

MATLAB

Not a statistics package, however it is a useful tool for the statistician.

Its basis is as a matrix math package

However there are many addons (toolboxes). With the Harvard site licence, the toolboxes included with the Windows version (6.5 R13) are:

Control System, Image Processing, Optimization, Signal Processing, **Statistics**, Symbolic Math, Simulink (Simulates dynamical systems)

This is the same set for version (6.1 R12) on ice. The default version on ice is 5.3 R11 and has many more tool boxes available.

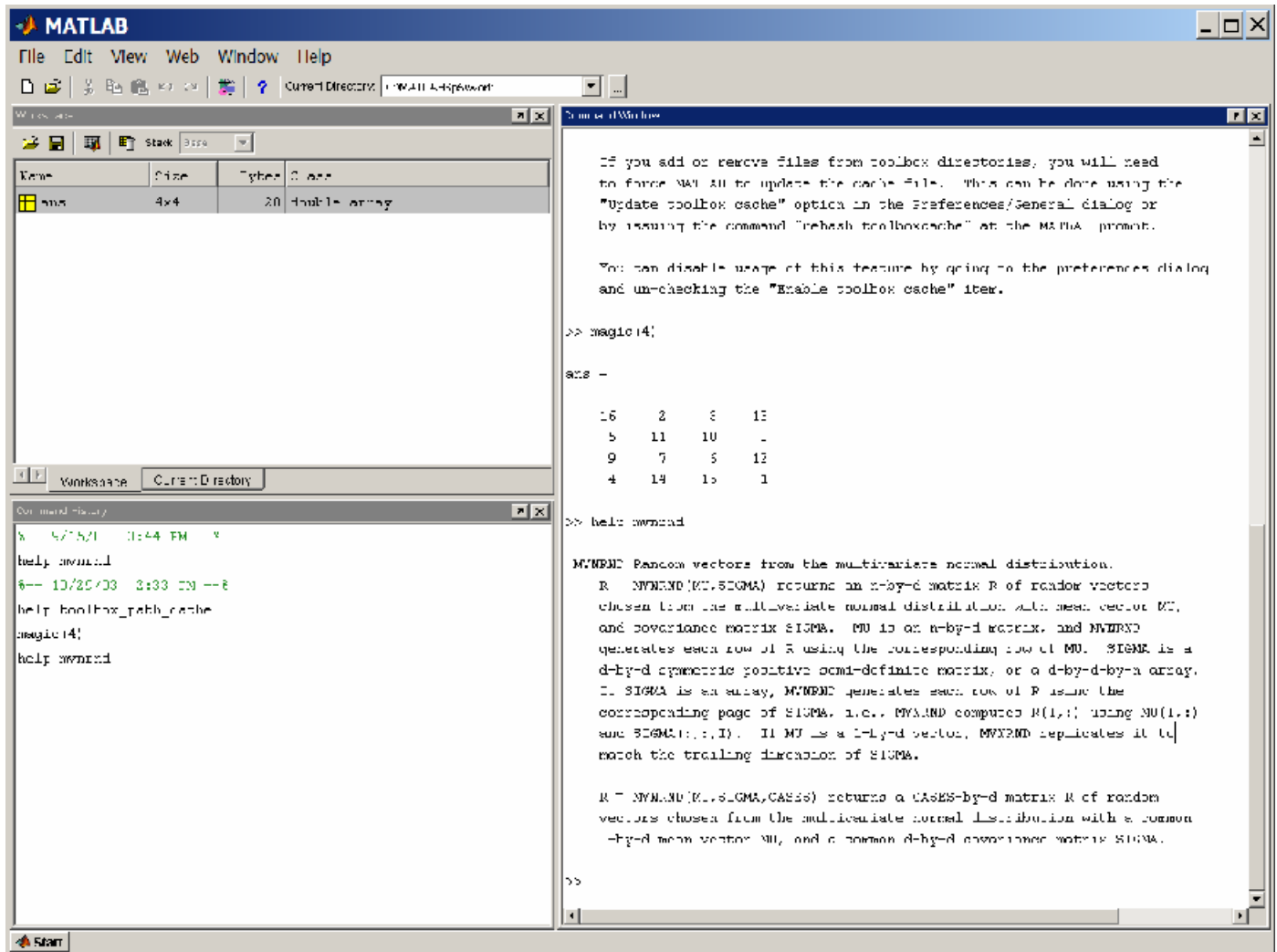
There are many more available from MathWorks (\$\$\$) plus free ones available on the net. These toolboxes are similar to what you get with `library()` in S-Plus (e.g. MASS, HMISC, trellis, etc).

