## Matlab Introduction for Incoming Economic Graduate Students

## Math Camp 2009, Brown University

## Text in Times New Roman Font is taken from "A MATLAB Primer" by Paul L. Fackler, Copyright, 1998, Paul L. Fackler, North Carolina State University.

**THE BASICS**

MATLAB is a programming language and a computing environment that uses matrices as one of its basic data types. It is a commercial product developed and distributed by MathWorks. Because it is a high level language for numerical analysis, numerical code to be written very compactly. For example, suppose you have defined two matrices (more on how to do that presently) that you call A and B and you want to multiply them together to form a new matrix C. This is done with the code

**C=A*B;**

(note that expressions generally terminate with a semicolon in MATLAB). In addition to multiplication, most standard matrix operations are coded in the natural way for anyone trained in basic matrix algebra. Thus the following can be used

**A+B**
**A-B**
**A'** for the transpose of A
**inv(A)** for the inverse of A
**det(A)** for determinant of A
**diag(A)** for a vector equal to the diagonal elements of A

With the exception of transposition all of these must be used with appropriate sized matrices, e.g., square matrices to **inv** and **det** and conformable matrices for arithmetic operations.

In addition, standard mathematical operators and functions are defined that operate on each element of a matrix. For example, suppose A is defined as the 2x1 matrix

[2 3]

then **A.^2** (.^ is the exponentiation operator) yields

[4 9]

(not **A*A** which is not defined for non-square matrices). Functions that operate on each element include

**exp**, **ln**, **sqrt**, **cos**, **sin**, **tan**, **arccos**, **arcsin**, **arctan**, and **abs**.

In addition to these standard mathematical functions there are a number of less standard but useful functions such as cumulative distribution functions for the normal: **cdfn** (in the STATS toolbox). The constant **pi** is also available.

MATLAB has a large number of built-in functions, far more than can be discussed here. As you explore the capabilities of MATLAB a useful tool is MATLAB's help documentation. Try typing **helpwin** at the command prompt; this will open a graphical interface window that will let you explore the various type of functions available. You can also type **help** or **helpwin** followed by a specific command or function name at the command prompt to get help on a specific topic. Be aware that MATLAB can only find a function if it is either a built-in function or is in a file

that is located in a directory specified by the MATLAB path. If you get a **function or variable not found** message, you should check the MATLAB path (using **path** to see if the functions directory is included) or use the command **addpath** to add a directory to the MATLAB path. Also be aware that files with the same name can cause problems. If the MATLAB path has two directories with files called TIPTOP.m, and you try to use the function TIPTOP, you may not get the function you want. You can determine which is being used with the **which** command, e.g., **which tiptop**, and the full path to the file where the function is contained will be displayed.

A few other built in functions or operators are extremely useful, especially

**index=start:increment:end;**

creates a row vector of evenly spaced values. For example,

**i=1:1:10;**

creates the vector [1 2 3 4 5 6 7 8 9 10]. It is important to keep track of the dimensions of matrices; the **size** function does this. For example, if A is 3x2,

**size(A,1)**

returns a 3 and

**size(A,2)**

returns a 2. The second argument of the **size** function is the dimension: the first dimension of a matrix is the rows, the second is the columns. If the dimension is left out a 1x2 vector is returned:

**size(A)**

returns [3 2].

There are a number of ways to create matrices. One is by enumeration

**X=[1 5;2 1];**

which defines X to be the 2x2 matrix

1 5
2 1

The **;** indicates the end of a row (actually it is a concatenation operator that allow you to stack matrices; more on that below). Other ways to create matrices include

**X=ones(m,n);**

and

**X=zeros(m,n);**

which create mxn matrices with each element equal to 1 or 0, respectively. MATLAB also has several random number generators with a similar syntax.

**X=rand(m,n);**

creates an mxn matrix of independent random draws from a uniform distribution (actually they are pseudo-random).

**X=randn(m,n);**

draws from the standard normal distribution.

Individual elements of a matrix the size of which has been defined can be accessed using **()**; for example if you have defined the 3x2 matrix B, you can set element 1,2 equal to cos(2.5) with the statement

**B(1,2)=cos(5.5);**

If you then what to set element 2,1 to the same value use

**B[2,1]=B[1,2];**

A whole column or row of a matrix can be referenced as well in the following way

**B(:,1);**

refers to column 1 of the matrix B and

**B(3,:);**

refers to its third row. The **:** is an operator that selects all of the elements in the row or column. An equivalent expression is

**B(3,1:end);**

where **end** indicates the column in the matrix.

