An Introduction to Matlab5

Phil Spector
Statistical Computing Facility
University of California, Berkeley

August 21, 2006

1 Background

Matlab was originally developed as a simple interface to the LINPACK and EISPACK subroutine libraries which have long been the standard for linear algebra and eigenvalue problems. Some of the original authors of these packages were involved in the development of matlab. Accordingly , matlab has great strengths in these fields, and, in general it is a very useful package for anything having to do with matrices. The matlab programming language has a FORTRAN-like flavor without requiring the user to worry about tasks like dimensioning matrices and formatting printed output. A variety of optional toolboxes expands matlab's capabilities into fields like signal processing, optimization and simulation. Matlab also provides a very flexible facility for producing graphics and constructing easy-to-use, point- and-click interfaces. Some of the key strengths of matlab include:

- excellent implementation of linear algebra routines
- suite of functions for working with sparse matrix
- flexible and well-designed graphics functions
- easy-to-learn, FORTRAN-like command language
- fairly easy to add new routines with C and/or FORTRAN
- well documented external interface for accessing matlab objects
- good documentation in general, including a hypertext user's guide

2 Basics

Matlab is oriented towards two-dimensional numerical matrices, but also has the capability to work with multi-dimensional arrays and a variety of structures containing character values. Variable names can be constructed from letters, digits and underscores, but remember that matlab is case sensistive. Matlab will accept variable names of any length, but will silently truncate them to 31 characters. You can display any matlab object which is currently active by typing its name, or using the `disp` command. You can display the names of currently active objects using the `who` command, and see their dimensions with the `size` function; the `whos` command will list all the active objects along with their sizes. To include comments in a matlab program, begin the line with a per cent sign (`%`). To type a command which is longer than one line, terminate each line which needs to be continued with three periods (`...`). To type several commands on the same line (with no carriage return) separate the commands with commas (`,`) or semicolons (`;`). You can examine and edit previous commands using arrow keys or by using control-P for previous command,

control-N for the next command, control-F for forward and control-B for backward. The results of your matlab statements will be displayed to the screen, unless you end your statements with a semicolon (`;`). If you forget to assign the output of a command to a variable, it will be temporarily available in the matrix named `ans`.

Many matlab commands automatically operate on each column of a matrix; for example the matlab command `mean(x)` will return a vector with as many elements as there are columns in the matrix x representing the means of each column. To get the overall mean of the matrix, you could use `mean(mean(x))`.

Some matlab functions can potentially return more than one result. For example, the `eig` command can return both eigenvalues and eigenvectors of a matrix. To assign multiple returned values to variables, use square brackets, as in

```
[u,v] = eig(x)
```

will place the eigenvectors in `u`, and the eigenvectors in `v`. The behavior of functions like this when they are assigned to a single value should be checked in the help file. For example,

```
z  =  eig(x)
```

returns a vector containing the eigenvalues of `x`.

There is excellent online help available; for any command type `help` *`commandname`*; in addition the command `helpdesk` will open a web browser on a hypertext manual, and the command `doc` *`topic`* will open a web browser on the documentation for *topic*. If you're not sure of the name of the command you want, you can provide keywords to the `lookfor` command to see some reasonable choices.

You can type a operating system command from within matlab by typing an exclamation point (`!`) as the first character on the line.

If you have a file of matlab commands whose file name ends in the extension `.m` in either the current directory , or a subdirectory in your home directory called matlab, you can execute the file by typing its name without the `.m` extension. The matlab interpreter will also find functions which are defined this way . Such files are known as M-files . You can view the contents of an M-file, including system provided M-files, with the `type` command. See Section 6 for more information.

matlab has a very handy debugger, which can be used to step through M-files. Type `help debug` from within matlab for more information.

Optional programs in matlab are organized into groups called toolboxes. You can see what toolboxes are available in a particular matlab installation by typing `help`; information about specific toolboxes can be obtained through a command like `help toolbox/optim`, which would display information about the optimization toolbox.

3 Specific Tasks

3.1 Entering and Exiting the Program

To start up matlab, use the command `matlab`. You will be in an interactive mode, with a prompt of "`>>`". You can read a file of matlab commands into matlab by using the usual UNIX redirection, or by creating an M-file (see Section 2).

To exit matlab, type `exit`.