I've found Matlab most useful for programming my own routines (great for MCMC) and graphics.

Its similar to S in that underlying the package is an objected-oriented, vectorized, programming language.

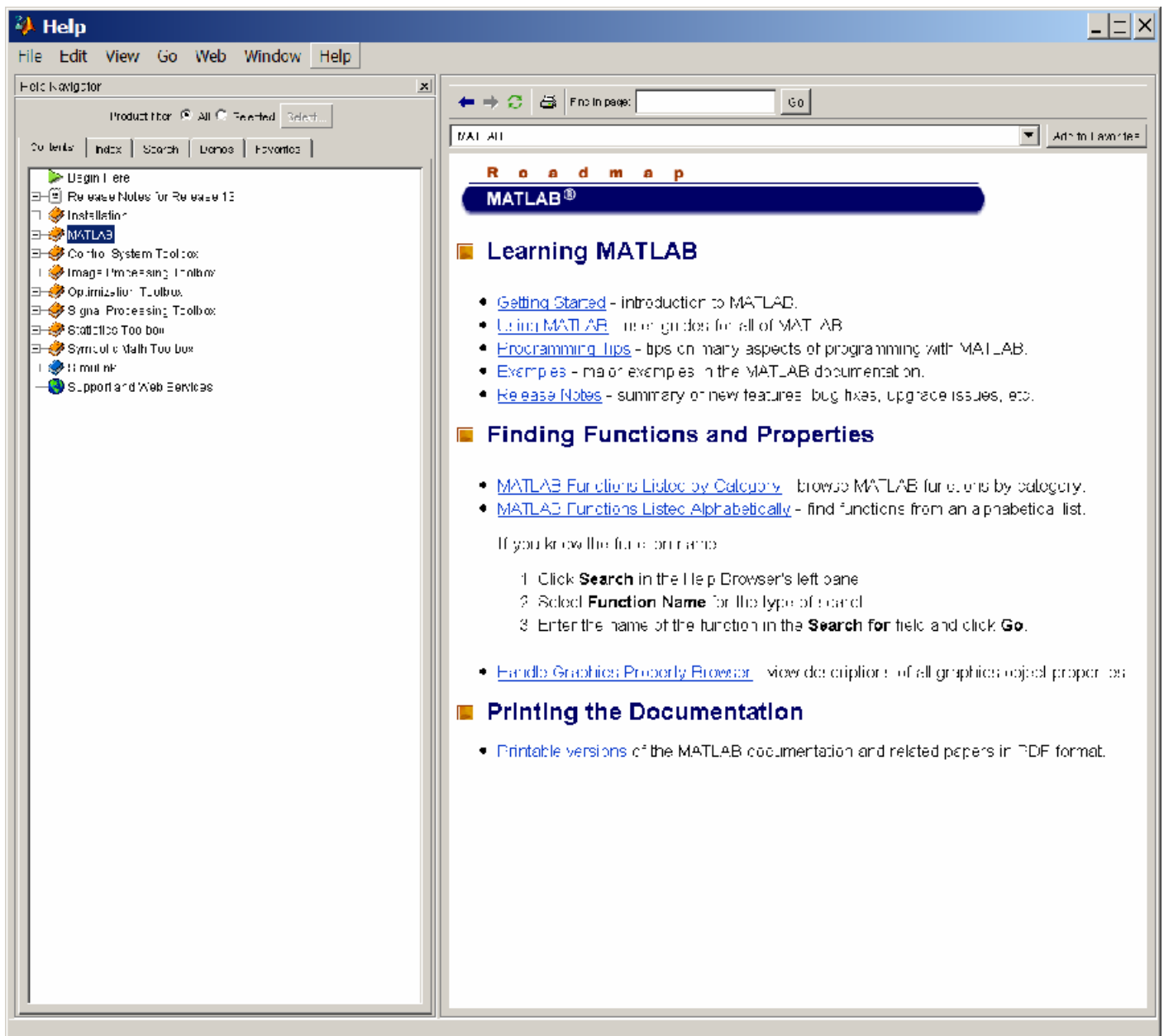
Starting up Matlab:

Same basic look in Windows, Unix/Linux, and Macintosh.



Default windows include Workspace (shows objects available in your current workspace), Command History (shows previous commands), and Command Window (type you commands and get output).