You can also pick and choose the elements you want, e.g.,

**C=B([1 3],2);**

results in a new 2x1 matrix equal to

$B_{12}$

$B_{32}$

Also the construction

**B(1:3,2);**

is used to refer to rows 1 through 3 and column 2 of the matrix B. The ability to access parts of a matrix is very useful but also can cause problems. One of the most common programming errors is attempting to access elements of a matrix that don't exist; this will cause an error message.

While on the subject on indexing elements of a matrix, you should know that MATLAB actually has two different ways of indexing. One is to use the row and column indices, as above, the other to use the location in the vectorized matrix. When you vectorize a matrix you stack its columns on top of each other. So a 3x2 matrix becomes a 6x1 vector composed of a stack of two 3x1 vectors. Element 1,2 of the matrix is element 4 of the vectorized matrix. If you want to create a vectorized matrix the command

**X(:)**

will do the trick.

**Example:** The Matlab functions det(), inv(),rank(),and rref() (reduced row echelon form) are useful in solving linear systems.

**Exercise:** Find the solution(s) to the linear system (1, 2, 3)' = ( 1 2; 3 4; 5 6)(x_1 x_2) using the rref command.

**Eigenvalues & eigenvectors:** Tomorrow we will learn about the eigenvalues & eigenvectors of a matrix. Today we can already explore these commands in Matlab.

An eigenvalue is a number λ such that $Ax = \lambda x$ for some vector $x \neq 0$. Then x is called an eigenvector associated with the eigenvalue λ.

**eig(A)** returns a vector with the eigenvalues of A.

**[V, D] =eig(A)** returns a matrix V of the eigenvectors and a vector D of the eigenvalues of A.

Compute A*V(:,1) to check!

### Plotting functions

MATLAB has a powerful set of graphics routines that enable you to visualize your data and models. For starters, it will suffice to note that routines **plot**, **mesh** and **contour**. For plotting in two dimensions, use **plot(x,y)**. Passing a string as a third argument gives you control over the color of the plot and the type of line or symbol used. **mesh(x,y,z)** provides plots of a 3-D surface, whereas **contour(x,y,z)** projects a 3-d surface onto two dimensions. It is easy to add titles, labels and text to the plots using **title**, **xlabel**, **ylabel** and **text**. Subscripts, superscripts and Greek letters can be obtained using TEX commands (eg., x_t, x^2 and \alpha\mu). To gain mastery over graphics takes some time; the documentation Using MATLAB Graphics available with MATLAB is as good a place as any to learn more.

**Example:** Plot e^(-x^2) for x from -1 to 1. Define x = -1:0.1:1 and y = exp(-x.^2). Use plot(x,y).

You may have noticed that statements sometimes end with **;** (semi-colon) and they don't. MATLAB is an interactive environment, meaning it interacts with you as it runs jobs. It communicates things to you via your display terminal. Any time MATLAB executes an assignment statement, meaning that is assigns new values to variables, it will display the variable on the screen UNLESS the assignment statement end with a semi-colon. It will also tell you the name of the variable, so the command

```
x=2+4
```

will display

```
x =
    6
```

on your screen, whereas the command

```
x=2+4;
```

displays nothing. If you ask MATLAB to make some computation but do not assign the result to a variable, MATLAB will assign it to an implicit variable called **ans** (short for ``answer''). Thus the command

```
2+4
```

will display

```
ans =
    6
```

Next Previous Top

### CONDITIONAL STATEMENTS AND LOOPING

```
if expression
...
end
if expression
...
else
```

...
**end**

and

**while** expression
...
**end**

The first two of these are single conditionals, for example

**if X>0, A=1/X; else A=0, end**

You should also be aware of the **switch** command (type **help switch**).

