A Short Tutorial on Matlab

(by T. Co, ©2003,2004 **)**

Part I. BASICS

I. Assigning Values to Variables, Vectors and Matrices

- Use the equality symbol, e.g. **`A=3`**
- Variables are case sensitive: **`temperature`** is not the same as **`Temperature`**
- To check values, just type the variable name
- There are special constants already available, e.g. **`pi, Inf`**
- For complex numbers, either use **`3+2i`** or **`3+2j`**
- Vectors and matrices are enclosed with square brackets:
  a) Elements of rows are separated by commas: **`A=[1,2,3]`**
  b) Elements of columns are separated by semicolons: **`B=[1;2;3]`**
  c) Matrices can be seen as columns of row vectors, **`C=[1,2,3;4,5,6]`**
- Alternatively, use spaces to separate row elements and return keys for column elements:

```
B = [ 1 2 3
4 5 6
1 0 1]
```

- To get size information,
  a) **`size(A)`** yields [# rows, # cols]
  b) **`size(A,1)`** yields # rows
  c) **`size(A,2)`** yields # cols
  d) **`length(v)`** yields # elements in vector v
- Some shortcuts are available:
  a) **`A = 1:0.1:2`** yields a row vector [1, 1.1, 1.2, … , 1.9, 2]
  b) **`diag([1,2,3])`** yields a diagonal matrix [1,0,0 ; 0,2,0 ; 0,0,3]
  c) **`diag([1,2,3],1)`** yields a matrix [0,1,0,0 ; 0,0,2,0 ; 0,0,0,3 ; 0,0,0,0]
  d) **`eye(4)`** yields an identity matrix of size 4x4
  e) **`zeros(3)`** yields a matrix of zeros of size 3x3
  f) **`zeros(size(A))`** yields a matrix of zeros having the same size as A
  g) **`ones(3), ones(2,5), ones(size(A))`** works the same as zeros() except it produces a matrix of ones
  h) **`rand(m,n)`** matrix of uniformly distributed random numbers of size m by n
  i) **`randn(m,n)`** matrix of normally distributed random numbers of size m by n
  j) **`linspace(a,b,n)`** creates a vector of n elements linearly spaced from a to b

k) **`logspace(a,b,n)`** creates a vector of n elements logarithmically spaced from 10^a to 10^b

- Vectors and matrices can be appended to build larger matrices, e.g. **`E=[A,B;C,D]`**
- To disable the echoing results you could include a semicolon after each command line.
- To check existing list, you could type **`who`** or **`whos`**
- To extract elements:
  - a) **`A(1,2)`** gives the element at row 1 and column 2
  - b) **`A(3,:)`** gives the third row of A
  - c) **`A(:,4)`** gives the fourth column of A
  - d) **`A([3,2],[2,1])`** extracts rows 3 and 2, and columns 2 and 1
- To remove rows and columns:
  - a) **`A(2,:)=[]`** removes the 2nd row
  - b) **`A(:,1)=[]`** removes the 1st column

- Reshaping and vectorization
  - a) **`A(:)`** yields a vectorized column $[a_{11}; a_{21}; \ldots; a_{n1}; \ldots; a_{1n}; \ldots; a_{nn}]$
  - b) **`reshape(A,m,n)`** yields a matrix of size m by n by building it column-wise

II. Matrix Operations and other operations

- *matrix operations*
  1. addition **`C=A+B`**
  2. multiplication **`C=A*B`**
  3. determinant **`c=det(A)`**
  3. inverse **`inv(A)`** or **`A^(-1)`**
  4. right division **`C=B/A`** (same as **`C=B*inv(A)`**)
  5. left division **`C=A\B`** (same as **`C= inv(A)*B`**)
  6. transpose **`C=A.'`** (Note: **`C=A'`** is conjugate transpose)
  7. exponentiation **`B=A^3`**
  8. matrix exponentiation **`B=expm(A)`**
  9. rank **`R=rank(A)`**
  10. eigenvalue/eigenvectors **`[Eigenvals,Eigenvecs] = eig(A)`**
  11. characeristic polynomial **`p = poly(A)`**
  12 Kronecker product **`C=kron(A,B)`**
  14. trace **`t=trace(A)`**
  15. pseudo-inverse **`M = pinv(A)`**

