or script, called P-code files, for later MATLAB sessions using the ***pcode*** function. If you develop an application that other people can use but you do not want to give source code, then you give them the corresponding p- files.
- **Mex – files** – are MATLAB callable FORTTRAN and C programs, with a .mex extension to the filenames.

4. **Conversing with MATLAB and General Commands**

**who -** MATLAB replies with the variables in your workspace.

**whos -** MATLAB replies with the variables in your workspace with their size.

**what -** MATLAB replies with the current directory and MATLAB files in the directory

**clear –** clears variables from environment

**clear all -** clears all variables from environment

**clear a b -** look at workspace, the variables a and b are gone

**clc/home - ** clears the cluttered command window.

**close all –** closes all figure windows

**clf –** clears figure window

**pwd –** shows current working directory

**cd -** changes current working directory

**dir/ls –** lists content of current working directory

**Path –** gets or sets MATLAB search path

**Edit path –** edits MATLAB search path

**copyfile –** copies a file

**mkdir –** creates a directory

**help –** Lists topic on which help is available. (The most important function for learning MATLAB on your own).

**help function name –** Lists help on function usage, related functions, and links to help document. To get info on how to use function)

**doc *topic* -** To get a nicer version of help with examples and easy-to-read descriptions

**doc + search tab -** To search for a function by specifying keywords

**lookfor *string*–** lists help topic containing string

**demo –** runs the demo program

**load -** To load variable bindings into the environment

**save -** To save variables to a file

**Control+C –** local abort, terminate current command execution

**Quit/Exit –** exits MATLAB

**B. Scripts**

Scripts are collection of command executed in sequence, written in editor and saved as M-files (.m extension). A script file is executed by typing its name in at command prompt. All of the MATLAB examples in this textbook are contained in M-files that are available at the MathWorks ftp site.

MATLAB requires that the M-file must be stored either in the working directory or in a directory that is specified in the MATLAB path list. For example, consider using MATLAB on a PC with a user-defined M-file stored in a directory called "\MATLAB\MFILES". Then to access that M-file, either change the working directory by typing **cd\matlab\mfiles** from within the MATLAB command window or by adding the directory to the path. Permanent addition to the path is accomplished by editing the **\MATLAB\matlabrc.m** file, while temporary modification to the path is accomplished by typing path(path,'\matlab\mfiles') from within MATLAB.

**C. Making and Manipulating Variables**

MATLAB is a weakly typed language and there is **No** need to initialize variables.

MATLAB supports various types, the most often used are

| | |
|---|---|
| **» 3.84** | 64-bit double (default) |
| **» ‘a’** | 16-bit char |

Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc.

- First character of **variable names** must be a LETTER after that, any combination of letters, numbers and _.
- Variable names are CASE SENSITIVE (a is different from A)

**Some examples of Built-in variables**

- **i** and **j** can be used to indicate complex numbers
- **pi** has the value 3.1415926…
- **ans** stores the last unassigned value (like on a calculator)

- **Inf** and **-Inf** are positive and negative infinity
- **NaN** represents ‘ Not a Number’

A variable can be given a value explicitly

```
» a = 10          shows up in workspace
```

Or as a function of explicit values and existing variables

```
» c = 1.3*45-2*a
```

To suppress output, end the line with a semicolon

```
» cool = 13/3;
```

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)

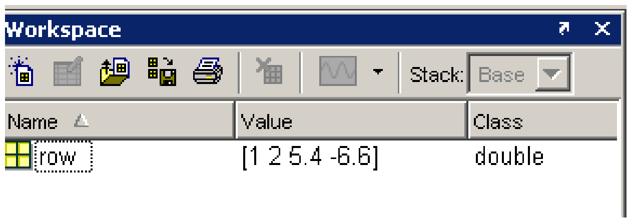
**Row vector**: comma or space separated values between brackets

**» row = [1 2 5.4 -6.6]** or **» row = [1, 2, 5.4, -6.6];**

Command Window:

```
>> row = [1 2 5.4 -6.6]

row =

   1.0000    2.0000    5.4000   -6.6000
```

