A BRIEF INTRODUCTION TO MATLAB

by
Christina Christara and Winky Wai

December 2001

1. What is MATLAB?

MATLAB is an object-oriented high-level interactive software package for scientific and engineering numerical computations. Its name stands for *matrix laboratory*. MATLAB enables easy manipulation of matrix and other computations without the need for traditional programming. MATLAB's basic data element is the matrix.

2. How to use MATLAB.

MATLAB can be started up by the

```
% matlab -nodesktop
```

or the

```
% matlab
```

commands at the Unix shell prompt. The `matlab -nodesktop` command starts up MATLAB in your current window (most likely an `xterm`), and can work on any ASCII terminal; it gives you a simple and light ASCII interface. The `matlab` command starts up MATLAB in a new window of the console you are on, assuming you are running Xwindows; it gives you a more powerful, but also complex and heavy interface. With `matlab` you will get one main window with various menus and buttons, and one (or more) inner windows, (e.g. a Command Window, a History Window, etc.), the collection of which is called **desktop** environment. If you run `matlab`, it is suggested that you start it up with `matlab &`, so that you can re-use the window in which you started it. Note that you can **not** get the desktop environment from a non-Xwindow system. Thus, if you are at home using a PC and Windows, and run telnet (ssh) to `eddie.cdf`, the command `matlab` will give you the simple ASCII interface. The standard MATLAB prompt is `>>` . In the following, whatever follows `>>` on a line starting with `>>` is typed by the user at the MATLAB prompt.

To quit MATLAB, type

```
>> quit
```

or

```
>> exit
```

2.1. M-files, functions and scripts.

MATLAB has a lot of built-in functions. It also allows users to define their own. Functions are saved in **M-files**, called so because the filename ends in `.m`. Functions may have one or more input parameters and may return one or more output variables. A M-file does not always need to be a function. It can also be a long sequence of MATLAB statements, i.e. a script. Scripts have neither formal input parameters, nor formal output variables.

2.2. Invoking MATLAB functions and scripts.

To execute a function M-file, type the name of the file without the `.m` extension, followed by the input arguments in parentheses. Should you need to save the output variables, precede the call by the output variables (in square brackets if more than one) and the '=' sign. For example,

```
>> lu(A)
```

displays the LU decomposition of matrix A, while

```
>> [L, U] = lu(A)
```

also saves the lower and upper triangular factors in matrices L and U respectively.

To execute a script M-file, just type the name of the file without the `.m` extension.

When MATLAB executes a statement in which a function or script is invoked, it checks if it is a built-in function. If it is not a built-in function, it assumes it is a user-defined function or script and goes through a search path of directories to look for that function/script. The first directory on the path is the current directory that you start up MATLAB. If the function/script is not found in this directory, MATLAB searches in a directory called `matlab` under your home directory. You can build up your own MATLAB library by creating such a directory and putting the M-files of all functions/scripts you write in it. Such a setup will allow the functions and scripts in the `matlab` directory to be used no matter in which directory you run MATLAB.

2.3. Setting your environment.

You can pick a directory name other than the default `matlab` for your own MATLAB library, if you wish. For example, suppose you want to have the M-files corresponding to your own MATLAB functions and scripts in a directory called `ownfunc` under your home directory. Then, include the following line in your `.cshrc` file (this file is in your home directory):

```
setenv MATLABPATH $HOME/ownfunc
```

Here, `$HOME` is the shell environment variable that gives the name of your home directory. You can arrange to have your MATLAB functions/scripts in more than one directory or subdirectory, if you wish. For example, suppose you want to have some functions/scripts in `ownfunc`, and some other functions/scripts in the directory `morefunc` under your `350` directory. Then, include the following line in your `.cshrc` file instead

```
setenv MATLABPATH $HOME/350/morefunc:$HOME/ownfunc
```

To find out what is the search path of MATLAB, after you get into MATLAB, type

```
>> path
```


