Matlab Mini Manual

(of Matlab version 4)

1 Introduction

MATLAB is an interactive programming environment that facilities dealing with matrix computation, numerical analysis and graphics.

MATLAB stands for *mat*rix *lab*oratory and was initially intended to provide interactive access to the LINPACK and EISPACK packages. These packages were written in FORTRAN and consist of some sophisticated code for matrix problems. With MATLAB it became possible to experiment interactively with these packages without having to write lengthy FORTRAN code (or C code nowadays). Other features of MATLAB are its extensive help facilities and the ease with which users can make their own library of functions.

Throughout this tutorial we will give you an overview of various things and refer you to MATLAB's on-line help for more information. The best way to learn is by experimentation, and the best way this tutorial can help you is by telling you the basics and giving you directions towards which you can take off exploring on your own.

2 MATLAB basics and help facilities

To begin a MATLAB session type at the prompt

```
Matlab
```

or from MSWINDOWS you can double-click the MATLAB icon to begin the session. If properly installed, the system will show for split second a small welcoming window and then some general information and something like

```
Commands to get started: intro, demo, help help
Commands for more information: help, whatsnew, info
```

If you are new to MATLAB you might take their advice and type

```
>> intro
```

The >> should not be typed, it is MATLAB's way of saying that it is ready for you to enter a command, like intro. Typing >> demo will bring up a window from which various demos can be executed, but it is not the way to learn MATLAB. To quit the MATLAB session type

```
>> quit
```

or

```
>> exit
```

Now you have learned to enter MATLAB and quit it again. These are the most important things to know about any program. Before going into the details in the following sections, let us do some simple tests. Enter MATLAB again and type

```
>> x=2
>> y=3;
>> z=x*y
```

After entering the first line, MATLAB immediately responds that x is indeed equal to 2. The second line produces nothing on screen although it does assign the value 3 to the variable y. The semicolumn ; tells MATLAB to execute the command silently. The third line has no ; and MATLAB responds with

```
z =

   6
```

If you type >> x*y then there is no variable to assign the value of x*y to. In such cases MATLAB assigns the outcome to the variable ans (from *ans*wer).

Typing the following three lines will produce a plot.

```
>> x=[0 1 2 3 4 5];
>> y=sin(x);
>> plot(x,y);
```

Here x is assigned the value of the array [0 1 2 3 4 5]. At this stage we do not want to go into the details of arrays and sin and plot yet (we come to it in later sections), instead we want to convey the message to use MATLAB's on-line help whenever you encounter something unfamiliar. Look at the second line. Apparently we can calculate the sine of an array! How is it defined? Type

```
>> help sin
```

and MATLAB will tell that sin(x) is the sine of *the elements* of x. Okay, so y=sin(x) assigns y the array [sin(0) sin(1) sin(2) sin(3) sin(4) sin(5)]. With the command plot(x,y) the actual plot is made, see Figure. 1. To see how plot works type

```
>> help plot
```

There is a lot of help available on plot. If it scrolls off your screen you might want to enter >> more on and then >> help plot again. The help on plot is extensive and it should provide enough information to answer most questions you may have about plotting.

Another way of getting help is with the lookfor command. Suppose you wish to save the plot in a file on disk. Typing help save turns out to be of little use. In such cases try
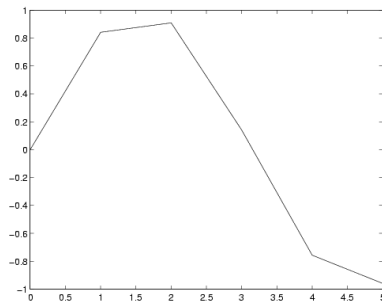
Figure 1: A coarse plot of $\sin(x)$.

```
>> lookfor save
```

This will print on screen a list of commands that have something to do with "save":

```
DIARY   Save the text of a MATLAB session.
SAVE    Save workspace variables on disk.
HARDCOPY Save figure window to file.
PRINT  Print graph or save graph to file.
WAVWRITE Saves Microsoft Windows 3.1 .WAV ..
IDUISESS Handles load and save session ..
```

Apparently `hardcopy` or `print` are the commands we need to look at. The help on `print` will tell you that

```
>> print -deps myfig.eps
```

saves the current plot in encapsulated postscript format in the file with name `myfig.eps`. Consult `help print` once again if you need a format other than postscript.

For more information on getting help, type

```
>> help help
```

which, you guessed it, gives help on using help. If you are on a Unix machine with a web browser, try also

```
>> doc
```

2.1 Loading, saving and m-files

The interactive nature of MATLAB is useful, but at the end of the day you want your commands and results saved in a file. Bring up an editor[1] and type in the following MATLAB program:

```
% These are comments. In Matlab you can
% add comments with a % character
% This is the standard hello world program

disp('Hello world!');
```

Save the program with the filename `hello.m`. Then typing `hello` at the MATLAB prompt should give

```
>> hello
Hello world!
>>
```

---

[1] On a Unix system you may use editors such as `emacs`, `vi` and many others. On MSWINDOWS there is for example `notepad`, but you might prefer to use MATLAB's inbuilt editor. Warning: If you use a word-processor to type in MATLAB programs, be sure to save the file as a plain text file or ASCII file. By default word processors save files in a format that MATLAB cannot read.

Notice that within MATLAB you don't have to type any funny commands to *load* your program. Simply type the name of the file (without the `.m`) and MATLAB will do its job. It may happen though that MATLAB complains that it cannot find `hello`. In this case use MATLAB's pathbrowser or commands `cd`, `ls` and `pwd`, to navigate in the file system and change your current MATLAB working directory to the directory where your file `hello.m` resides. If you want to be able to have access to files on two or more separate directories *simultaneously* type

```
>> help path
```

for more information. Be aware that MATLAB will only look for the files with extension `.m` so you are forced to use it. There are no work-arounds for this.

In the previous section you saw how to save a plot. It is also possible to save and load your variables on disk. Try

```
>> help save
```

and to load them the next session, try

```
>> help load
```

See also Subsection 3.4.2. For simple problems there is no need to save and load the variables. Simply save the m-file that has all the commands and execute the m-file when needed. Yet another way to save results is with the `diary` command. From the moment you type

```
>> diary myoutput.txt
```

all subsequent commands that you type, and its output on screen, are also send to the file `myoutput.txt`. In fact *everything* that appears on screen will also appear in `myoutput.txt`. With

```
>> diary off
```

the file `myoutput.txt` is closed.

3 Scalars, arrays, matrices and strings

MATLAB version 4 has three basic data types: strings, scalars and matrices. Arrays are just matrices that have only one row. MATLAB has also lots of built-in functions to work with these things. You have already seen the `disp` function in our hello-world program.