Also get the help system when starting up as the default. The help system is HTML based.



You can reorganize how you want your things setup (resize windows and what is showing) and the system remembers how you want things to look.

Matlab is similar to S-Plus/R in that you type commands in the Command Window and the output follows

One big difference between S-Plus/R and Matlab is that the basic data type is an array. However instead of dataframes and lists, Matlab uses structures (similar to C) and cell arrays.

One other thing that Matlab is good with is sparse matrix routines. A sparse matrix is one that is mainly zeros. If a matrix is recognized as being sparse, there can be great savings in memory and more efficient calculation for some problems.

Basic command structure

There are 3 basic command structures in Matlab

```
output = function(args)
```

```
function(args)
```

```
command args
```

The first one is when you want to store the output of some function into a Matlab object. The last two are for when there is no result from the function.

Note that for a number of commands, either of the following approaches can be used

```
function(args)
```

```
command args
```

For example

```
title 'plot title'
```

and

```
title('plot title')
```

will both give the current plot a title of 'plot title'. However the first form is limited in the number of options that can be given, where the second form is much more flexible. For example, if your plot title is in a variable `plot_title`, you would have to use

```
title(plot_title)
```

get what you want.

In Matlab, the output from a function may take more than one value. For example get the dimensions of matrix, the command

```
[r, c] = size(A);
```

will store the dimensions of the matrix A in r and c. Note that in Matlab, some of the functions can output their results different ways. The dimensions of matrix A can also be stored with

```
dims = size(A);
```

dims will be a vector of length 2. As with S-Plus/R, you can have argument lists change depending on which options you are interested in. The dimensions r and c can also be stored with

```
r = size(A, 1);
```

```
c = size(A, 2);
```

The semicolon at the end of the line is to suppress the printing of the output. For example

```
>> r = size(A, 1)
```

```
r =
```

```
4
```

```
>> r = size(A, 1);
```

>>

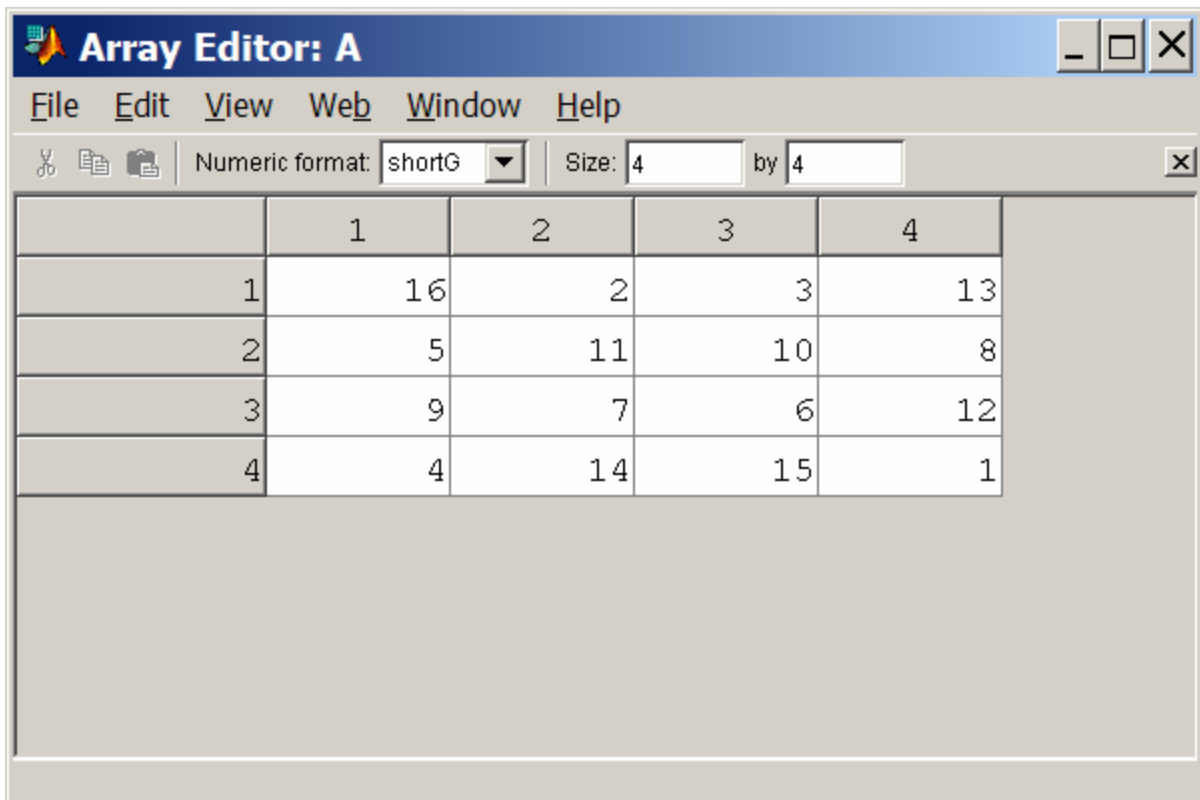
This is particularly desirable when writing your own scripts and functions as you don't want all the steps being printed out. However it can also be useful for debugging Matlab code.

