# An Introductory Guide to MATLAB

Ian Cavers
Department of Computer Science
University of British Columbia

## 1 Introduction

MATLAB provides a powerful interactive computing environment for numeric computation, visualization, and data analysis. Its wide range of commands, functions, and language constructs permit users to solve and analyze difficult computational problems from science and engineering without programming in a general purpose language.

This document is not intended to be a complete manual for MATLAB users. Instead, it provides a brief introduction to MATLAB, outlining features that might be of particular use to CPSC 303 students. Additional sources of MATLAB documentation and examples to be used in conjunction with this guide include:

**On-Line Help:** MATLAB provides access to extensive on-line documentation with the commands `help` and `lookfor`. To learn more about these commands enter `help help <rtn>` and `help lookfor <rtn>` at the MATLAB prompt. (Section 2 describes how to initiate a MATLAB session.) You will find `help` an valuable supplement to the information in this guide.

**MATLAB Manuals:** Two sets of MATLAB manuals are available for reference use in the CICSR/CS reading room (CICSR/CS 262). Please note that these manuals may NOT be taken out of the reading room. In addition, a third set of MATLAB manuals is on course reserve in the Sedgewick library.

**MATLAB Primer:** The second edition of the *MATLAB Primer* by Kermit Sigmon (University of Florida) is available in the file */ugrad0/cs303/lib/primer35.ps*. You can browse through this document with any postscript previewer, for example *ghostview*. Sigmon's primer is similar in nature to this guide, but provides a more detailed discussion and additional examples of MATLAB features.

**MATLAB Demos:** Enter `demo <rtn>` at the MATLAB prompt to start up the *MATLAB Expo* and then use your mouse to select and interact with the different demos. Demonstrations are available for most of MATLAB's wide ranging features. To start, I suggest *visiting* the MATLAB option of the *MAIN MAP* to check out the demonstrations on Matrices, Numerics and Visualization. If you have the opportunity to use color X-terminal many demonstrations are especially interesting.

In addition, you can also enter `intro <rtn>` at the MATLAB prompt for a tour of MATLAB lasting a few minutes. (This demonstration is controlled by the keyboard, rather than the mouse.)

**MATLAB FAQ:** The URL of MATLAB's FAQ (most **f**requently **a**sked **q**uestions) for Matlab 4 is `http://www.mathworks.com/faq.html`. Choose "Matlab 4 FAQ" at the top right of the page.
This extensive FAQ contains some interesting notes, but you may have to sift through answers to questions of little interest to you at the moment. More advanced MATLAB users, however, may find some of the information very helpful.

I have also provided an unofficial MATLAB FAQ in the text file */ugrad0/cs303/lib/FAQ*.

Each 303 assignment will require some use of MATLAB. The extent to which you use MATLAB beyond these requirements is up to you. For those who would like to explore the full potential of MATLAB, its "programming language" permits one to largely avoid languages such as C, Pascal or FORTRAN, for example. One warning, however. You may discover built-in numerical functions that you think implement a portion of one or more of your assignments. You may not use these function unless I approve them. (Of course, you could always verify your solution if you discover an appropriate MATLAB function.) Please check with me well before assignment due dates if you need this rule clarified for special MATLAB functions.

We have approximately 25 or 30 MATLAB licenses available to Rick Lab users. Please respect the needs of other users and exit from MATLAB rather than leaving it idle for extended periods of time. To save the current state of a MATLAB session and initialize a subsequent MATLAB session with the same variables, use the commands `save` and `load`.

*While reading the remainder of this document you are strongly encouraged to go to Rick's Lab and experiment with the different features described. Be sure to explore some of the* MATLAB Expo *demos too!*

## 2 MATLAB Basics

To initiate a MATLAB session in Rick's Lab simply enter `matlab` at the shell prompt of an *xterm*. MATLAB will display some introductory information before printing the MATLAB prompt `>>`. (Hint: Due to difficulties with the license server, MATLAB may not always start the first time you try it, even though licenses are available. Before you conclude all licenses are in use, try to start MATLAB several ($> 5$) times.) At the prompt users are free to enter sequences of MATLAB expressions, commands and assignment statements. (See Section 3.) MATLAB places all ASCII output in the *xterm* window from which MATLAB was invoked, but automatically creates additional windows for graphical output.