Workspace:



| Name | Value | Class |
|---|---|---|
| row | [1 2 5.4 -6.6] | double |

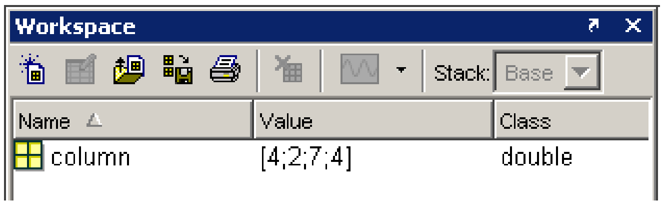
**Column vector**: semicolon separated values between brackets

**» column = [4; 2; 7; 4]**

Command window:        >> column = [4; 2; 7; 4]

```
column =

   4

   2

   7

   4
```

Workspace:



| Name | Value | Class |
|---|---|---|
| column | [4;2;7;4] | double |

You can tell the difference between a row and a column vector by:

- Looking in the workspace
- Displaying the variable in the command window
- Using the size function

```
>> size(row)                            >> size(column)

ans =                                   ans =


     1     4                                  4     1
```

To get a vector's length, use the length function

```
 >>length (row)                    >> length(column)

ans =                              ans =


        4                                   4
```

**Make matrices like vectors**

- Element by element **>> a= [1 2; 3 4]**

```
a =


  1    2

  3    4
```

- By concatenating vectors or matrices (dimension matters)

```
»a = [1 2];

»b = [3 4];

»c = [5; 6];

»d = [a; b];

»e = [d c];

»f = [[e e] ; [a b a]];

»str = ['Hello, I am '  'John'];
```

Strings are character vectors

**Basic Scalar Operations**

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

  **» 7/45**

**» (1+i) * (2+i)**

**» 1 / 0**

**» 0 / 0**

- Exponentiation (^)

  **» 4^2**

  **» (3+4*j) ^ 2**

- Complicated expressions, use parentheses

  **» ((2 + 3) * 3) ^ 0.1**

- Multiplication is **NOT** implicit given parentheses

  **» 3 (1 + 0.7)** **gives an error**

- The transpose operators turn a column vector into a row vector and vice versa

  **» a = [1 2 3 4+i]**

  **» transpose(a)**

  **»a'**

  **»a.'**

The **'** gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers

For vectors of real numbers **.'** and **'** give same result.

If your expression does not fit on one line, use an ellipsis (three or more periods at the end of the line) and continue on the next line.

```
» c = 1+2+3+...

5+6+7;
```

**Built-in Functions**

MATLAB has an enormous library of built-in functions

• Call using parentheses –passing parameter to function

**»sqrt(2)**

**»log(2), log10(0.23)**

**»cos(1.2), atan(-.8)**

**»exp(2+4*i)**

**»round(1.4), floor(3.3), ceil(4.23)**

**»angle(i); abs(1+i);**

Note: Explore these functions using help or doc

MATLAB is based on matrix and vector algebra; even scalars are treated as 1x1 matrices.

Therefore, vector and matrix operations are as simple as common calculator operations.

## Addition and Subtraction

Addition and subtraction are element-wise; sizes must match (unless one is a scalar):

• The following would give an error

**» c = row + column**

• Use the transpose to make sizes compatible

**»c = row'+ column**

**»c = row + column'**

• Can sum up or multiply elements of vector

**» s = sum (row);**

**» p = prod (row);**

**Element-Wise Functions**

•All the functions that work on scalars also work on vectors

**»t = [1 2 3];**

**»f = exp(t);** is the same as **»f = [exp(1) exp(2) exp(3)];**

•If in doubt, check a function's help file to see if it handles vectors elementwise

•Operators (* / ^) have two modes of operation

- element-wise
- standard

**Operators: element-wise**

•To do element-wise operations, use the dot: '**.**'(.* , ./ , .^). BOTH dimensions must match (unless one is scalar)!