2.4. On-line help.

MATLAB offers a lot of on-line help. Type

```
>> help
```

to get all the help topics and

```
>> help topic
```

to get information on any MATLAB topic. For example,

```
>> help inv
```

will tell you how to use the *inv* built-in function that gives the inverse of a matrix. To get on-line help through a html window environment, type

```
>> helpwin
```

2.5. Command line editing.

MATLAB allows command line editing. That is, you can use arrow keys at the MATLAB prompt to correct mistyped commands, or to recall previous commands. The following is a summary of the line editing commands:

Up Arrow, Ctrl-P	recall previous line
Down Arrow, Ctrl-N	recall next line
Left Arrow, Ctrl-B	move left one character
Right Arrow, Ctrl-F	move right one character
Back Space, Ctrl-D	delete character at cursor
Delete	delete character before the cursor
Ctrl-L	move left one word
Ctrl-R	move right one word
Ctrl-A	move to beginning of line
Ctrl-E	move to end of line
Ctrl-K	delete to end of current line
Ctrl-U	cancel current line
Ctrl-T	toggle insert and overtype mode

3. MATLAB statements.

A MATLAB statement is of the form

variable = expression

The above statement assigns the result of *expression* to the *variable*. **Variables' names are case sensitive in MATLAB.** A MATLAB statement can also be simply

expression

in which case the result is assigned to the default variable **ans**. ans is a special variable in MATLAB. It saves the result of the last execution of a MATLAB command when that result is not explicitly saved in a user-specified variable. Finally, a MATLAB statement can also be simply

variable

in which case, if the *variable* was previously assigned a value, no assignment or calculation takes place, otherwise a warning about an undefined variable is displayed.

MATLAB statements are executed when they are followed by a carriage return (newline). By default, MATLAB statements followed by a carriage return (newline) echo their result(s) on the standard output. **If you wish to suppress automatic echoing, follow the MATLAB statement by a semicolon (;),** then by a carriage return.

Examples of MATLAB statements:

```
>> 100 + (32-17)*5 + 2^3
```

results in doing the calculation 100 + (32-17)*5 + 2^3 and outputing the result as follows

```
ans =
   183
```

while

```
>> a = 1.1
```

results in assigning the value 1.1 to variable `a` and echoing the result as follows:

```
a =
    1.1000
```

On the other hand,

```
>> b = 2;
```

results in assigning the value 2 to variable `b` without echoing and

```
>> b
```

results in outputing the value of variable `b` as previously assigned.

```
b =
     2
```

To clear all variables from the workspace, type

```
>> clear
```

To clear a particular variable, say `var`, type

```
>> clear var
```

In MATLAB, whatever follows the percent mark (%) in the same line is considered a comment.

When a MATLAB statement is long to fit on one line, it can be split in two (or more) lines using the ... continuation mark in each but the last line of the statement.

```
>> % this is a comment
>> % the following is an example of a long
>> % statement split in two lines
>> avariablewithalongname = 100 + (32-17)*5 ...
                          + 2^3 - log(10)/log(2)
avariablewithalongname =
  179.6781
```

4. Relational and logical operators in MATLAB.

The following are the relational and logical operators of MATLAB:

Relational Operator	Meaning
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
~=	not equal
Logical Operator	**Meaning**
&	AND
|	OR
~	NOT
0	FALSE
non-zero number	TRUE

5. Working with matrices in MATLAB.

MATLAB requires neither the dimension of matrices nor the type of their entries to be specified. The simplest way to declare a matrix is to directly list its entries (row-by-row) enclosed in square brackets and assign the result to a variable:

```
>> A = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]
```

will result in the output

```
A =
     1     2     3
     4     5     6
     7     8     9
```

Matrix entries in the same row are separated by one or more blank spaces (or by a comma), while rows are separated by a semicolon. If the matrix is large, we can put each row in one line. That is,

```
>> A = [ 1 2 3
         4 5 6
         7 8 9 ]
```

will give us the same result, as above.