3.1 Strings

Starting with strings, you can assign a string to a variable like this:

```
name = 'Indiana Jones';
```

Note that it is a syntax error to quote the string with anything other than the forward quote marks. So, the following are *wrong!*

```
name = "Indiana Jones";    % wrong!
name = `Indiana Jones`;    % wrong!
```

In a MATLAB program you can prompt the user and ask him to enter in a string with the `input` command:

```
% This is a rather more social program,
% assuming you're a man
you = input('Hello! Who are you? ','s');
dad = input('What''s your dad''s name? ','s');
fprintf(1,'Hail oh %s son of %s.\n',you,dad);
```

The `input` command takes two arguments. The first argument is the string that you want the user to be prompted with. You could stick in a variable instead of a fixed string if you wanted to. The second argument, `'s'`, tells MATLAB to expect the user to enter a string. If you omit the second argument, then MATLAB will be expecting a number, and upon you entering your name, MATLAB will complain. Finally, it *returns* the value that the user enters, and that value is passed on through assignment to the variable `you`.

The `fprintf` command gives you more flexibility in producing output than the `disp` command. Briefly, `fprintf` takes two or three or more arguments. The first argument is a *file descriptor*. File descriptors are integers that reference places where you can send output and receive input from. In MATLAB, file descriptor 1 is what you use when you want to send things to the screen. The terminology you may hear is that file descriptor 1 sends things to the *standard output*.

The rest of the arguments depend on what you want to print. If all you want to print is a fixed string, then you just put that in as your second argument. For example:

```
fprintf(1,'Hello world!\n');
```

The `\n` sequence will switch you over to the next line. `disp` will do this automatically, in `fprintf` you must explicitly state that you wish to go to a new line. This is a feature, since there may be situations where you do *not* want to go to a new line.

If you want to print the values of variables interspersed with your string then you need to put appropriate markers like `%s` to indicate where you want your variables to go. Then, in the subsequent arguments you list the variables in the appropriate order, making sure to match the markers. There are many markers and the MATLAB online help will refer you to a C manual. The most commonly used markers are the following:

`%s`	Strings
`%d`	Integers (otherwise you get things like `5.0000`)
`%g`	Real numbers in scientific notation, like `1e-06`

In our example above, we just used `%s`. You will see further examples later on.

Note that if you merely want to print a variable, it is better to use `disp` since it will format it for you. `fprintf` is more useful for those occasions where you want to do the formatting yourself as well as for sending things to a file.

In the appendix, Sections 6.13 and 6.14 you find listings of the standard string commands and the input/output commands like `fprintf`.

3.2 Scalars

Scalars can be assigned, inputed and printed in a similar fashion. Here is an example:

```
% yet another one of these happy programs
```

```
age = input('How old are you? ');
if (age < 75)
  ll = 365.25*24*(75 - age);
  fprintf(1,'You have %g hours left of',ll);
  fprintf(1,' average life expectancy.\n');
else
  fprintf(1,'Geez! You are that old?!\n');
end
```

Note the following:

- String and numeric variables look the same. You don't have to declare the type of the variable anywhere. MATLAB will make sure to do the right thing.

- When we use `input` to get the value of a numeric variable we omit the second `'s'` argument. This way, MATLAB will do error-checking and complain if you entered something that's not a number.

- You can use `fprintf` to print numeric variables in a similar fashion, but you got to use the `%g` marker. If you are printing an integer you must use the `%d` marker, otherwise MATLAB will stick in a few zeroes as decimal places to your integer. It is obvious that you can mix strings and numbers in an `fprintf` command, so long as you don't mix up the order of the variables listed afterwards.

- On the line

  ```
  ll = 365.25*24*(75 - age);
  ```

  we see how you can do simple computations in MATLAB. It's very similar to C and FORTRAN and to learn more about the operators you have available type

  ```
  >> help ops
  >> help relops
  ```

- Finally, we have an example of an `if` statement. We will talk of that more later. The meaning should be intuitively obvious.

In addition to ordinary numbers, you may also have complex numbers. The symbols `i` and `j` are reserved for such use. For example you can say:

```
z = 3 + 4*i;
```

or

```
z = 3 + 4*j;
```

where `i` and `j` represent $\sqrt{-1}$. If you are already using the symbols `i` and `j` as variables, then you can get a new complex unit and use it in the usual way by saying:

```
ii = sqrt(-1);
z = 3 + 4*ii;
```

Lists on what you can do with scalars and predefined constants, are in the appendix, Sections 6.2, 6.4, 6.6 and 6.7.

3.3 Arrays

Next we talk about arrays. Arrays and matrices are MATLAB's distinguishing feature. In MATLAB arrays are dynamic and they are indexed from 1. You can assign them element by element with commands like:

```
a(1) = 23;
a(2) = input('Enter a(2)');
a(3) = a(1)+a(2);
```

It is a syntax error to assign or refer to `a(0)`. This is unfortunate since in some cases the 0-indexing is more convenient. Note that you don't have to initialize the array or state it's size at any point. The array will make sure to grow itself as you index higher and higher indices.

Suppose that you do this:

```
a(1) = 10;
a(3) = 20;
```

At this point, `a` has grown to size 3. But `a(2)` hasn't been assigned a value yet. In such situations, during growth any unset elements are set to zero. It is good programming practice however not to depend on this and always initialize all the elements to their proper values.

Notice that for the sake of efficiency you might not like the idea of growing arrays. This is because every time the array is grown, a new chunk of memory must be allocated for it, and contents have to be copied. In that case, you can set the size of the array by initializing it with the `zeros` command:

```
a = zeros(1,100);
```

This will set `a(1),a(2),...,a(100)` all equal to zero. Then, so long as you respect these boundaries, the array will not have to be grown at any point.

Here are some other ways to make assignments to arrays:

```
x = [3 4 5 6];
```

will set `x` equal to the array of 4 values with `x(1)=3`, `x(2)=4`, `x(3)=5` and `x(4)=6`. You can recursively add elements to your array `x` in various ways if you include `x` on the right hand side. For example, you can make assignments like

```
x = [x 1 2]    % append 1 and 2 at the end
x = [1 x x 3] % append 1, x and 3 to x
```

How about making deletions? Well, first of all notice that we can access parts of the array with the following indexing scheme:

```
y = x(2:4);
```

will return the an array of `x(2)`, `x(3)`, `x(4)`. So, if you want to delete the last element of the array, you just have to find the size of the array, which you can do with the `size` or `length` command, and then

```
x=x(1:length(x)-1);
```

effectively deletes the last entry from `x`.

You can use multiple indices on the left hand side of the assignment:

```
x([1 5 10])=[0.1 0.2 0.3];
```

This assigns `x(1)=0.1`, `x(5)=0.2` and `x(10)=0.3`. By the way, this opens up another way to delete entries from an array:

```
x=[1 2 3 4 5];
x([3 5])=[];
```

Now `x=[1 2 4]`. In MATLAB `[]` is an array or matrix with zero elements, so `x([3 5])=[]` instructs MATLAB to remove `x(3)` and `x(5)` from the array `x`.

Yet another way to setup arrays is like this:

```
x = 3:1:6;
```

This will set `x` equal to an array of equidistant values that begin at 3, end at 6 and are separated from each other by steps of 1. You can even make backwards steps if you provide a negative stepsize, like this:

```
x = 6:-1:3;
```

It is common to set up arrays like these when you want to plot a function whose values are known at equidistant points.