The last is for looping. Usually you use **while** for looping when you don't know how many times the loop is to be executed and use a **for** loop when you know how many times it will be executed. To loop through a procedure n times for example, one could use the following code:

**for i=1:n, X(I)=3*X(i-1)+1; end**

A common use of **while** for our purposes will be to iterate until some convergence criteria is met, such as

```
P=2.537;
X=0.5;
DX=0.5;
while DX<1E-7;
 DX=DX/2;
 if normcdf(X)>P, X=X-DX; else X=X+DX; end
 disp(X)
end
```

(can you figure out what this code does?). One thing in this code fragment that has not yet been explained is **disp(X)**. This will write the matrix X to the screen.

Next Previous Top

## SCRIPTS AND FUNCTIONS

When you work in MATLAB you are working in an interactive environment that stores the variables you have defined and allows you to manipulate them throughout a session. You do have the ability to save groups of commands in files that can be executed many times. Actually MATLAB has two kinds of command files, called M-files. The first is a script M-file. If you save a bunch of commands in a script file called MYFILE.m and then type the word MYFILE at the MATLAB command line, the commands in that file will be executed just as if you had run them each from the MATLAB command prompt (assuming MATLAB can find where you saved the file). A good way to work with MATLAB is to use it interactively, and then edit you session and save the edited commands to a script file. You can save the session either by cutting and pasting or by turning on the **diary** feature (use the on-line help to see how this works by typing **help diary**).

The second type of M-files is the function file. One of the most important aspects of MATLAB is the ability to write your own functions, which can then be used and reused just like intrinsic MATLAB functions. A function file is a file with an m extension (e.g., MYFUNC.m) that begins with the word **function**.

**function Z=DiagReplace(X,v)**

```
% DiagReplace Replace the diagonal elements of a matrix X with a vector v
% SYNTAX:
% Z=DiagReplace(X,v);
n=size(X,1);
Z=X;
ind=(1:n:n*n) + (0:n-1);
Z(ind)=v;
```

You can see how this function works by typing the following code at the MATLAB command line:

```
m=3; x=randn(m,m);v=rand(m,1); x,v,xv=diagreplace(x,v)
```

Any variables that are defined by the function that are not returned by the function are lost after the function has finished executing (**n** and **ind** in the example). Here is another example:

```
function x = rndint(k,m,n)
% RANDINT Returns an mxn matrix of random integers between 1 and k (inclusive).
% SYNTAX:
%   x= rndint(k,m,n);
% Can be used for sampling with replacement.
x=ceil(k*rand(m,n));
```

Documentation of functions (and scripts) is very important. In M-files a **%** denotes that the rest of the line is a comment. Comment should be used liberally to help you and others who might read your code understand what the code is intending to do. The top lines of code in a function file are especially important. It is here where you should describe what the function does, what its syntax is and what each of the input and output variables are. These top line become an online help feature for your function. For example, typing **help randint** at the MATLAB command line would display the four commented lines on your screen.

A note of caution on naming files is in order. It is very easy to get unexpected results if you give the same name to different functions, or if you give a name that is already used by MATLAB. Prior to saving a function that you write, it is useful to use the **which** command to see if the name is already in use.

An important feature of MATLAB is the ability to pass a function to another function. For example, suppose that you want to find the value that maximizes a particular function, say $f(x)= x*\exp(-0.5x^2)$. It would useful not to have to write the optimization code every time you need to solve a maximization problem. Instead, it would be better to have solver that handles optimization problems for arbitrary functions and to pass the specific function of interest to the solver. For example, suppose we save the following code as a MATLAB function file called MYFUNC.m

```
function fx=myfunc(x)
fx=x.*exp(-0.5*x.^2);
```

Furthermore suppose we have another function call MAXIMIZE.m which has the following calling syntax

```
function x=MAXIMIZE(f,x0)
```

The two arguments are the name of the function to be maximized and a starting value where the function will begin its search (this is the way many optimization routines work). One could then

call MAXIMIZE using

**x=maximize('myfunc',0)**

and, if the MAXIMIZE function knows what it's doing, it will return the value 1. Notice that the word **myfunc** is enclosed in single quotes. It is the name of the function, passed as a string variable, that is passed in. The function MAXIMIZE can evaluate MYFUNC using the **feval** command. For example, the code

**fx=feval(f,x)**

is used to evaluate the function. It is important to understand that the first argument to **feval** is a string variable (you may also want to find out about the command **eval**, but this is only a primer, not a manual).

**Exercise:** Write a function that takes a matrix as input and checks for definiteness.