Displaying objects – type the name

```
>> A
```

```
A =
```

```
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```



	1	2	3	4
1	16	2	3	13
2	5	11	10	8
3	9	7	6	12
4	4	14	15	1

Scripts and Logs

Similar to S-Plus/R you can write Matlab scripts to automate your jobs. The script can be created in any text editor and then run in Matlab. The name of your script must end in `.m` (e.g. `myscript.m`). To run the script in Matlab just type the name (without the `.m`) and hit return.

To store your Matlab session in a file, use the `diary` command. The form is

```
diary diaryfile
```

If `diaryfile` is omitted, the log is stored in the file `diary`. `diary on` and `diary off` can be used to toggle writing the output to the logfile.

To quit Matlab, type

```
quit
```

Creating arrays

```
>> a = [1 2 3]
```

```
a =
```

```
    1    2    3
```

```
>> b = [ 4 5 6 ]'
```

```
b =
```

```
    4
```

```
    5
```

```
    6
```

```
>> c = [a ; 8:10]
```

```
c =
```

```
    1    2    3
```

```
    8    9   10
```

```
>> d = [b (5:0.25:6)']  
??? Error using ==> horzcat  
All matrices on a row in the bracketed  
expression must have the same number of  
rows.
```

```
>> d = [b (9:-2:5)']
```

```
d =
```

```
     4     9  
     5     7  
     6     5
```

Standard arrays

```
>> ones(2)
```

```
ans =
```

```
     1     1  
     1     1
```

```
>> zeros(2,3)
```

```
ans =
```

```
     0     0     0  
     0     0     0
```

```
>> eye(2)
```

```
ans =
```

```
    1    0
    0    1
```

```
>> eye(2,4)
```

```
ans =
```

```
    1    0    0    0
    0    1    0    0
```

```
>> rand(size(c))
```

```
ans =
```

```
    0.4565    0.8214    0.6154
    0.0185    0.4447    0.7919
```

Reshaping arrays

```
A =
```

```
    1    4    7   10
    2    5    8   11
    3    6    9   12
```

```
B = reshape(A,2,6)
```

```
B =
```

```
    1     3     5     7     9    11
    2     4     6     8    10    12
```

```
B = reshape(A,2,[ ])
```

```
B =
```

```
    1     3     5     7     9    11
    2     4     6     8    10    12
```

```
>> reshape(1:12,3,2,2)
```

```
ans(:, :, 1) =
```

```
    1     4
    2     5
    3     6
```

```
ans(:, :, 2) =
```

```
    7    10
    8    11
    9    12
```

```
>> B = reshape(1:6,[3 1 2])
```

```
B(:, :, 1) =
```

```
1  
2  
3
```

```
B(:, :, 2) =
```

```
4  
5  
6
```

```
>> C = squeeze(B)
```

```
C =
```

```
1    4  
2    5  
3    6
```

Similar to S-Plus/R, you can deal with subarrays. However they are a couple of differences.

First Matlab uses `()`, not `[]` to indicate that you want to look a subarray. To look at the (1,2) entry of an array `A`, use `A(1,2)` (S: `A[1,2]`). The ordering of dimension in arrays is the same however: rows, columns, layers, etc.

Another difference deals with the situation when you deal with all the entries for one dimension. For example to deal with the first column of a matrix in S, you would do `A[,1]`. (Full dimensions are indicated by excluding it from the list). However in Matlab you need to indicate a full dimension with a `:`, giving `A(:,1)`.

The enteries for each dimension need to be vectors, with integers components in the range 1 to dimension size. Note you can have duplicate entries, such as

```
>> A([1 3 3 2],:)
```

```
ans =
```

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

In S-Plus/R, you cannot refer to entries outside the current array dimensions. For example if you have a 3x3 array named A, you can't do something like $A[4,4] \leftarrow 10$. However you can in Matlab. For example