3.3.1 Polynomials

A polynomial $p_n s^n + p_{n-1} s^{n-1} + \cdots + p_0$ is in MATLAB represented by the array of its coefficients

$$\mathtt{p} = [\ p_n \quad p_{n-1} \quad \cdots \quad p_0\ ]$$

Formally, then, arrays and polynomials are the same thing in MATLAB. Be aware of the reversed order of the coefficients: `p(1)` equals p_n and `p(length(p))` is the constant coefficient. MATLAB has several commands for polynomials. For example the product of two polynomials `p` and `q` can be found with

```
r=conv(p,q)
```

And what about this one:

```
pz=roots(p)
```

It returns an array of the zeros of `p`. As you probably know, there is no general analytic expression for the zeros of a polynomial if its degree exceeds 4. Numerical computation of zeros is inherently an iterative process and it is not at all a trivial task.

See the appendix, Sections 6.2, 6.4, 6.6, 6.7, 6.9 and 6.10 for listings of the standard array and polynomial commands. Several of the *toolboxes* have many more commands that deal with polynomials. See `help` for a list of the installed toolboxes.

3.4 Matrices

In MATLAB, arrays are matrices that have only one row. Like arrays, matrices can be defined element by element like this:

```
a(1,1) = 1; a(1,2) = 0;
a(2,1) = 0; a(2,2) = 1;
```

Like arrays, matrices grow themselves dynamically as needed when you add elements in this fashion. Upon growth, any unset elements default to zero just like they do in arrays. If you don't want that, you can use the `zeros` command to initialize the matrix to a specific size and set it equal to zero, and then take it from there. For instance, the following example will create a zero matrix with 4 rows and 5 columns:

```
A = zeros(4,5);
```

To get the size of a matrix, we use the `size` command like this:

```
[rows,columns] = size(A);
```

When this command executes, the variable `rows` is set equal to the number of rows and `columns` is set equal to the number of columns. If you are only interested in the number of rows, or the number of columns then you can use the following variants of `size` to obtain them:

```
rows = size(A,1);
columns = size(A,2);
```

Since arrays are just matrices with one row, you can use the `size(array,2)` construct to get hold of the size of the array. Unfortunately, if you were to say:

```
s = size(array);
```

it would be wrong, because `size` returns both the number of rows and columns and since you only care to pick up one of the two numbers, you pick up the number of rows, which for arrays is always equal to 1. Not what you want!

Naturally, there are a few other ways to assign values to a matrix. One way is like this:

```
A = [ 1 0 0 ; 0 1 0 ; 0 0 1]
```

This will set `A` equal to the 3 by 3 identity matrix. In this notation you list the rows and separate them with semicolons.

In addition to that you can extract pieces of the matrix, just like earlier we showed you how to extract pieces of the arrays. Here are some examples of what you can do:

```
a = A(:,2);       % the 2nd column of A
b = A(3,:);       % the 3rd row of A
c = A(1:4,3);     % a 4 by 1 submatrix of A
d = A(:,[2 10]);  % columns 2 and 10 of A
```

In general, if `v` and `w` are arrays with integer components, then `A(v,w)` is the matrix obtained by taking the elements of A with row subscripts from `v` and column subscripts from `w`. So:

```
n = size(A,2);
A = A(:,n:-1:1);
```

will reverse the columns of A. Moreover, you can have *all* of these constructs appear on the left hand side of an assignment and MATLAB will do the right thing. For instance

```
A(:,[3 5 10]) = B(:,1:3)
```

replaces the third, fifth and tenth columns of `A` with the first three columns of `B`.

In addition to getting submatrices of matrices, MATLAB allows you to put together block matrices from smaller matrices. For example if `A,B,C,D` are a square matrices of the same size, then you can put together a block matrix like this:

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

Finally, you can get the complex conjugate transpose of matrix by putting a ′ mark next to it. For example:

```
A = [ 2 4 1 ; 2 1 5 ; 4 2 6];
Atrans = A';
```

MATLAB provides functions that return many special matrices. These functions are listed in the appendix, Section 6.5 and we urge you to look these up with the `help` command and experiment.

To display the contents of matrices you can simply use the `disp` command. For example, to display the 4 by 4 Hilbert matrix you want to say:

```
>> disp(hilb(4))
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
```

The Hilbert matrix is a famous example of a badly conditioned matrix. It is also famous because the exact inverse of it is known analytically and can be computed with the `invhilb` command:

```
>> disp(invhilb(5))
     16    -120     240    -140
   -120    1200   -2700    1680
    240   -2700    6480   -4200
   -140    1680   -4200    2800
```

This way you can interactively use these famous matrices to study concepts such as ill-conditioning.

3.4.1 Matrix/Array Operations

- **Matrix addition/subtraction**: Matrices can be added and subtracted like this:

  ```
  A = B + C;
  A = B - C;
  ```

  It is necessary for both matrices `B` and `C` to have the same size. The exception to this rule is when adding with a scalar:

  ```
  A = B + 4;
  ```

  In this example, all the elements of `B` are increased by 4 and the resulting matrix is stored in `A`.

- **Matrix multiplication**: Matrices `B` and `C` can be multiplied if they have sizes $n \times p$ and $p \times m$ correspondingly. The product then is evaluated by the well-known formula

  $$A_{ij} = \sum_{k=1}^{p} B_{ik} C_{kj}, \ \forall i \in \{1, \ldots, n\}, \ \forall j \in \{1, \ldots, m\}$$

  In MATLAB, to do this you say:

  ```
  A = B*C;
  ```

  You can also multiply all elements of a matrix by a scalar like this:

  ```
  A = 4*B;
  A = B*4;   % both are equivalent
  ```

  Since vectors are matrices that are $1 \times n$, you can use this mechanism to take the dot product of two vectors by transposing one of the vectors. For example:

```
x = [2 4 1 5 3];
y = [5 3 5 2 9];
p = x'*y;
```

This way, x' is now a $n \times 1$ "matrix" and y is $1 \times n$ and the two can be multiplied, which is the same as taking their dot product $\sum_{k=1}^{k=n} \bar{x}(k) y(k)$.

Another common application of this is to apply $n \times n$ matrices to vectors. There is a catch though: In MATLAB, vectors are defined as matrices with one row, i.e. as $1 \times n$. If you are used to writing the matrix product as Ax, then you have to transpose the vector. For example:

```
A = [1 3 2; 3 2 5; 4 6 2];  % 3*3 matrix
x = [2 5 1];                % row vector
y = A*x;                    % WRONG!
y = A*x';                   % correct
```

- **Array multiplication (element-wise multiplication):** This is an alternative way to multiply matrices:

$$C_{ij} = A_{ij} B_{ij}, \quad \forall i \in \{1, \ldots, n\}, \ \forall j \in \{1, \ldots, m\}$$

  This is not the traditional matrix multiplication but it's something that shows up in many applications. You can do it in MATLAB like this:

```
C = A.*B;
```

  For example to plot the function $y(t) = t^2$ one can do

```
t=-1:0.1:2; % a gridded interval [-1,2]
y=t.*t;
plot(t,y);
```