3.2 Reading Data

For very small matrices, you can use the assignment statement directly . Surrounding values with square brackets (`[ ]`) informs matlab that you are creating a matrix; a semicolon (`;`) inside the brackets indicates a new row. Thus,

```
a = [1 3 4 5];
```

creates a vector of length four, while

```
x = [1 2 4 5; 2 4 5 3; 1 2 7 9];
```

creates a 3×4 matrix. Note that character arrays can contain only single character elements. To store character strings, use cell arrays or structures as explained in Section 4 below .

The matlab command `load` is used to read in either raw (ASCII) data or matlab save files (see Section 3.3) For raw data, matlab expects there to be at least one blank or tab between each value in the input file. The command

```
load filename.ext
```

creates a matlab matrix named `filename` with as many rows as there are lines in the file, and as many columns as there are white-space separated entries on each line. Note that the filename must appear without quotes. To load a file without an extension, use

```
load filename -ascii
```

Commands like `load` also have a function form, which allows you to use character strings (either stored in variables, or as a name surrounded by quotes) instead of as unquoted strings. This is especially useful if you need to interactively open files. The last example could be replaced by the function form of `load` as follows:

```
load('filename','-ascii')
```

To load matrices saved in a previous matlab session, use the command

```
load fname
```

This retrieves the variables from a file called `fname.mat`, typically created by the matlab `save` command.

3.3 Storing Data Sets

The matlab command `save` will store one or more matrices in a file with a `.mat` extension; these matrices can then be read using the load command (See Section 3.2) With no arguments, the `save` command saves all the active matrices into a file called `matlab.mat`; the command

```
save matrix
```

creates a file in the current directory called `matrix.mat` which will contain a single matrix, while the command

```
save filename mat1  mat2  ...
```

saves the named matrices in a file called `filename.mat`. In addition, matlab will write a ASCII repr esentation of a matrix through the `-ascii` flag of the save command. This can be useful if you wish to transport the results of a matlab session to some other program.

3.4 Accessing and Creating Variables

Everything in matlab behaves like a matrix, so subscripting and other matrix operations are the means for accessing and creating variables; there are no automatic labels to identify different variables (columns) within a matrix. Subscripts to matrices appear in parentheses after the matrix name; to refer to an entire row or column include a colon (`:`) in place of the missing dimension. Note that if you use a single subscript for a matlab matrix, it will treat the matrix as a vector by stacking together the columns of the matrix. You can explicitly change the shape of a matrix by using the `reshape` function. You can refer to a range of rows or columns by using the colon (`:`) as a sequence operator; for example `x(:,3:5)` refers to a

matrix containing three columns, namely the third, fourth and fifth columns of the matrix `x`. With three arguments (like `1:2:10`) , the middle argument (`2`) is used as an increment. You can also enter a list of indices surrounded by square brackets; `x([1 3 5],:)` represents the first, third and fifth rows of `x` . To add a row or column to a matrix, you can simply use square brackets; for example to add a new column to the matrix `a`, consisting of the logarithm of the third column of `a`, you could use

```
a  =  [a log(a(:,3))]
```

To add a new row, use a semicolon inside the square brackets to separate the original matrix and the row to be added.

3.5 Subsetting Data

Much of the subsetting required in matlab can be achieved by using the subscripting operations described in Section 3.4. In addition, logical expressions can be used as subscripts if they are the same length as the dimension they are representing. When used as a subscript, a value of 1 (true) includes the element and a value of (false) excludes it. For example, suppose the third column of a matrix called `data` contains the age of subjects in a study , and we would like to create a matrix `youngdata` containing only those cases where age is less than 20.

```
youngdata = data(data(:,3) < 20,:)
```

The matrix `youngdata` will consist of all the columns from the rows for which the third column of data was less than 20.

Since many functions in matlab automatically return a vector with values for each column of the matrix, some subsetting tasks are simplified. For example, given a matrix `x`, suppose we wanted to extract those columns for which the mean was greater than 20. We could simply say

```
x[:,mean(x) >  20]
```

Note that the expression `mean(x) > 20` evaluates to a logical vector whose length is equal to the number of columns in the matrix `x`. The transpose operator (`'`) can be used to perform a similar operation on the rows of a matrix:

```
x[mean(x') > 20,:]
```

When replacing part of a matrix, matlab will report an error if the dimensions of the replacement do not exactly match the dimensions of the data to be replaced. For example consider the 3×4 matrix `y`

```
y = 12    19    23    14
    15    23    19    20
    17    35    43    15
```

Suppose we wished to replace the third column of `y` with the integers 1 through 3. The matlab statement

```
y(:,3) = 1:3;
```

will result in an error, while the statement

```
y(:,3) = [1:3]';
```

will give the desired result, since `1:3` is considered a 1×3 matrix, and `y(:,3)` is considered a 3×1 matrix. An exception to this rule is the case where the value to be assigned is a scalar; in this case, matlab will repeat the scalar as necessary in order to fill in the right hand side of the assignment. Thus the statement

```
y(:,5) =  1;
```

will create a new column in y of the appropriate size, containing all ones.