```
>> A
```

```
A =
```

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

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

```
A =
```

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

Matlab will pad out the array with 0s to deal with the change in dimension. While this is useful, it can be inefficient, so you want to avoid this when you are programming in Matlab.

Reading in data

Data files (text) can be read in using the `load` command. It is expected that all entries in the file will be numeric, except for missing values which are indicated with NaN. For example, to read in a modified version of the furnace data for the next homework assignment

```
>> load furnace.txt
```

will create the matrix `furnace`. The matrix is given a name based on the file name (the extension is dropped).

If the file extension is `.mat`, the file is treated as a matlab file; otherwise it is considered a text file. These can be overridden with the `-ascii` and `-mat` options to the `load` command.

The command

```
>> load all a b c
```

will load only variables `a`, `b` and `c` from the file `all.mat`, whereas the command

```
>> load all
```

will load all variables in the file.

Storing Matlab variables

Unlike S-Plus which automatically stores the objects created or altered during a session, or R which asks whether you want to save the Workspace Image, Matlab will just quit and clear the memory. You need to explicitly save your Matlab objects.

```
>> save allfile
```

will save the complete workspace in a file `allfile.mat`.

```
>> save furnacefile furnace
```

will save only the variable `furnace` in the file `furnacefile.mat`.

To store a variable in a text file, do

```
>> save furnacetxt.txt furnace -ascii
```

If you give multiple arrays, they will be concatenating one after the other (not side by side).

Arithmetic in Matlab

As Matlab is a matrix language, all arithmetic defaults to standard matrix arithmetic. So for example, for two 3x3 matrices A & B, A+B, A-B will do standard matrix addition (like S). However for A*B

```
>> A = magic(3)
```

```
A =
```

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

```
>> B
```

```
B =
```

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

```
>> A * B
```

```
ans =
```

```
    28    73   118
    34    79   124
```

28 73 118

you get matrix multiplication (like $A \%* \% B$ in S). To get componentwise multiplication

```
>> A .* B
```

```
ans =
```

```
     8     4    42
     6    25    56
    12    54    18
```

The `.` modifier also exists with `./`, `.\`, and `.^`. Since the default is to do matrix multiplication, you need to watch your dimension. For example

```
>> (1:3) * (1:3)
```

```
??? Error using ==> *
```

```
Inner matrix dimensions must agree.
```

```
>> (1:3) * (1:3)'
```

```
ans =
```

```
14
```

All with matrix division you need to be careful. A / B gives $A * \text{inv}(B)$, where $A \setminus B$ gives $\text{inv}(A) * B$.

When dealing with scalars in arithmetic, they act as expected.

Similar to S-Plus/R, Matlab is a vectorized language. So for example

```
>> angle = pi*(0:1/4:1)
```

```
angle =
```

```
0    0.7854    1.5708    2.3562    3.1416
```

```
>> sin(angle)
```

```
ans =
```

```
0    0.7071    1.0000    0.7071    0.0000
```

Where possible, you should use the vectorized approach to calculations. It generally will be more efficient.

Help

There are two basic ways to get help in Matlab. There is the help system that I mentioned earlier. If you close the help window, you can get it back by selecting it under the View menu.

You can also use the command help.

```
>> help help
```

```
HELP On-line help, display text at command line.
```

```
HELP, by itself, lists all primary help topics. Each primary topic corresponds to a directory name on the MATLABPATH.
```

```
"HELP TOPIC" gives help on the specified topic. The topic can be a command name, a directory name, or a MATLABPATH relative partial pathname (see HELP PARTIALPATH). If it is a command name, HELP displays information on that command. If it is a directory name, HELP displays the Table-Of-Contents for the specified directory. For example, "help general" and "help matlab/general" both list the Table-Of-Contents for the directory toolbox/matlab/general.
```

and so on

```
For tips on creating help for your m-files 'type help.m'.
```

```
See also LOOKFOR, WHAT, WHICH, DIR, MORE.
```

If your not sure what the command name is, the lookfor command can be useful.