- **Array division (element-wise division):** Likewise you can divide arrays in MATLAB according to the formula:

$$C_{ij} = \frac{A_{ij}}{B_{ij}}, \quad \forall i \in \{1, \ldots, n\}, \ \forall j \in \{1, \ldots, m\}$$

  by using the `./` operator like this:

```
C = A./B;
```

- **Matrix division:** There are two *matrix division* operators in MATLAB: / and \. In general

```
X = A\B is a solution to A*X = B
X = B/A is a solution to X*A = B.
```

  This means that A\B is defined whenever A has full row rank and has as many rows as B. Likewise B/A is defined whenever A has full column rank and has as many columns as B. If A is a square matrix then it is factored using Gaussian elimination. Then the equations A*X(:,j)=B(:,j) are being solved for every column of B. The result is a matrix with the same dimensions as B. If A is not square, it is factored using the Householder orthogonalization with column pivoting. Then the corresponding equations will be solved in the least squares fit sense. Right division B/A in MATLAB is computed in terms of left division by B/A = (A'\B')'. For more information type

```
>> help slash
```

MATLAB has a *plethora* of commands that do almost anything that you would ever want to do to a matrix. And we have only discussed a subset of the operations that are permitted with matrices. A list of matrix commands can be found in the appendix, Sections 6.2, 6.4, 6.6 and 6.8 and 6.5.

3.4.2 Importing and exporting matrices and arrays

To load data in MATLAB that is not generated by MATLAB you need to have this data in plain text (ASCII) format. For example if you have a file `measurents.txt` with contents

```
1  1.1  1.11e+01
2  2.2  2.22e-07
3  3.3  3.333333
4  4.4  4.444444
```

then to load this data in MATLAB you type

```
load measurements.txt
```

This defines the matrix `measurements` (the name of the file without the extension `.txt`). This matrix will have 3 columns and 4 rows, and `measurements(2,3)` will equal $2.22 \cdot 10^{-7}$.

If you need to *save* this matrix `measurements` in ASCII format then it suffices to do

```
save myfile.huh measurements -ASCII
```

This will make an ASCII file `myfile.huh` that will look similar to `measurements.txt`.

4 Flow Control

So far, we have spent most of our time discussing the data structures available in MATLAB, and how they can be manipulated, as well as inputted and outputted. Now we continue this discussion by discussing how MATLAB deals with *flow control*.

4.1 For loops

In MATLAB, a for-loop has the following syntax:

```
for v = matrix
  statement1;
  statement2;
  ....
end
```

The columns of `matrix` are stored one at a time in the variable v, and then the statements up to the `end` statement are executed. If you wish to loop over the rows of the matrix then you simply transpose it and plug it in. It is recommended that the commands between the `for` statement and the `end` statement are indented by two spaces so that the reader of your code can visually see that these statements are enclosed in a loop.

In many cases, we use arrays for matrices, and then the for-loop reduces to the usual for-loop we know in languages like FORTRAN and C. In particular using expressions of the form `for v=1:10` will effectively make the loop variable be a scalar that goes from 1 to 10 in steps of 1. Using an expression of the form `start:step:stop` will allow you to loop a scalar from

`a == b`	True if `a` equals `b`
`a > b`	True if `a` is greater than `b`
`a < b`	True if `a` is smaller than `b`
`a <= b`	True if `a` is smaller or equal to `b`
`a >= b`	True if `a` is greater or equal to `b`
`a ~= b`	True if `a` is not equal to `b`
`a & b`	True if both booleans `a` and `b` are true
`a	b`	True if at least one of `a` or `b` is true.
`a xor b`	True only if only one of `a` or `b` is true.
`~a`	True if `a` is false.

Table 1: Some boolean expressions in MATLAB.

value `start` all the way to value `stop` with stepsize `step`. For instance the Hilbert matrix is defined by the equation:

$$A_{ij} = \frac{1}{i+j+1}$$

If we didn't have the `hilb` command, then we would use for-loops to initialize it, like this:

```
N = 10;
A = zeros(N,N);
for i = 1:N
  for j = 1:N
    A(i,j) = 1/(i+j-1);
  end
end
```

4.2   While loops

In MATLAB while loops follow the following format:

```
while variable
  statement1;
  statement2;
  ....
  statementn;
end
```

where `variable` is almost always a *boolean expression* of some sort. In MATLAB you can compose boolean expressions as shown in Table 1.

Here is an example of a MATLAB program that uses the while loop:

```
n = 1;
while prod(1:n) < 1.e100
  n = n + 1;
end
disp(n);
```

This program will display the first integer for which *n*! is a 100-digit number. The `prod` function takes an array (or matrix) as argument and returns the product of it's elements. In this case, `prod(1:n)` is the factorial *n*!.

MATLAB does not have repeat-until loops.

4.3   If and Break statements

The simplest way to set-up an `if` branch is like this:

```
if variable
  statement1;
  ....
  statementn;
end
```

The statements are executed only if the real part of the `variable` has all non-zero elements[2]. Otherwise, the program continues with executing the statements right after the `end` statement. The variable is usually the result of a boolean expression. The most general way to do if-branching is like this:

```
if variable
  statement1;
  ......
  statementn;
elseif variable2
  statement21;
  ......
  statement2n;
  % as many elseif statements as you want
else
  statementm1;
  ......
  statementmn;
end
```

In this case, if `variable` is true, then the statements right after it will be executed until the first `else` or `elseif` (whichever comes first), and then control will be passed over to the statements after the `end`. If `variable` is not true, then we check `variable2`. Now, if that one is true, we do the statements following thereafter until the next `else` or `elseif` and when we get there again we jump to `end`. Recursively, upon consecutive failures we check the next `elseif`. If all the `elseif` variables turn out to be false, then we execute the statements after the `else`. Note that the `elseif`s and/or the `else` can be omitted all together.

Here is a general example that illustrates the last two methods of flow control.

```
% Classic 3n+1 problem from number theory:
%
while 1
  n = input('Enter n, negative n quits. ');
  if n <= 0
    break
  end
  while n > 1
    if rem(n,2) == 0 % if n is even
      n = n/2;        % then divide by two
    else
      n = 3*n+1;
    end
  end
end
```

This example involves a fascinating problem from number theory. Take any positive integer. If it is even, divide it by 2; if it is odd, multiply it by 3 and add 1. Repeat this process until your integer becomes a 1. The fascinating unsolved problem is: Is there any integer for which the process does not terminate? The

---

[2]Note that in the most general case, `variable` could be a complex number. The `if` statement will only look into its real part.

conjecture is that such integers do not exist. However nobody has managed to prove it yet.

The `rem` command returns the remainder of a Euclidean division of two integers. (in this case `n` and `2`.)

5 Functions

MATLAB has been written so that it can be extended by the users. The simplest way to do that is to write `functions` in MATLAB code. We will illustrate this with an example: suppose you want to create a function called `stat` which will take an array and return it's mean and standard deviation. To do that you must create a file called `stat.m`. Then in that file, say the following:

```
function [mean, stdev] = stat(x)
% stat -- mean and standard deviation
%    The stat command returns the mean and
%    standard deviation of the elements of
%    and array. Typical syntax is like this:
%    [mean,dev] = stat(x);
m=length(x);
mean=sum(x)/m;
y=x-mean;
stdev=sqrt(sum(y.^2)/m);
```

The first line of the file should have the keyword `function` and then the syntax of the function that is being implemented. Notice the following things about the function syntax:

- The function name, `stat`, in the first line is the same as the name of the file, `stat.m`, but without the `.m`.