1-indexing

To refer to a matrix entry that has been assigned earlier, type the matrix variable name followed by `(i, j)`, where `i` is the row index and `j` is the column index of the entry. For example, assuming the previous declaration of the matrix `A`,

```
>> A(2, 3)
```

results in

```
ans =
     6
```

The dimension of matrices in MATLAB is set dynamically. It can be changed on demand. Executing

```
>> A(4, 3) = 10
```

will give

```
A =
     1     2     3
     4     5     6
     7     8     9
     0     0    10
```

The size of the matrix is changed automatically to accommodate the new element, and the old values are kept. Any undefined elements are set to 0. The built-in function *size* gives the size of a matrix. The answer is given in the form `[m, n]`, where `m` is the number of rows and `n` is the number of columns. The result can be stored with a multiple assignment

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

where `A` is an already defined matrix.

In MATLAB, it is easy to extract parts of the matrix.

```
>> A(i:j, k:l)
```

gives the submatrix of `A` defined by rows `i` through `j` and columns `k` through `l` of matrix `A`. If the index is just a colon (`:`), without numbers that define a range, all rows or all columns of the matrix are included. For example,

```
>> A(:, k:l)
```

gives the submatrix of `A` defined by columns `k` through `l` of all rows of matrix `A`.

In general, the colon can be used to generate a sequence of numbers forming a vector (or one-dimensional matrix).

```
>> m:k:n
```

range

gives a list of numbers, starting from `m` and ending to `n` with step `k` and

```
>> m:n
```

inclusive on both sides [m, n]

gives a list of numbers, starting from `m` and ending to `n` with step 1.

Matrix concatenation can also be easily done. If `A` and `B` are two matrices of the same number of rows,

```
>> C = [A B]
```

creates a matrix `C`, with the same number of rows as `A` and `B`. The number of columns of `C` is the sum of the number of columns of `A` and `B`, with the columns of `A` followed by the columns of `B`. On the other hand, if `A` and `B` are two matrices of the same number of columns,

```
C = [A; B]
```

creates a matrix `C`, with the same number of columns as `A` and `B`. The number of rows of `C` is the sum of the number of rows of `A` and `B`, with the rows of `A` followed by the rows of `B`.

Some basic operations on matrices:

A'	transpose of A
inv(A)	inverse of matrix A
A + B	matrix addition
A - B	matrix subtraction
A * B	matrix multiplication
A \ B	left multiplication of B by inverse of A if A is not singular. i.e. same as inv(A)*B
B / A	right multiplication of B by inverse of A if A is not singular. i.e. same as B*inv(A)
A .* B	element-wise matrix multiplication
A .\ B	element-wise division (right divided by left)
B ./ A	element-wise division (left divided by right)

6. Saving output and plots from MATLAB.

Typing

```
>> diary outfile
```

causes a copy of all subsequent terminal input and the resulting output, except plots, to be written to the file `outfile`. If the file already exists, the result is appended to the file. Typing

```
>> diary off
```

suspends the above copying to the file.

To produce a hard copy of the graphs, use

```
>> print plotfile.ps
```

after each statement that does a plot. This creates (**overwrites**) the postscript file `plotfile.ps`. Graphs are accumulated if the file `plotfile.ps`, if you use

```
>> print -append plotfile.ps
```

To get the graphs printed on paper, type

```
% lpr plotfile.ps
```

at the Unix shell prompt.