**»a = [1 2 3]; b = [4; 2; 1];**

**»a .* b, a./ b, a. ^ b** all errors

**»a .*b', a./ b', a .^ (b')** all valid

**Operators: standard**

•Multiplication can be done in a standard way or element-wise

•Standard multiplication (*) is either a dot-product or an outer-product

•Standard exponentiation (^) can only be done on square matrices or scalars

•Left and right division (/ \) is same as multiplying by inverse

**Automatic Initialization**

•Initialize a vector of ones, zeros, or random numbers

**»o = ones(1,10)** % row vector with 10 elements, all 1

**»z = zeros(23,1)** % column vector with 23 elements, all 0

**»r = rand(1,45)** % row vector with 45 elements (uniform [0,1])

**»n = nan(1,69)** % row vector of NaNs (useful for representing uninitialized variables)

The general function call is:

>> **var = zeros(M,N);** where M is Number of rows and N is Number of columns.

Note: Explore these functions using help or doc

To initialize a linear vector of values use linspace

**»a = linspace (0, 10 ,5)** % starts at 0, ends at 10 (inclusive), 5 values

• Can also use colon operator (:)

**»b = 0:2:10** % starts at 0, increments by 2, and ends at or before 10

% increment can be decimal or negative

**»c = 1:5** % if increment isn't specified, default is 1

•To initialize logarithmically spaced values use **logspace**, similar to **linspace**, but see help

**Vector Indexing**

•MATLAB indexing starts with 1, not 0

•a(n) returns the nth element.

•The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.

```
»x = [12 13 5 8];

»a = x (2:3);          a = [13   5];

»b = x (1: end-1);     b = [12 13 5];
```

**Matrix Indexing**

•Matrices can be indexed in two ways

- using subscripts(row and column)
- using linear indices(as if matrix is a vector)

•Picking submatrices

```
»A = rand(5)           % shorthand for 5x5 matrix

»A(1:3,1:2)            % specify contiguous submatrix

»A([1 5 3], [1 4])     % specify rows and columns
```

To select rows or columns of a matrix, use the:

```
>> c = [12 5; -2 13]

c =

   12    5

   -2   13
```

```
» d = c(1, :)          >> e = c(:, 2)          >> c (2, :) = [3 6]

d =                    e =                     c =

     12    5                5                       12    5

                           13                        3    6

                                               % replaces second row of c
```

MATLAB contains functions to help you find desired values within a vector or matrix

**»vec = [5 3 1 9 7]**

•To get the minimum value and its index:

**»[minVal, minInd] = min(vec);** % **max** works the same way

•To find any the indices of specific values or ranges

**»ind = find(vec == 9);**

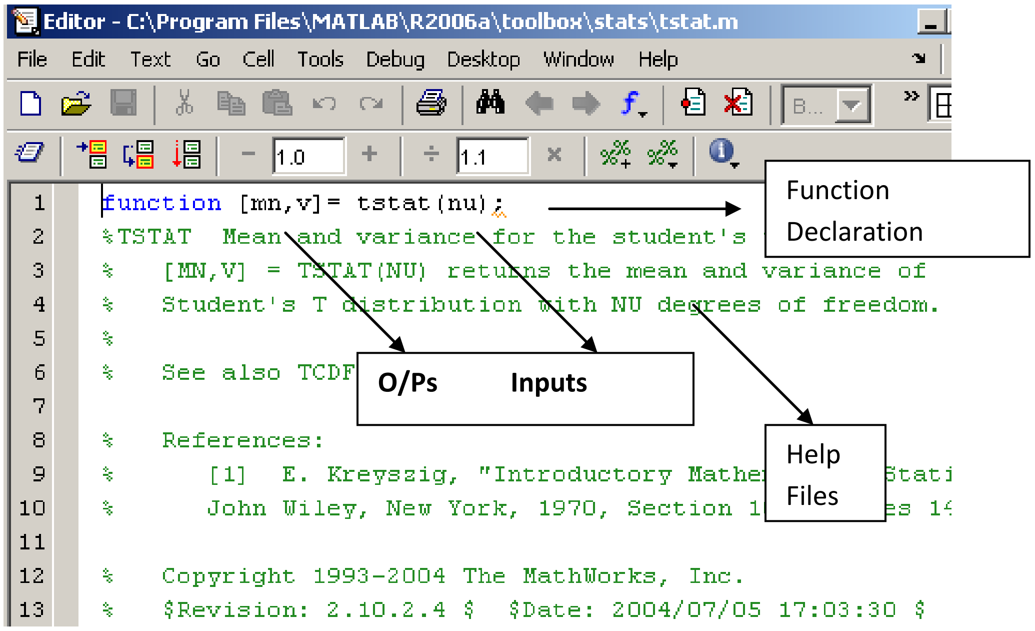
**»ind = find(vec > 2 & vec < 6);** %find expressions can be very complex,