- Functions can have an arbitrary number of arguments. In this case there is only one such argument: `x`. The arguments are being passed by *value*. This means that the variable that the calling code passes to the function is *copied* and the copy is being given to the function. This means that if the function internally changes the value of `x`, the change will *not* reflect on the variable that you use as argument on your main calling code. Only the copy will be changed, and that copy will be discarded as soon as the function completes it's call.

- Of course it is not desirable to only pass things by value. The function has to communicate some information back to the calling code. In MATLAB the variables listed in brackets `[mean, stdev]`, are returned by the function to the user. This means that any changes made to those variables while inside the function will reflect on the corresponding variables on the calling code. So, if one were to call the function with:

  ```
  aa = 1:20;
  me = 0;
  st = 0;
  [me,st] = stat(aa);
  ```

  then the values of the variables `me` and `st` change after the call to `stat`.

- All other variables used in the function, like `y` and `m`, are local variables and are discarded when the function completes it's call.

- The lines with a `%` following the function definition are comments. However, these comments are what will be spit out if you type `help stat`.

  ```
  >> help stat
  stat -- mean and standard deviation
     The stat command returns the mean and
     standard deviation of the elements of
     and array. Typical syntax is like this:
     [mean,dev] = stat(x);
  >>
  ```

  You usually want to make the first comment line be a summary of what the function does because in some instances only the first line will get to be displayed, so it should be complete. Then in the lines afterwards, you can explain how the function is meant to be used.

In MATLAB you can put more than one function in a single m-file, but only the first function is *directly* accessible to the users. So if you were to have a file `mystat.m` with contents

```
function [mean, stdev] = mystat(x)
% mystat -- my mean and standard deviation
m=length(x);
mean=mymean(x); % defined below in this file
y=x-mean;
stdev=sqrt(sum(y.^2)/m);

function mm=mymean(x)
mm=sum(x)/length(x);
```

then `mystat` can be called by the users but the local function `mymean` that also appears in this file can only be called from within this file; trying to call `mymean` directly by the user will produce an error message.

A thing that could cause you problems is *name collision*: what if the name you choose for one of your functions happens to be the name of an existing built-in MATLAB function? Then, your function will be ignored and MATLAB will call up the built-in version instead. To find out if this is the case use the `which` command to see what MATLAB has to say about your function.

5.1 Array promotion

In many cases MATLAB provides ways to eliminate the use of `for`-loops and when you make use of them, your programs will generally run faster. A typical example is *array promotion*. Take a very simple function

```
function y = f(x)
```

which takes a number `x` and returns another number `y`. Typical such functions are `sin`, `cos`, `tan` and you can always write your own. Now, if instead of a number you plug in an array or a matrix, then the function will be applied on every element of your array or matrix and an array of the same size will be returned. This is the reason why you don't have to specify in the function definition whether `x` and `y` are simple numbers, or arrays in the first place! To MATLAB everything is a matrix as far as functions are concerned. Ordinary numbers are seen as 1×1 matrices rather than numbers. You should keep that

in mind when writing functions: sometimes you may want to multiply your `x` and `y` with the `.*` operator instead of the `*` to handle array promotion properly. Likewise with division. Expect to be surprised and be careful with array promotion.

Let's look at an example more closely. Suppose you write a function like this:

```
function x = foo(y,z)
x = y+z;
```

Then, you can do the following on the MATLAB prompt:

```
>> disp(foo(2,3))
     5
>> a = 1:1:6;
>> b = 1:2:12;
>> disp(foo(a,b))
     2     5     8    11    14    17
```

What you will *not* be allowed to do is this:

```
>> a = 1:1:10
>> b = 1:1:20
>> disp(foo(a,b))
??? Error using ==> +
Matrix dimensions must agree.

Error in ==> foo.m
On line 2  ==> x = y+z;
```

The arguments `a` and `b` can not be added because they don't have the same size. Notice by the way that we used addition as our example on purpose. We challenge you to try `*` versus `.*` and see the effects!

One use of functions is to build complex algorithms progressively from simpler ones. Another use is to automate certain commonly-used tasks. Functions do not solve all of the worlds problems, but they can help you a lot and you should use them when you have the feeling that your MATLAB program is getting too long and complicated and needs to be broken down to simpler components.

A good way to learn how to write functions is by copying from existing functions. To see the contents of an m-file, say, `hilb.m`, do `type hilb`.

```
>> type hilb

function H = hilb(n)
%HILB   Hilbert matrix.
%   HILB(N) is the N by N matrix with elements 1/(i+j-1),
%   which is a famous example of a badly conditioned matrix.
%   See INVHILB for the exact inverse.
%
%   This is also a good example of efficient Matlab programming
%   style where conventional FOR or DO loops are replaced by
%   vectorized statements.  This approach is faster, but uses
%   more storage.

%   C. Moler, 6-22-91.
%   Copyright (c) 1984-96 by The MathWorks, Inc.
%   $Revision: 5.4 $  $Date: 1996/08/15 21:52:09 $

%   I, J and E are matrices whose (i,j)-th element
%   is i, j and 1 respectively.

J = 1:n;
J = J(ones(n,1),:); % J(r,k)=k
I = J';              % I(r,k)=r
E = ones(n,n);
H = E./(I+J-1);
```

To include this code in the file you're editing you can either cut-and-paste it from the MATLAB-window or you can insert the file `hilb.m` directly. For the latter you need to locate the file first with the command

```
>> which hilb
```

Finally, we urge you to try to keep the function files readable. Functions written today may be difficult to understand in a few days if the layout and variable names are illogical. For reasons of readability it is sometimes useful to split commands like

```
A=myfunction(a,b,c,d,e,f,a2,b2,c2,d2,e2,f2)
```

into several parts. This can be done in MATLAB with three or more dots at the end of a line:

```
A=myfunction(a, b, c, d, e, f,  ...
             a2,b2,c2,d2,e2,f2)
```

6   Appendix: index of functions, commands and operations

This appendix contains lists of the functions, commands and operations that come with any distributation of MATLAB. The lists were extracted from the on-line help of MATLAB, version 4.2 for Unix. The commands are grouped by subject.

An installation of MATLAB has next to the standard set of functions and such, also several *toolboxes*. Toolboxes are packages of m-files that concentrate on a particular set of math or engineering problems. Including all the toolboxes in this appendix would have been too much. Type `help` to see what toolboxes there are on your machine. The toolboxes that at present can be

Chemometrics	Communications
Control System	Financial Toolbox
Frequency Domain System Identific.	Fuzzy Logic
Higher-Order Spectral Analysis	Image Processing
LMI Control	Mapping
Model Predictive Control	μ-Analysis and Synthesis
NAG	Neural Network
Optimization	Partial Differential Eqn.
QFT Control Design	Robust Control
Signal Processing	Spline
Statistics	System Identification
Symbolic/Extended Symbolic Math	Wavelet

Table 2: The toolboxes.

acquired from The Mathworks[3] are listed in Table 2. The list continues to grow.