4 Cell Arrays

To accomodate objects other than numbers, matlab provides two extensions to the concept of a matrix: cell arrays and structures. Cell arrays are indexed like matrices, but can contain character strings. Structures consist of named elements whose contents are arbitrary . For example, to construct a cell array containing the names of variables in a matrix, the following command could be used:

```
varnames = [{'id'} {'weight'} {'height'}];
```

The curly braces tell matlab to construct cells instead of the usual character arrays. Curly braces (`{}`) are used instead of parentheses to indicate that we are storing a cell, not a regular value. Thus, there are two equivalent ways to set the elements of a cell array:

```
varnames{1} =  'idnum';
varnames(1) = {'idnum'};
```

Both of the preceding statements change the first cell in varnames to the value "`idnum`".

Similarly , the expression

```
name  = varnames(1)
```

creates a cell array called `name` which contains a single element, namely the string "`idnum`", while

```
name  = varnames{1}
```

creates a 1×5 character matrix whose elements are "`i`","`d`","`n`","`u`", and "`m`".

Cell arrays can be arbitrarily complex. For example, to create a cell array in which the first column contains a name and the second column contains a matrix, you could use statements like the following:

```
cell{1,1} = 'fred';
cell{1,2} = [1 2 3; 4 5 6];
cell{2,1} = 'joe';
cell{2,2} =  [10  9  8;  7  6 5];
```

When you print such an array (using the function `disp` or by typing the object's name), matlab displays the following:

```
cell =
    'fred'    [2x3 double]
    'joe'     [2x3 double]
```

To see the full details of what each cell contains, you can use the matlab function `celldisp`.

You can extract several elements from a cell array at once by using a comma separated list on the left hand side of the assignment, and the `deal` function on the right hand side. For example, using the cell array cell produced in the previous example, we could use

```
[a,b]  = deal(cell{:,2})
```

to set `a` equal to `cell{1,2}` and `b` equal to `cell{2,2}`.

5 Structures

Structures are very much like cell arrays in that they can store several disparate items within a single object, but they access the elements by named fields rather than numbers. Structures can be created in a number of ways, and they automatically fill in missing elements with blanks or zeroes of the appropriate size.

To specify the name of a structure field, follow the name of the structure with a period, and then the field name. For example, to create a structure called `info` with a field called `id` and to set that field to the number 10, you could use

```
info.id = 10;
```

Usually structures will be indexed by a subscript to differentiate between field values for different units. In fact the previous statement actually created an array of structures for which the first would have an `id` of 10. To provide additional information, such as `income`, or `department`, we simply name the required fields:

```
info(1).id = 10;
info(1).income = 20000;
info(1).department = 'Sales';
```

Additional elements could be defined in the same way:

```
info(2).id = 11;
info(2).income = 23000;
info(2).department =  'Tech Support';
```

Alternatively , the `struct` function can be used to assign field values for several records at once:

```
info  =  struct('id',{10 11},'income',{20000 23000}, ...
                'department',{'Sales','Tech Support'})
```

Note the use of curly braces (`{}`) to provide values to the `struct` function - structure values passed to this function must be cell arrays, not regular arrays, or they will be treated as single values.

6 Missing Values

Missing values in matlab are represented by the symbol `NaN` (not a number) on input. (In fact, the case is not important; any combination of upper and lower case can be used.) To test for missing values the `isnan` function must be used; using the equality operator (==) will not work. You can, however use the symbol `NaN` as a missing value on the right hand side of an assignment statement.

7 Scripts and Functions

While matlab is designed for interactive use, there are times when it is easier to compose a file of matlab statements outside of matlab itself, and then have matlab execute the statements from the file. While this can be done with UNIX redirection, a sometimes more useful way is to construct what is known as an M-file. An M-file is simply a file whose name ends in the extension "`.m`", and which is located either in your current directory , or a subdirectory of your home directory called `matlab`. Ordinary matlab statements entered in a file in this manner will be executed by matlab just as if they were typed at the terminal; when the M-file is finished, the matlab pr ompt r eturns and you can continue your matlab session.