•To convert between subscripts and indices, use **ind2sub**, and **sub2ind.** Use help.

**D. Functions**

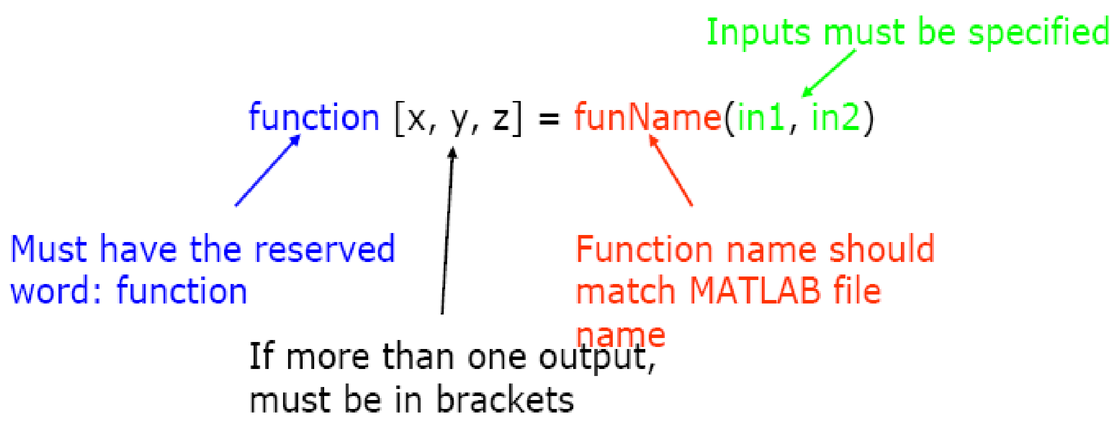
User-defined Functions

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


Editor - C:\Program Files\MATLAB\R2006a\toolbox\stats\tstat.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
Function Declaration
O/Ps
Inputs
Help Files

```
function [mn,v]= tstat(nu);
%TSTAT  Mean and variance for the student's
%    [MN,V] = TSTAT(NU) returns the mean and variance of
%    Student's T distribution with NU degrees of freedom.
%
%    See also TCDF

%    References:
%       [1]  E. Kreyszig, "Introductory Mathe         Stati
%       John Wiley, New York, 1970, Section 1         es 1

%    Copyright 1993-2004 The MathWorks, Inc.
%    $Revision: 2.10.2.4 $  $Date: 2004/07/05 17:03:30 $
```

Some comments about the function declaration


Inputs must be specified
function [x, y, z] = funName(in1, in2)
Must have the reserved word: function
If more than one output, must be in brackets
Function name should match MATLAB file name

No need for return: MATLAB 'returns' the variables whose names match those in the function declaration

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

**E. Flow Control**

Relational Operators < > <= >= == ~=

•MATLAB uses mostly standard relational operators

- equal ==
- notequal ~=
- greater than >
- less than <
- greater or equal >=
- less or equal <=

| •Logical operators | elementwise | short-circuit (scalars) |
|---|---|---|
| • And | & | && |
| • Or | \| | \|\| |
| • Not | ~ | |
| • Xor | xor | |
| • All true | all | |
| • Any true | any | |

•Boolean values: zero is false, nonzero is true

•See **help .** for a detailed list of operators

**if/else/elseif -** Execute statements if additional condition is true

Syntax

```
if expression1
    statements1
elseif expression2
    statements2
end
```

Description

If expression1 evaluates as false and expression2 as true, MATLAB executes the one or more commands denoted here as statements2.