MATLAB is still very much in development and even if the standard commands listed below are up to date "now", they need not be in future releases. This is usually not a big problem but be aware of it and always consult the on-line help.

6.1   General purpose commands

Extracted from `help general`.

---

[3]See http://www.mathworks.com/

MANAGING COMMANDS AND FUNCTIONS	
help	On-line documentation
doc	Load hypertext documentation
what	Directory listing of M-, MAT- and MEX-files
type	List M-file
lookfor	Keyword search through the HELP entries
which	Locate functions and files
demo	Run demos
path	Control MATLAB's search path

MANAGING VARIABLES AND THE WORKSPACE	
who	List current variables
whos	List current variables, long form
load	Retrieve variables from disk
save	Save workspace variables to disk
clear	Clear variables and functions from memory
pack	Consolidate workspace memory
size	Size of matrix
length	Length of vector
disp	Display matrix or text

WORKING WITH FILES AND THE OPERATING SYSTEM	
cd	Change current working directory
dir	Directory listing
delete	Delete file
getenv	Get environment value
!	Execute operating system command
unix	Execute operating system command & return result
diary	Save text of MATLAB session

CONTROLLING THE COMMAND WINDOW	
cedit	Set command line edit/recall facility parameters
clc	Clear command window
home	Send cursor home
format	Set output format
echo	Echo commands inside script files
more	Control paged output in command window

STARTING AND QUITTING FROM MATLAB	
quit	Terminate MATLAB
startup	M-file executed when MATLAB is invoked
matlabrc	Master startup M-file

GENERAL INFORMATION	
info	Info about MATLAB and The MathWorks, Inc
subscribe	Become subscribing user of MATLAB
hostid	MATLAB server host identification number
whatsnew	Info about new features not yet documented
ver	matlab, simulink, and toolbox version info

6.2 Operators and special characters

Extracted from `help ops`.

OPERATORS AND SPECIAL CHARACTERS		
+	Plus	see `help arith`
–	Minus	"
*	Matrix multiplication	"
.*	Array multiplication	"
^	Matrix power	"
.^	Array power	"
\	Backslash or left division	see `help slash`
/	Slash or right division	"
./	Array division	"
kron	Kronecker tensor product	see `help kron`
:	Colon	see `help colon`
( )	Parentheses	see `help paren`
[ ]	Brackets	"
.	Decimal point	see `help punct`
..	Parent directory	"
...	Continuation	"
,	Comma	"
;	Semicolon	"
%	Comment	"
!	Exclamation point	"
'	Transpose and quote	"
=	Assignment	"
==	Equality	see `help relop`
< >	Relational operators	"
&	Logical AND	"
|	Logical OR	"
~	Logical NOT	"
xor	Logical EXCLUSIVE OR	see `help xor`

LOGICAL CHARACTERISTICS	
exist	Check if variables or functions are defined
any	True if any element of vector is true
all	True if all elements of vector are true
find	Find indices of non-zero elements
isnan	True for Not-A-Number
isinf	True for infinite elements
finite	True for finite elements
isempty	True for empty matrix
isreal	True for real matrix
issparse	True for sparse matrix
isstr	True for text string
isglobal	True for global variables

6.3 Programming language constructs

Extracted from `help lang`.

MATLAB AS A PROGRAMMING LANGUAGE	
script	About MATLAB scripts and M-files
function	Add new function
eval	Execute string with MATLAB expression
feval	Execute function specified by string
global	Define global variable
nargchk	Validate number of input arguments
lasterr	Last error message

CONTROL FLOW	
if	Conditionally execute statements
else	Used with IF
elseif	Used with IF
end	Terminate the scope of FOR, WHILE and IF
for	Repeat statements a specific number of times
while	Repeat statements an indefinite number of times
break	Terminate execution of loop
return	Return to invoking function
error	Display message and abort function

INTERACTIVE INPUT	
input	Prompt for user input
keyboard	Invoke keyboard as if it were a Script-file
menu	Generate menu of choices for user input
pause	Wait for user response
uimenu	Create user interface menu
uicontrol	Create user interface control

DEBUGGING COMMANDS	
dbstop	Set breakpoint
dbclear	Remove breakpoint
dbcont	Resume execution
dbdown	Change local workspace context
dbstack	List who called whom
dbstatus	List all breakpoints
dbstep	Execute one or more lines
dbtype	List M-file with line numbers
dbup	Change local workspace context
dbquit	Quit debug mode
mexdebug	Debug MEX-files

6.4 Elementary matrices and matrix manipulation

Extracted from `help elmat`.

ELEMENTARY MATRICES	
zeros	Zeros matrix
ones	Ones matrix
eye	Identity matrix
rand	Uniformly distributed random numbers
randn	Normally distributed random numbers
linspace	Linearly spaced vector
logspace	Logarithmically spaced vector
meshgrid	X and Y arrays for 3-D plots
:	Regularly spaced vector

SPECIAL VARIABLES AND CONSTANTS	
ans	Most recent answer
eps	Floating point relative accuracy
realmax	Largest floating point number
realmin	Smallest positive floating point number
pi	3.1415926535897.....
i, j	Imaginary unit
inf	Infinity
NaN	Not-a-Number
flops	Count of floating point operations
nargin	Number of function input arguments
nargout	Number of function output arguments
computer	Computer type
isieee	True for computers with IEEE arithmetic
isstudent	True for the Student Edition
why	Succinct answer
version	MATLAB version number

TIME AND DATES	
clock	Wall clock
cputime	Elapsed CPU time
date	Calendar
etime	Elapsed time function
tic, toc	Stopwatch timer functions

MATRIX MANIPULATION	
diag	Create or extract diagonals
fliplr	Flip matrix in the left/right direction
flipud	Flip matrix in the up/down direction
reshape	Change size
rot90	Rotate matrix 90 degrees
tril	Extract lower triangular part
triu	Extract upper triangular part
:	Index into matrix, rearrange matrix

6.5 Specialized matrices

Extracted from `help specmat`.

SPECIALIZED MATRICES	
compan	Companion matrix
gallery	Several small test matrices
hadamard	Hadamard matrix
hankel	Hankel matrix
hilb	Hilbert matrix
invhilb	Inverse Hilbert matrix
kron	Kronecker tensor product
magic	Magic square
pascal	Pascal matrix
rosser	Classic symmetric eigenvalue test problem
toeplitz	Toeplitz matrix
vander	Vandermonde matrix
wilkinson	Wilkinson's eigenvalue test matrix

6.6 Elementary math functions

Extracted from `help elfun`.