- *polynomials*
  16. roots of polynomial — **`R = roots([ 1, 4, 2] )`**
      (= roots of $s^2+4s+2=0$)
  17. polynomial, given roots — **`p=poly([r1,r2])`**
  18. polynomial multiplication — **`p=conv( [1, 4, 2],[-2, 3] )`**
      (= $(s^2 + 4s + 2)(-2s+3)$)

- *column-wise operation*
  19. column-wise sum — **`s=sum([1,2,3;4,5,6])`**
      (= [5,7,9])
  20. column-wise product — **`p=prod([1,2,3;4,5,6])`**
      (= [4, 10, 18])

- *sorting*
  21. sort cols in ascending order — **`V = sort(A)`**
      if A = [5, 3] then V= [1, 3]
      [1, 4] [5, 4]
  22. sorting and getting indices — **`[V,K] = sort(A)`**

- Elementwise operation: precede operator by a dot, e.g. **`A.*B`**, **`A./B`**, **`A.^2`**

III. Plotting

- *2D plots*

`plot(x,y)`	uses default
`plot(x,y,x,z)`	combines plot of y vs x and z vs x
`plot(x,y,'-',x,z,'--')`	specifies how each plot is drawn: solid line for x-y and dashed for x-z

- *Labeling*

`xlabel('Time (sec)')`	labeling the x-axis
`ylabel('Temperature')`	labels the y-axis
`title('Case 1')`	puts title on top of plot
`text(1.2, 3.0, 'Case 1')`	puts text on specified location
`gtext('Case 2')`	text is place with the aid of the mouse

(or use the menu bar in the figure window)

You can use a limited version of TeX formatting for

a) greek letters and other symbols (see TeX manual)
   `' \alpha \nabla T '` yields a text : $\alpha\nabla T$
b) subscript and superscripts:
   `'T_{surround}^{a+b}'` yields a text : $T_{surround}^{a+b}$

- *Axis settings*

axis([0,10,-1,1])	specifies x-range as 0 to 10, and y-range as -1 to 1
axis('square')	makes the plot square
axis	returns to automatic axis mode
hold	freezes the previous plot

- *3D plots*

[x,y]=meshgrid(-10:0.5:10, -2:0.2:2)
initializes x and y matrices

(for the following try **z=sin(x/2).*sin(y)**)

mesh(x,y,z)	mesh plot
surf(x,y,z)	surface plot
surfl(x,y,z)	surface plots with lighting
shading interp	smoothens shading scheme as interpolated
colormap(gray)	changes coloring scheme to gray scale
brighten(-0.5)	darkens picture (if >0 then it brightens)
view([10,60])	changes view to azimuth=10^o and elevation=60^o

IV. Printing

print	sends plot to printer
print h:\sample1.ps	saves the plot as a file
print -deps h:\sample2.eps	specifies format

(or you can use the menu in the figure window)

V. Workspaces

clear	erases all variables
clear a, B, Temperature	erases only specified variables
diary h:\case1.dry	saves all terminal activity and saves in file **case1.dry**
save h:\assign1.mat	saves workspace
save h:\assign1.mat a, B	saves only specified variables
save h:\assign1.mat a, B -ascii	saves in ascii format
load h:\assign1.mat	loads workspace
load h:\data1.any	loads from a file and creates a matrix named **data1**

(or you can use menu items for importing data and changing paths)

VI. Some helpful tips

`help`	grabs help information from functions or script files
cursor up and down keys	scrolls through previous commands

VI. Multidimensional Arrays

a) Creating arrays of higher dimensions

- **Method 1**: set using assignments
  Given: **`A=[1,2; 3,4]`**
  Then type: **`A(:,:,2)= [11, 22; 33, 44]`**