A true expression has either a logical 1 (true) or nonzero value. For nonscalar expressions, (for example, is matrix A less then matrix B), true means that every element of the resulting matrix has a true or nonzero value.

**For -** Execute block of code specified number of times

Syntax

```
for variable = expression
    statements
end
```

Description

The columns of the expression are stored one at a time in the variable while the following statements, up to the end, are executed.

In practice, the expression is almost always of the form scalar: scalar, in which case its columns are simply scalars.

The scope of the “**for”** statement is always terminated with a matching end.

**while -** Repeatedly execute statements while condition is true

Syntax

```
while expression
    statements
end
```

Description

while repeats statements an indefinite number of times. The statements are executed while the real part of expression has all nonzero elements. expression is usually of the form expression relational operator expression where relational operator is ==, <, >, <=, >=, or ~=. The scope of a while statement is always terminated with a matching end.

**F. Plotting**

**»x = linspace(0,4*pi,10);**

**»y = sin(x);**

•Plot values against their index

**»plot(y);**

•Usually we want to plot y versus x

**»plot( x , y);**

Plot generates dots at each (x, y) pair and then connects the dots with a line

•To make plot of a function look smoother, evaluate at more points

**»x = linspace (0, 4*pi, 1000);**

**»plot(x, sin(x));**

•x and y vectors must be same size or else you’ll get an error

**»plot ([1 2], [1 2 3])** error!

•Can change the line color, marker style, and line style by adding a string argument

**»plot (x, y, ’k.-’);**

•Can plot without connecting the dots by omitting line style argument

**»plot (x, y, ’.’)**

•Look at help plot for a full list of colors, markers, and linestyles

Some Other Useful Commands:

**xlabel, ylabel, title, xlim, ylim, grid, axis, hold, stem**

Cartesian Plots Commands: **plot, semilogx, semilogy, loglog**

3D Line Plot Command: **plot3, surf, contour, quiver, xlim, ylim, zlim**

**subplot** for Multiple Plots in one Figure

## G. Solving Equations

Systems of Linear Equations

•Given a system of linear equations

x + 2y - 3z = 5,    -3x – y + z = -8,    x – y + z = 0

•Construct matrices so the system is described by Ax = b

**»A = [1 2 -3; -3 -1 1; 1 -1 1];**

**»b = [5; -8; 0];**

•And solve with a single line of code!

**»x = A\b;**    % x is a 3x1 vector containing the values of x, y, and z

•The \ will work with square or rectangular systems.

•Gives least squares solution for rectangular systems. Solution depends on whether the system is over or underdetermined.

Given a matrix

**»m = [1 2 -3; -3 -1 1; 1 -1 1];**

•Calculate the rank of a matrix

**»r = rank(m);**    % the number of linearly independent rows or columns

•Calculate the determinant

**»d = det(m);**    % m must be square, if determinant is nonzero, matrix is invertible

•Get the matrix inverse

**»E = inv(m);**    % if an equation is of the form A* x = b with A a square matrix, x = A\b is the same as x = inv(A) * b

**Polynomials**

•Many functions can be well described by a high-order polynomial

•MATLAB represents polynomials by a vector of coefficients

if vector P describes a polynomial $ax^3 + bx^2 + cx + d$

•P = [1 0 -2] represents the polynomial $x^2 - 2$

•P = [2 0 0 0] represents the polynomial $2x^3$

P is a vector of length N+1 describing an N-th order polynomial

•To get the roots of a polynomial

**» r = roots(P)** % r is a vector of length N

•Can also get the polynomial from the roots

**»P = poly(r)** % r is a vector length N

**Numerical Differentiation**

•MATLAB can 'differentiate' numerically

**»x = 0:0.01:2*pi;**

**»y = sin(x);**

**»dydx = diff(y) ./ diff(x);** % diff computes the first difference

•Can also operate on matrices

**»mat = [1 3 5; 4 8 6];**

**»dm = diff(mat,1,2)** % first difference of mat along the 2nddimension, dm=[2 2;4 -2]

**Numerical Integration**

•MATLAB contains common integration methods

•Adaptive Simpson's quadrature (input is a function)