```
>> lookfor regress
```

```
LEVERAGE Regression diagnostic.
```

```
REGRESS Multiple linear regression using  
least squares.
```

```
REGSTATS Regression diagnostics for linear  
models.
```

```
RIDGE Ridge regression.
```

```
ROBUSTFIT Robust linear regression
```

```
STEPWISE Interactive tool for stepwise  
regression.
```

```
TREEDISP Show classification or regression  
tree graphically.
```

```
TREEFIT Fit a tree-based model for  
classification or regression.
```

This will list possibilities for the command you are interested, based on their help files.

Also, similar to S-Plus/R you can see Matlab functions. Give a command similar to

```
>> type mean.m
```

```
function y = mean(x,dim)
%MEAN    Average or mean value.
%    For vectors, MEAN(X) is the mean value of the elements in X. For
%    matrices, MEAN(X) is a row vector containing the mean value of
%    each column. For N-D arrays, MEAN(X) is the mean value of the
%    elements along the first non-singleton dimension of X.
%
%    MEAN(X,DIM) takes the mean along the dimension DIM of X.
%
%    Example: If X = [0 1 2
%                    3 4 5]
%
%    then mean(X,1) is [1.5 2.5 3.5] and mean(X,2) is [1
%                                                    4]
%
%    See also MEDIAN, STD, MIN, MAX, COV.
%
%    Copyright 1984-2002 The MathWorks, Inc.
%    $Revision: 5.17 $    $Date: 2002/06/05 17:06:39 $

if nargin==1,
    % Determine which dimension SUM will use
    dim = min(find(size(x)~=1));
    if isempty(dim), dim = 1; end

    y = sum(x)/size(x,dim);
else
    y = sum(x,dim)/size(x,dim);
end
```


The file name to print is the function name followed by a `.m`. The filenames for all functions and scripts in Matlab must have a `.m` extension.

Writing scripts and functions

When writing scripts in Matlab, just put the commands that you would type at the prompt in a text file. As mentioned before, the file name has to end with `.m`, such as `script.m`. To run the script, just type the name of the script file, without the `.m`.

To comment your code, start the comment with a `%` (see the code for `mean.m` earlier).

Also as I mentioned before, functions need to also go into text files with names ending in `.m`. Usually the name of the file is the name of the function with `.m` appended to it (e.g. the `mean` function is in `mean.m`). Actually when you call a function, Matlab does the call based on the file name and not what the function is called inside the code

The structure of a function is as follows

```
function output = funname(arg1, arg2, arg3)
% One line description of the function.
% Detailed descriptions. The first block
% of lines starting with % characters is
```

```
% the output when help funname is called.  
% The first line of this comment block is  
% used by the lookfor command.
```

```
function code
```

In matlab you don't need to explicitly return the value, as the last value as calculated is returned

```
function [product, quotient] = proddiv(x, y)
```

```
% Calculates product and quotient of two numbers
```

```
% proddiv(x, y) returns x*y and x/y.
```

```
product = x*y;
```

```
quotient = x/y;
```

When creating any function you may use any text editor, however Matlab (at least the Windows version) comes with a M-file editor.

In functions that return more than a single object, like the above, you need to store the output for each object to be able to access everything as the default is to return only the first entry of the output list.

```
>> proddiv(5,2)
```

```
ans =
```

```
10
```

```
>> pr = proddiv(5,2)

pr =

    10
>> [pr, qu] = proddiv(5,2)

pr =

    10

qu =

    2.5000
```

You do not need to store everything however. If you only need to access the 1st and 3rd output entries out of 4, you only need to store the 1st 3 of the list.

Matlab is a full featured programming language, including the standard flow control commands, including for, while, if-else, and switch.

For example, the for loops work as follows

```
fact = 1;
for i = 1:10
    fact = fact * i;
end
```

Matlab indicates the end of the loop with an end statement instead of S-Plus\R using { and } to indicate the code inside the loop.

The if-else command in Matlab is more complicated than the one in S-Plus/R. The form of the call is

```
if condition1
    (commands if condition1 is true)
elseif condition2
    (commands if condition2 is true)
else
    (commands if none of the conditions is
true)
end
```

You can have as many elseif sets as you want and the else block is not needed.

Another structure that I haven't seen in other programming languages is the try-catch block. It is used for error checking.

```
>> A = magic(3);
>> B = reshape(1:9,3,3);
>> C = magic(4);
```

```
>> try
    z = A*B;
catch
    z = nan;
    disp('A and B are not conformable')
end
>> z
```

```
z =

    28    73   118
    34    79   124
    28    73   118
```

```
>> try
    z = A*C;
catch
    z = nan;
    disp('A and C are not conformable')
end
A and C are not conformable
>> z
```

```
z =

    NaN
```

Next time – Statistics toolbox and graphics