TRIGONOMETRIC FUNCTIONS	
sin	Sine
sinh	Hyperbolic sine
asin	Inverse sine
asinh	Inverse hyperbolic sine
cos	Cosine
cosh	Hyperbolic cosine
acos	Inverse cosine
acosh	Inverse hyperbolic cosine
tan	Tangent
tanh	Hyperbolic tangent
atan	Inverse tangent
atan2	Four quadrant inverse tangent
atanh	Inverse hyperbolic tangent
sec	Secant
sech	Hyperbolic secant
asec	Inverse secant
asech	Inverse hyperbolic secant
csc	Cosecant
csch	Hyperbolic cosecant
acsc	Inverse cosecant
acsch	Inverse hyperbolic cosecant
cot	Cotangent
coth	Hyperbolic cotangent
acot	Inverse cotangent
acoth	Inverse hyperbolic cotangent

EXPONENTIAL FUCNTIONS	
exp	Exponential
log	Natural logarithm
log10	Common logarithm
sqrt	Square root

COMPLEX FUNCTIONS	
abs	Absolute value
angle	Phase angle
conj	Complex conjugate
imag	Complex imaginary part
real	Complex real part

NUMERIC	
fix	Round towards zero
floor	Round towards minus infinity
ceil	Round towards plus infinity
round	Round towards nearest integer
rem	Remainder after division
sign	Signum function

6.7 Specialized math functions

Extracted from `help specfun`.

SPECIALIZED MATH FUNCTIONS	
besselj	Bessel function of the first kind
bessely	Bessel function of the second kind
besseli	Modified Bessel function of the first kind
besselk	Modified Bessel function of the second kind
beta	Beta function
betainc	Incomplete beta function
betaln	Logarithm of beta function
ellipj	Jacobi elliptic functions
ellipke	Complete elliptic integral
erf	Error function
erfc	Complementary error function
erfcx	Scaled complementary error function
erfinv	Inverse error function
expint	Exponential integral function
gamma	Gamma function
gcd	Greatest common divisor
gammainc	Incomplete gamma function
lcm	Least common multiple
legendre	Associated Legendre function
gammaln	Logarithm of gamma function
log2	Dissect floating point numbers
pow2	Scale floating point numbers
rat	Rational approximation
rats	Rational output
cart2sph	Transform Cartesian to spherical coordinates
cart2pol	Transform Cartesian to polar coordinates
pol2cart	Transform polar to Cartesian coordinates
sph2cart	Transform spherical to Cartesian coordinates

6.8 Matrix functions - numerical linear algebra

Extracted from `help matfun`.

MATRIX ANALYSIS	
cond	Matrix condition number
norm	Matrix or vector norm
rcond	LINPACK reciprocal condition estimator
rank	Number of linearly independent rows or columns
det	Determinant
trace	Sum of diagonal elements
null	Null space
orth	Orthogonalization
rref	Reduced row echelon form

LINEAR EQUATIONS	
\ and /	Linear equation solution; use `help slash`
chol	Cholesky factorization
lu	Factors from Gaussian elimination
inv	Matrix inverse
qr	Orthogonal-triangular decomposition
qrdelete	Delete a column from the QR factorization
qrinsert	Insert a column in the QR factorization
nnls	Non-negative least-squares
pinv	Pseudoinverse
lscov	Least squares in the presence of known covariance

EIGENVALUES AND SINGULAR VALUES	
eig	Eigenvalues and eigenvectors
poly	Characteristic polynomial
polyeig	Polynomial eigenvalue problem
hess	Hessenberg form
qz	Generalized eigenvalues
rsf2csf	Real block diagonal form to complex diagonal form
cdf2rdf	Complex diagonal form to real block diagonal form
schur	Schur decomposition
balance	Diagonal scaling to improve eigenvalue accuracy
svd	Singular value decomposition

MATRIX FUNCTIONS	
expm	Matrix exponential
expm1	M-file implementation of expm
expm2	Matrix exponential via Taylor series
expm3	Matrix exponential via eigenvalues and eigenvectors
logm	Matrix logarithm
sqrtm	Matrix square root
funm	Evaluate general matrix function

6.9 Data analysis and Fourier transforms

Extracted from `help datafun`.

BASIC OPERATIONS	
max	Largest component
min	Smallest component
mean	Average or mean value
median	Median value
std	Standard deviation
sort	Sort in ascending order
sum	Sum of elements
prod	Product of elements
cumsum	Cumulative sum of elements
cumprod	Cumulative product of elements
trapz	Numerical integration using trapezoidal method

FINITE DIFFERENCES	
diff	Difference function and approximate derivative
gradient	Approximate gradient
del2	Five-point discrete Laplacian

VECTOR OPERATIONS	
cross	Vector cross product
dot	Vector dot product

CORRELATION	
corrcoef	Correlation coefficients
cov	Covariance matrix
subspace	Angle between subspaces

FILTERING AND CONVOLUTION	
filter	One-dimensional digital filter
filter2	Two-dimensional digital filter
conv	Convolution and polynomial multiplication
conv2	Two-dimensional convolution
deconv	Deconvolution and polynomial division

FOURIER TRANSFORMS	
fft	Discrete Fourier transform
fft2	Two-dimensional discrete Fourier transform
ifft	Inverse discrete Fourier transform
ifft2	Two-dimensional inverse discrete Fourier transform
abs	Magnitude
angle	Phase angle
unwrap	Remove phase angle jumps across 360 deg boundaries
fftshift	Move zeroth lag to center of spectrum
cplxpair	Sort numbers into complex conjugate pairs
nextpow2	Next higher power of 2

6.10 Polynomial and interpolation functions

Extracted from `help polyfun`.

POLYNOMIALS	
roots	Find polynomial roots
poly	Construct polynomial with specified roots
polyval	Evaluate polynomial
polyvalm	Evaluate polynomial with matrix argument
residue	Partial-fraction expansion (residues)
polyfit	Fit polynomial to data
polyder	Differentiate polynomial
conv	Multiply polynomials
deconv	Divide polynomials

DATA INTERPOLATION	
interp1	1-D interpolation (1-D table lookup)
interp2	2-D interpolation (2-D table lookup)
interpft	1-D interpolation using FFT method
griddata	Data gridding

SPLINE INTERPOLATION	
spline	Cubic spline data interpolation
ppval	Evaluate piecewise polynomial

6.11 Graphics

Extracted from `help plotxy`, `help plotxyz`, `help graphics` and `help color`.

ELEMENTARY X-Y GRAPHS.	
plot	Linear plot
loglog	Log-log scale plot
semilogx	Semi-log scale plot
semilogy	Semi-log scale plot
fill	Draw filled 2-D polygons

SPECIALIZED X-Y GRAPHS	
polar	Polar coordinate plot
bar	Bar graph
stem	Discrete sequence or "stem" plot
stairs	Stairstep plot
errorbar	Error bar plot
hist	Histogram plot
rose	Angle histogram plot
compass	Compass plot
feather	Feather plot
fplot	Plot function
comet	Comet-like trajectory

GRAPH ANNOTATION	
title	Graph title
xlabel	X-axis label
ylabel	Y-axis label
zlabel	Z-axis label for 3-D plots
text	Text annotation
gtext	Mouse placement of text
grid	Grid lines

3-D OBJECTS	
cylinder	Generate cylinder
sphere	Generate sphere

6.11.2 General purpose graphics functions

FIGURE WINDOW CREATION AND CONTROL	
figure	Create Figure (graph window)
gcf	Get handle to current figure
clf	Clear current figure
close	Close figure