**Write a function ' y = myFun(x)' where y = cos (exp(x) +x.^2 - 1)**

**»q1 = quad ('myFun', 0, 10);**        % q1 is the integral of the function myFun from 0 to 10

**»q2 = quad (@(x) sin(x)*x, 0, pi)**   % q2 is the integral of sin(x)*x from 0 to pi

See function handle

•Trapezoidal rule (input is a vector)

**»x = 0:0.01:pi;**

**»z = trapz(x, sin(x));**     %z is the integral of sin(x) from 0 to pi

**H. Data Structures**

**We have used 2D matrices**

- Can have n-dimensions
- Every element must be the same type (ex. integers, doubles, characters…)
- Matrices are space-efficient and convenient for calculation
- Large matrices with many zeros can be made sparse:

**»a = zeros(100); a(1,3) =10; a(21, 5) = pi; b = sparse(a);**

• Sometimes, more complex data structures are more appropriate

- **Cell array**: it's like an array, but elements don't have to be the same type
- **Structs**: can bundle variable names and values into one structure

– Like object oriented programming in MATLAB

## I. Symbolics

Symbolic Variables

•Symbolic variables are a type, like double or char

•To make symbolic variables, use **sym**

```
»a=sym ('1/3');

»b=sym ('4/5');

»mat=sym ([1 2;3 4]);              % fractions remain as fractions

»c=sym ('c', 'positive');          % can add tags to narrow down scope

                                   % see help sym for a list of tags
```

•Or use **syms**

```
» syms x y real                    % shorthand for x=sym ('x', 'real'); y=sym ('y', 'real');
```

**Symbolic Expressions**

• Multiply, add, divide expressions

```
»d = a*b                   % does 1/3*4/5=4/15;

»expand ((a-c) ^ 2);       % multiplies out

»factor (ans)              % factors the expression

»matInv = inv(mat)         % Computes inverse symbolically
```

**Cleaning up Symbolic Statements**

```
»pretty (ans)                                  % makes it look nicer

»collect (3*x+4*y-1/3*x^2-x+3/2*y)             % collects terms
```

**»simplify (cos(x) ^ 2+sin(x)^2)** % simplifies expressions

**»subs (‘c^2’, c, 5)** % Replaces variables with numbers or expressions.

To do multiple substitutions pass a cell of variable names followed by a cell of values

**»subs(‘c^2’, c, x/7)**

**<u>POST LAB EXERCISE</u>**

For problems 1- 4, write a script called **simpleProblems.m** and put all the commands in it. Separate and label different problems using comments.

1. **Scalar variables**. Make the following variables

    a. a = 10

    b. b = 2.5 ×10 $^{23}$

    c. c = 2 + 3i , where i is the imaginary number

    d. d = e$^{j\ 2ð\ /3}$, where j is the imaginary number and e is Euler’s number (use **exp, pi**)

2. **Vector variables.** Make the following variables

a. $aVec = \begin{bmatrix} 3.14 & 15 & 9 & 26 \end{bmatrix}$

b. $bVec = \begin{bmatrix} 2.71 \\ 8 \\ 28 \\ 182 \end{bmatrix}$

c. $cVec = \begin{bmatrix} 5 & 4.8 & \cdots & -4.8 & -5 \end{bmatrix}$ (all the numbers from 5 to -5 in increments of -0.2)

d. $dVec = \begin{bmatrix} 10^0 & 10^{0.01} & \cdots & 10^{0.99} & 10^1 \end{bmatrix}$ (logarithmically spaced numbers between 1 and 10, use **logspace**, make sure you get the length right!)

e. $eVec = Hello$ ( $eVec$ is a string, which is a vector of characters)