7. Simple examples.

```
>> % matrix multiplication
>> A = [1 0 1; 0 1 1; 1 1 1]
A =
     1     0     1
     0     1     1
     1     1     1

>> B = [1 2; 3 4; 5 6]
B =
     1     2
     3     4
     5     6
>> A*B
ans =
     6     8
     8    10
     9    12
>> % save the result in C
>> C = A*B
C =
     6     8
     8    10
     9    12

>> % see again the contents of A
>> A
A =
     1     0     1
     0     1     1
     1     1     1
>> % find the determinant of A
>> det(A)
ans =
    -1
>> % find the inverse of A
>> inv(A)
ans =
     0    -1     1
    -1     0     1
     1     1    -1
>> % find the eigenvalues of A
>> eig(A)
ans =
    1.0000
    2.4142
   -0.4142
>> % Find the eigenvectors/values of A
>> % and save them in ec/el, respectively.
>> % Each column of ec is an eigenvector of A.
>> % The diagonal entries of el are the
>> % eigenvalues of A.
>> [ec, el] = eig(A)
ec =
   -0.7071    0.5000   -0.5000
    0.7071    0.5000   -0.5000
    0.0000    0.7071    0.7071
el =
    1.0000         0         0
         0    2.4142         0
         0         0   -0.4142

>> % see the transpose of C
>> C'
ans =
     6     8     9
     8    10    12

>> % matrix multiplications with wrong dimensions
>> V = [1; 2; 3; 4]
V =
     1
     2
     3
     4
>> A*V
??? Error using ==> *
Inner matrix dimensions must agree.

>> % solving a linear system Ax = b
>> A = [1 2 3; 4 5 0; 7 8 9]
A =
     1     2     3
     4     5     0
     7     8     9
>> b = [5; -4; 11]
b =
     5
    -4
    11
>> x = A\b  % note the backslash!
x =
    -1
     0
     2

>> % concatenation of matrices
>> % append columns
>> D = [A B]
D =
     1     2     3     1     2
     4     5     0     3     4
     7     8     9     5     6
>> % append rows
>> E = [A; B']
E =
     1     2     3
     4     5     0
     7     8     9
     1     3     5
     2     4     6

>> % a simple plot -- try it on a workstation
>> % define two vectors (one-dimensional matrices)
```



```
>> % and plot one versus the other
>> x = [1 2 3 4 5];
>> y = [10 100 1000 10000 1e5];
>> plot(x, y)
>> % try logarithmic with respect to y scale
>> semilogy(x, y)

>> % An example of a function
>> % Suppose we have the following function
>> % saved in the file addmat.m:
function [C, m, n] = addmat(A, B)
% Add matrices A and B and return
% the sum C and its dimensions.
% This block comment will be displayed
% if the user types ''help addmat''.

% first check if the dimensions match
if (any(size(A) ~= size(B)))
    error('matrices'' dimensions don''t agree')
end
[m, n] = size(A);
% do the addition using a traditional for-loop
for i = 1:m
    for j = 1:n
        C(i, j) = A(i, j) + B(i, j);
    end
end
% this is the end of file addmat.m

>> % An example of using the function addmat
>> % Define two matrices of same dimensions
>> % A is the identity matrix of size 3
>> A = eye(3)
A =
     1     0     0
     0     1     0
     0     0     1
>> % B is a 3x3 matrix of 4's
>> B = 4*ones(3, 3)
B =
     4     4     4
     4     4     4
     4     4     4
>> % compute C = A + B
>> [C, m, n] = addmat(A, B)
C =
     5     4     4
     4     5     4
     4     4     5
m =
     3
n =
     3

>> % the same result is obtained using MATLAB's
>> % high-level programming constructs and
>> % built-in functions
>> C = A + B
C =
     5     4     4
     4     5     4
     4     4     5
>> [m, n] = size(A)
m =
     3
n =
     3
```

8. Sparse matrices

Several applications result in large sparse matrices. Storage and manipulation of such matrices in dense form is at best inefficient and at worst impossible. MATLAB provides sparse storage and special functions for such matrix computations. For example, the following MATLAB code

```
>> n = 10;
>> e = ones(n^2, 1);
>> S = spdiags([-e, -e, 4*e, -e, -e], ...
                [-n, -1, 0, 1, n], n^2, n^2);
```

generates a sparse matrix `S`, of size $n^2 \times n^2$, with 5 nonzero bands, the main diagonal (all 4s), the subdiagonal, the superdiagonal, and another 2 diagonals at distance `n` from the main diagonal (all 1s). With

```
>> spy(S)
```

we can visualise the sparsity pattern of the matrix `S` (on a workstation). Try

```
>> help sparfun
```

to find out all the built-in functions related to sparse matrices.