GRAPH ANNOTATION	
title	Graph title
xlabel	X-axis label
ylabel	Y-axis label
text	Text annotation
gtext	Mouse placement of text
grid	Grid lines

6.11.1 Three dimensional graphics

LINE AND AREA FILL COMMANDS	
plot3	Plot lines and points in 3-D space
fill3	Draw filled 3-D polygons in 3-D space
comet3	3-D comet-like trajectories

CONTOUR AND OTHER 2-D PLOTS OF 3-D DATA	
contour	Contour plot
contour3	3-D contour plot
clabel	Contour plot elevation labels
contourc	Contour plot computation (used by contour)
pcolor	Pseudocolor (checkerboard) plot
quiver	Quiver plot

SURFACE AND MESH PLOTS	
mesh	3-D mesh surface
meshc	Combination mesh/contour plot
meshz	3-D Mesh with zero plane
surf	3-D shaded surface
surfc	Combination surf/contour plot
surfl	3-D shaded surface with lighting
waterfall	Waterfall plot

VOLUME VISUALIZATION	
slice	Volumetric visualization plots

GRAPH APPEARANCE	
view	3-D graph viewpoint specification
viewmtx	View transformation matrices
hidden	Mesh hidden line removal mode
shading	Color shading mode
axis	Axis scaling and appearance
caxis	Pseudocolor axis scaling
colormap	Color look-up table

AXIS CREATION AND CONTROL	
subplot	Create axes in tiled positions
axes	Create axes in arbitrary positions
gca	Get handle to current axes
cla	Clear current axes
axis	Control axis scaling and appearance
caxis	Control pseudocolor axis scaling
hold	Hold current graph

HANDLE GRAPHICS OBJECTS	
figure	Create figure window
axes	Create axes
line	Create line
text	Create text
patch	Create patch
surface	Create surface
image	Create image
uicontrol	Create user interface control
uimenu	Create user interface menu

HARDCOPY AND STORAGE	
print	Print graph or save graph to file
printopt	Configure local printer defaults
orient	Set paper orientation
capture	Screen capture of current figure

MOVIES AND ANIMATION	
moviein	Initialize movie frame memory
getframe	Get movie frame
movie	Play recorded movie frames

MISCELLANEOUS	
ginput	Graphical input from mouse
ishold	Return hold state
graymon	Set graphics window defaults for gray-scale monitor
rbbox	Rubberband box
rotate	Rotate an object about a specified direction
terminal	Set graphics terminal type
uiputfile	Put up dialog box for saving files
uigetfile	Put up dialog box which queries for file names
whitebg	Set graphics window defaults for white background
zoom	Zoom in and out on a 2-D plot
waitforbuttonpress	Wait for key/buttonpress over figure

6.11.3 Color control and lighting model functions

COLOR CONTROLS	
colormap	Color look-up table
caxis	Pseudocolor axis scaling
shading	Color shading mode

COLOR MAPS	
hsv	Hue-saturation-value color map
gray	Linear gray-scale color map
hot	Black-red-yellow-white color map
cool	Shades of cyan and magenta color map
bone	Gray-scale with a tinge of blue color map
copper	Linear copper-tone color map
pink	Pastel shades of pink color map
prism	Prism color map
jet	A variant of HSV
flag	Alternating red, white, blue, and black color map

COLOR MAP RELATED FUNCTIONS	
colorbar	Display color bar (color scale)
hsv2rgb	Hue-saturation-value to red-green-blue conversion
rgb2hsv	Red-green-blue to hue-saturation-value conversion
contrast	Gray scale color map to enhance image contrast
brighten	Brighten or darken color map
spinmap	Spin color map
rgbplot	Plot color map

LIGHTING MODELS	
surfl	3-D shaded surface with lighting
specular	Specular reflectance
diffuse	Diffuse reflectance
surfnorm	Surface normals

6.12 Sound processing functions

Extracted from `help sounds`.

GENERAL SOUND FUNCTIONS	
sound	Convert vector into sound
saxis	Sound axis scaling

COMPUTER-SPECIFIC SOUND FUNCTIONS	
auwrite	Write mu-law encloded audio file
auread	Read mu-law encloded audio file
wavwrite	Write MS Windows .WAV audio file
wavread	Read MS Windows .WAV audio file
mu2lin	Mu-law to linear conversion
lin2mu	Linear to mu-law conversion

6.13 Character string functions

Extracted from `help strfun`.

GENERAL	
strings	About character strings in MATLAB
abs	Convert string to numeric values
setstr	Convert numeric values to string
isstr	True for string
blanks	String of blanks
deblank	Remove trailing blanks
str2mat	Form text matrix from individual strings
eval	Execute string with MATLAB expression

STRING COMPARISON	
strcmp	Compare strings
findstr	Find one string within another
upper	Convert string to uppercase
lower	Convert string to lowercase
isletter	True for letters of the alphabet
isspace	True for white space characters
strrep	Replace a string with another
strtok	Find a token in a string

STRING TO NUMBER CONVERSION	
num2str	Convert number to string
int2str	Convert integer to string
str2num	Convert string to number
mat2str	Convert matrix to string
sprintf	Convert number to string under format control
sscanf	Convert string to number under format control

HEXADECIMAL TO NUMBER CONVERSION	
hex2num	Convert hex string to IEEE floating point number
hex2dec	Convert hex string to decimal integer
dec2hex	Convert decimal integer to hex string

6.14 Low-level file I/O functions

Extracted from `help iofun`.

FILE OPENING AND CLOSING	
fopen	Open file
fclose	Close file

UNFORMATTED I/O	
fread	Read binary data from file
fwrite	Write binary data to file

FORMATTED I/O	
`fscanf`	Read formatted data from file
`fprintf`	Write formatted data to file
`fgetl`	Read line from file, discard newline character
`fgets`	Read line from file, keep newline character

FILE POSITIONING	
`ferror`	Inquire file I/O error status
`feof`	Test for end-of-file
`fseek`	Set file position indicator
`ftell`	Get file position indicator
`frewind`	Rewind file

STRING CONVERSION	
`sprintf`	Write formatted data to string
`sscanf`	Read string under format control

WK1 FORMAT	
`wk1const`	WK1 record definitions
`wk1read`	Read WK1 file/range
`wk1write`	Write out matrix in a WK1 formatted file
`wk1wrec`	Write a WK1 record header

CSV FORMAT	
`csvread`	Read Comma Separated Value formatted file into a matrix
`csvwrite`	Write out matrix in a CSV formatted file

ASCII DELIMITED FORMAT	
`dlmread`	Read ASCII delimited file into a matrix
`dlmwrite`	Write out matrix in ASCII delimited file format

7 Web links

This document is based on Eleftherios Gkioulekas' tutorial *Programming with Matlab* available at
`www.amath.washington.edu/~elf/tutorials/`

The present version (what you are reading right now) was adapted from Gkioulekas' tutorial by Gjerrit Meinsma. It is available at
`www.math.utwente.nl/~meinsma/courses/`

Other useful web links are

`www.math.washington.edu/~burke/crs/515/`

`ftp://ftp.math.ufl.edu/pub/matlab/`

`www.mathworks.com/`