3. **Matrix variables.** Make the following variables

a. $aMat = \begin{bmatrix} 2 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 2 & \cdots & 2 \end{bmatrix}$ a 9x9 matrix full of 2's (use **ones** or **zeros**)

b. $bMat = \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ 0 & \ddots & 0 & \ddots & \\ \vdots & 0 & 5 & 0 & \vdots \\ & \ddots & 0 & \ddots & 0 \\ 0 & & \cdots & 0 & 1 \end{bmatrix}$ a 9x9 matrix of all zeros, but with the values $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}$ on the main diagonal (use **zeros**, **diag**).

c. $cMat = \begin{bmatrix} 1 & 11 & \cdots & 91 \\ 2 & 12 & \ddots & 92 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 20 & \cdots & 100 \end{bmatrix}$ a 10x10 matrix where the vector 1:100 runs down the columns (use **reshape**).

d. $dMat = \begin{bmatrix} NaN & NaN & NaN & NaN \\ NaN & NaN & NaN & NaN \\ NaN & NaN & NaN & NaN \end{bmatrix}$ a 3x4 NaN matrix (use **nan**)

e. $eMat = \begin{bmatrix} 13 & -1 & 5 \\ -22 & 10 & -87 \end{bmatrix}$

f. Make *fMat* be a 5x3 matrix of random integers with values on the range -3 to 3 (use **rand** and **floor** or **ceil**).

4. **Common functions and indexing.**

a. Make $cSum$ the column-wise sum of $cMat$. The answer should be a row vector (use **sum**).

b. Make $eMean$ the mean across the rows of $eMat$. The answer should be a column (use **mean**).

c. Replace the top row of $eMat$ with $\begin{bmatrix}1 & 1 & 1\end{bmatrix}$.

d. Make $cSub$ the submatrix of $cMat$ that only contains rows 2 through 9 and columns 2 through 9.

e. Make the vector $lin = \begin{bmatrix}1 & 2 & \cdots & 20\end{bmatrix}$ (the integers from 1 to 20), and then make every other value in it negative to get $lin = \begin{bmatrix}1 & -2 & 3 & -4 & \cdots & -20\end{bmatrix}$.

f. Make $r$ a 1x5 vector using **rand**. Find the elements that have values <0.5 and set those values to 0 (use **find**).

5. **Plotting multiple lines and colors.** Open a script and name it **twoLinePlot.m**

To plot a sine wave and a cosine wave over one period

(a) Make a time vector *t* from 0 to 2ð with enough samples to get smooth lines, Plot sin t.

(b) Use hold on.

(c) Plot cos t using a red dashed line.

(d) Add labels to the plot and create a legend to describe the two lines you have plotted by using **legend.**

(e) Use **xlim** to set the x axis to be from 0 to 2ð and use **ylim** to set the y axis to be from -1.4 to 1.4.

6. **Plot a circle.** Write the function [x, y] = getCircle (center,r) to get the x and y coordinates of a circle. The circle should be centered at center (2-element vector containing the x and y values of the center) and have radius r. Return x and y such that plot(x, y) will plot the circle.

7. **Loops and flow control**. Make function called loopTest(N) that loops through the values 1 through N and for each number n it should display 'n is divisible by 2', 'n is divisible by 3', 'n is divisible by 2 AND 3' or 'n is NOT divisible by 2 or 3'. Use a **for** loop, the function **mod** or **rem**

to figure out if a number is divisible by 2 or 3, and **num2str** to convert each number to a string for displaying. You can use any combination of **if**, **else**, and **elseif.**

8. **Linear system of equations.** Solve the following system of equations:

**3a + 6b + 4c = 1.    a + 5b = 2.        7b + 7c = 3**

9. **Practice with cells.** Usually, cells are most useful for storing strings, because the length of each string can be unique. a. Make a 3x3 cell where the first column contains the names: ‘Joe’, ’Sarah’, and ’Pat’, the second column contains their last names: ‘Smith’, ‘Brown’, ‘Jackson’, and the third column contains their salaries: $30,000, $150,000, and $120,000. Display the cell using **disp**.