To execute the commands in an m-file, type the name of the file, not including the "`.m`". For example, suppose the following lines are stored in a file called `doplot.m`:

```
x  =  [1  2  3  4  5];
y  = [12 19 17 21 22];
plot(x,y,':')
```

Then typing doplot at the matlab prompt will execute the statements stored in `doplot.m` just as if they were individually typed in. A special kind of M-file, namely a function file, allows you to extend the capabilities of matlab through functions which you write yourself. The difference between an M-file script and an M-file function is that variables in a script are evaluated in the same workspace as statements you type to the interpreter, while with a function, the only communication between your current workspace

and the function is through variables passed to the function in the argument list. Creating a function M-file is easy. As a simple example, suppose we wish to scale the elements of a vector by dividing each of them by the maximum value encountered in the vector. We first create a file called `scale.m` containing the following statements:

```
function [new,themax]  =  scale(x)
% SCALE scale a vector by dividing each element by the maximum value
%    new  = scale(x) returns a vector new, like  x,  but with each element
%           divided  by  the maximum value  of  the original vector
%    [new,themax] = scale(x) also returns the maximum value  of  the original
%           vector in themax
themax = max(x);
new = x / themax;
```

This file would either be stored in the current directory, or in a subdirectory of your home directory called `matlab`. Then the function can be called with any vector . For example, to scale the numbers from 1 to 10, we could type

```
z = scale(1:10)
```

at the matlab prompt, and the value of z would now be the scaled version of the numbers from 1 to 10.

A few notes about m-files:

1. matlab recognizes an M-file as a function file by the appearance of the word function at the very beginning of the file. This is followed by a dummy variable (or variables enclosed in brackets) to indicate the values which the function will return.

2. The name of the function is determined by the name of the file in which it is stored, not by the internal function name. In general, making the file name and the function name the same is the best practice.

3. Comments can be placed in an M-file by making a percent sign (`%`) the first character of the line. The first set of lines after the function line which begin with a percent sign will be printed (without the percent sign) whenever help is requested for the function.

4. The M-file returns control to the regular matlab session whenever the last line in the m-file is executed, or whenever a return statement is encountered. Note that the variable values which are returned will be the current values of the dummy variables named on the function definition line at the time that the M-file returns.

8 Graphics

matlab provides a number of easy-to-use, high level graphics including `plot`, `hist`, `bar`, `stem`, and `contour` (for 2-dimensional plotting) and `plot3`, `contour3`, `mesh`, and `surf` (for 3-dimensional plotting). These routines are often sufficient to produce suitable graphics, but they can be easily modified with the functions `text`, which places text in a specified position on the plot or `gtext` which allows you to specify text placement using the mouse. In addition, matlab provides a "handle" graphics system, wherein each part of a graph has an identifier, known as a handle, which allows you to either list (using the `get` command) or modify (using the `set` command) any property of that part of the graph. Both the `get` and `set` functions take a handle as their first argument, and additional arguments consist of a succession of keywords (in single quotes) followed by the value specified for each keyword.

As an example, consider a simple scatterplot, produced using the `plot` command. To use the handle graphics system, set the value of a variable, in this case, `p` to the handle of the plot:

```
p  = plot(x,y)
```

Now suppose we wish to change the positions of the x-axis tick marks. The tick marks are actually part of the parent of the plot, so we need to get the parent's handle:

```
phand = get(p,'parent')
```

To see what keywords can be used to modify the parent graphic, pass the handle to the `get` routine with no other arguments:

```
get(phand)
```

Among the attributes displayed is one called XTick; to change it, you can use the `set` command:

```
set(phand,'XTick',[-3 3])
```

Only the tick values will change. By examining the output from the `get` command, you can find the keywords to change virtually any aspect of a plot.

In addition to plotting, matlab graphics also support the development of graphical user interfaces (GUIs) There are functions provided to create a wide array of GUIs to control matlab procedures. A function called `guide` provides a graphically-oriented method for creating GUIs. See the online documentation for more information.

9 Resources

9.1 Online Documentation

When using the Java-based front end to matlab, you can type the command

```
helpdesk
```

to access the Matlab help system.

9.2 Distributor's Web Page: `http://www.mathworks.com`

Extensive documentation is available at:

```
http://www.mathworks.com/access/helpdesk/help/helpdesk.html
```

9.3 Usenet Newsgroup

```
comp.soft-sys.matlab
```

10 Books

1. *Mastering Matlab 5*, by Duane Hanselman and Bruce Littlefield, Prentice-Hall, Upper Saddle River , 1998.

2. *The Matlab 5 Handbook*, by Darren Redfern and Colin Campbell, Springer , New York, 1997.