- **Method 2**: using cat function

  Given: **`A=[1,2 ; 3,4]`**
  `B = [11 , 22; 33, 44]`
  Then type: **`C=cat(3,A,B)`**

- **Method 3**: using "**`ones`**", "**`zeros`**" and "**`repmat`**"

b) Dimensions and Reshaping arrays

`ndims(A)`	gets the number of dimensions of A
`size(A)`	gets the size of each dimension of A
`reshape(A,[m,n,p])`	reshapes A into a multidimensional array of size [m,n,p] by first collecting all elements of A in a column and then distributing according to the specified size.

c) Permuting arrays

`permute(A,[2,1,3])` (where 1,2,3 are the dimensions, not sizes)	changes the dimension indexing, this is essentially like the transpose, except it applies on any dimension.

VII. Structures

Features/Motivation:

- helps encapsulate and organize information of different types
- uses text fields, instead of numeric fields
- can be collected as arrays
- can have subfields
- most commands require the options to come from structures
- most commands can output to structures

a) Creating Structures

- **Method 1**: Use period to create fields and subfields

```
student.IDNumber=1234;
student.Name='Hank Hill';
```

- **Method 2**: Use the **`struct`** command

```
student=struct('IDNumber',1234,...
     'Name','Hank Hill');
```

- For building structure arrays, just start numbering

```
student(2)=struct('IDNumber',5678,...
     'Name','Robert Plant');
```

b) Extracting Information: just type the structure name including the fields

```
student(2)
student(2).Name
```

c) Removing fields

```
student = rmfield( student , 'IDNumber' )
```

VIII. Cell Arrays

Features/Motivation

- Cell arrays are similar to structures except you use numeric indices.
- Instead of matrices, cell arrays can have different types of elements

a) Creation:

Use assignments with "curly brackets"

```
A = { [1,2;3,4],  'Testing'; {[1,2] ;3}} , 3+5i }
```

b) Extracting information:

`A{1,1}`	should yield the element in {1,1}
`A{2,1}{1,1}`	assuming element {2,1} is a cell, then extracts the {1,1}th element

c) Translating to structures and vice versa:

```
C=cell2struct(A(:),...
    {'matrix','name','vector','cell'})
```

`D=struct2cell(C);`	this converts structure C to cell array arranged in a column
`E=reshape(D,2,2);`	this will reshape cell array D

IX. M-files

Features/Motivation

- using the Matlab editor or any text editor such as notepad
- to write a user-define function that could take different inputs and yields different outputs
- collect subfunctions together for better organization of files
- several Matlab functions need user-supplied functions in forms of m-files.

a) **Example 1. (simple function)**

```
function    y = example1(A,x,b)
%
%   y = example1(A,x,b)
%   ===================
%
%   solves for y=Ax+b
%

%   (c)2004 Tom Co
%   @ Michigan Technological University
%

    y = A*x + b ;
```

b) **Example 2. (different inputs and outputs)**

```
function [y,norm] = example2(A,x,b)
%
%   [y,norm] = example2(A,x,b)
%   ==========================
%
%   obtains
%       a) y = Ax+b
%       b) or y = Ax if b not included
%       c) norm = sqrt( y' * y )

    y = A*x;

    if  nargin==3
        y = y + b;
    end

    if  nargout == 2, % if norm is required
        norm=y'*y;
    end
```

c) Example 3. (includes subfunctions)

```
function ynorms = example3(Y)
%
%   ynorms = example3(Y)
%   ====================
%
%   obtains ynorms = [ ynorm1, ..., ynormn ]
%   where   ynormi= xnorm( y(:,i) )
%           xnorm(x) = sqrt(x'*x)
%


    nycols = size(Y,2);
    ynorms =[];
    for i = 1:nycols
        ycol = Y(:,i);
        ynorms=[ynorms, newnorm(ycol)];
    end

    function xnorm = newnorm(x);
        xnorm = sqrt(x'*x);
```