9. More examples.

```
>> % diag is a very useful function
>> % diag(A), where A is a matrix,
>> % returns the main diagonal of A
>> C
C =
     6     8
     8    10
     9    12
>> diag(C)
ans =
     6
    10

>> % diag(A, n), where A is a matrix,
>> % returns the nth superdiagonal of A, if n > 0,
>> % and the |n|th subdiagonal of A, if n < 0
>> diag(C, 1)
ans =
     8
>> diag(C, -1)
ans =
     8
    12
```



```
>> % diag(V), where V is a vector,
>> % returns a diagonal matrix with V on its
>> % main diagonal
>> V = 1:4
V =
     1     2     3     4
>> diag(V)
ans =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4

>> % diag(V, n), where V is a vector,
>> % returns a diagonal matrix with V on its
>> % nth superdiagonal, if n > 0,
>> % and on its |n|th subdiagonal, if n < 0.
>> diag(V, 1)
ans =
     0     1     0     0     0
     0     0     2     0     0
     0     0     0     3     0
     0     0     0     0     4
     0     0     0     0     0
>> diag(V, -1)
ans =
     0     0     0     0     0
     1     0     0     0     0
     0     2     0     0     0
     0     0     3     0     0
     0     0     0     4     0

>> % more advanced examples
>> n = 5;
>> 3*eye(n) - diag(ones(n-1, 1), -1) ...
 + 2*diag(ones(n-1, 1), +1)
ans =
     3     2     0     0     0
    -1     3     2     0     0
     0    -1     3     2     0
     0     0    -1     3     2
     0     0     0    -1     3
>> 3*eye(n) - diag(diag(ones(n-1)), -1) ...
 + 2*diag(diag(ones(n-1)), +1)
ans =
     3     2     0     0     0
    -1     3     2     0     0
     0    -1     3     2     0
     0     0    -1     3     2
     0     0     0    -1     3

>> % three different ways of defining
>> % the same vector
>> % first, using colon
>> v = 1:n
v =
     1     2     3     4     5
>> % second, using concatenation
>> v = [];
>> for j = 1:n
       v = [v j];
   end
>> v
v =
     1     2     3     4     5
>> % third, using a traditional for-loop
>> clear v
>> for j = 1:n
       v(j) = j;
   end
>> v
v =
     1     2     3     4     5
>> % Among the above 3 ways of defining a vector
>> % the first (colon) is the fastest and neatest.

>> % a component-wise vector operation
>> w = 10.^v
w =
       10      100     1000    10000   100000
>> % another vector
>> u = 1:2:2*n
u =
     1     3     5     7     9

>> % two vectors plotted versus the same vector
>> % note the diacritical marks to distinguish
>> % the two lines plotted
>> semilogy(v, w, '+', v, u, 'o')
>> % note the different types of lines and colors
>> semilogy(v, w, 'y-', v, u, 'c--')
>> % diacritical marks, lines and colors altogether
>> % semilogy(v, w, '+', v, u, 'o', ...
              v, w, 'y-', v, u, 'c--')
```

10. For more information.

The file `/u/ccc/matlab.primer.ps` on the CDF machines is a brief but comprehensive introduction to MATLAB, based on a previous version of MATLAB, version 4. Although there are differences between versions 4 and 6, it is still a useful source of information. You can view the file by the command

```
% ghostview /u/ccc/matlab.primer.ps
```

given at the Unix shell prompt on any Xwindow workstation.

You may also wish to take a look at the main MATLAB web-page, the MathWorks page, at `http://www.mathworks.com`, for books, software, news, newsgroups and on-line documentation about MATLAB.

Please note that "The Student Edition of MATLAB", which runs on personal computers, has some limitations which the CDF version does not have.