To recall or edit previous command lines MATLAB provides command line editing using Emacs-style keystrokes. Using the $\uparrow$ and $\downarrow$ keys you can cycle through previous commands. The $\leftarrow$ and $\rightarrow$ keys move the cursor left and right along the current command line. In its default mode MATLAB inserts characters before the cursor. `Cntrl-T` toggles the command line editor between the insert and over-write modes. The *delete* key eliminates the character lying under the cursor. Once all necessary corrections (if any) have been made to the command line, it can be entered by hitting `<rtn>` with the cursor in any position. For more information consult the `cedit` help page by entering

```
>> help cedit.
```

To terminate a MATLAB session enter the command

```
>> exit
```

or

```
>> quit
```

## 3 Variables, Expressions and Statements

MATLAB statements typically take one of two forms:

*variable* = *expression* or
*expression*

All variable (and function) names consist of a letter followed by any number of numbers, letters and underscores. MATLAB is case sensitive and only the first 19 characters of any name are significant.

Expressions are composed from operators, function calls and variable names. A carriage return normally signifies the end of a statement, causing MATLAB to interpret the command and print its result. If the last character of a statement is a ; (semicolon), however, display of the result is suppressed. This feature may be especially useful when the result of a computation is a large matrix. Finally, several statements separated by commas may be placed on a single line.

When an expression is not explicitly assigned to a variable with the assignment operator (=), MATLAB automatically stores the result in the special variable `ans`.

During a MATLAB session you may forget the names of variables stored in your *workspace*. The command `who` lists the name of all your variables. If you want to know their size as well, use the command `whos`. By default MATLAB stores all variables until the session is terminated. To remove a variable from the workspace use the command `clear var_name`. WARNING: `clear` with no arguments removes all variables from the workspace.

At any time you can interrupt the computation of a MATLAB statement with `Cntrl-C`.

## 4 Matrices and MATLAB

Essentially, the only data objects in MATLAB are rectangular numerical matrices. There are no restrictions placed on the dimensions of a matrix (except by system resources), but special meaning is sometimes attached to $1 \times 1$ matrices (scalars) and matrices with only one row or column (vectors). The memory required for the storage of each matrix is automatically allocated by MATLAB.

The easiest way to enter a matrix into MATLAB is to provide an explicit list of elements enclosed in square brackets [ ]. MATLAB uses the following conventions:

- A matrix element can be any valid MATLAB expression.
- Matrix elements are separated by blanks or commas.

- A semicolon or carriage return is used to indicate the end of a row.

For example, entering the assignment statement
```
>> A = [1 2 4.5; 8/2.0 6 5]
```
results in the output

```
A =
    1.0000    2.0000    4.5000
    4.0000    6.0000    5.0000
```

The $2 \times 3$ matrix is saved in variable A for future reference. If you want to see the contents of this or any other variable, simply enter its name as a command. To reference individual elements enclose their subscripts in parentheses after the variable name in the usual fashion.
```
>> A(2,3)

ans =
     5
```

It is important to realize that MATLAB distinguishes between row and column vectors. `[1 2 3]` is a row vector, while `[1; 2; 3]` is a column vector. Column vectors can also be created by applying MATLAB's transpose operator `'` (prime) to a row vector. For example
```
>> [1 2 3]'
```
produces

```
ans =
     1
     2
     3
```

The transpose operator may be applied to matrices of any dimension.

# 5 Numbers and Arithmetic Expressions

MATLAB uses conventional decimal notation to enter numbers. The leading minus sign and decimal point are optional, and numbers may be specified using scientific notation. The following examples are valid numbers in MATLAB.

| | | |
|---|---|---|
| 4 | -189 | .032 |
| -2.21e-23 | 0.972E18 | 0.000001 |

WARNING: If you try to import data produced by a FORTRAN routine that prints the values of double precision variables, MATLAB will not understand the use of "D" in place of "E" in scientific notation. (You can use the Unix command `tr` to replace all "D"s with "E"s.)

To build expressions, MATLAB provides the usual arithmetic operators.

| | | | | | |
|---|---|---|---|---|---|
| $+$ | addition | $-$ | subtraction | * | multiplication |
| / | right division | \ | left division | ^ | power |

(MATLAB's use of two division operators will be explained in Section 7.) To change the normal precedence of these operators enclose portions of an expression in parentheses in the usual manner.

On the Rick Lab's HP servers MATLAB uses IEEE double precision floating point arithmetic to perform its computations. Although MATLAB stores the full precision of all computations, by default it displays results in a 5 digit fixed point format. The output format can be changed using the `format` command. The following example prints the result of entering the vector `[5/7.3 7.7432e-7]` under different format settings.
```
format short
```

```
      0.6849    0.0000
format short e
      6.8493e-01   7.7432e-07
format long
      0.68493150684932   0.00000077432000
format long e
      6.849315068493150e-01     7.743200000000001e-07
```

You can learn more about the format command by entering `help format`. The remainder of the examples in this guide assume that the "`short e`" format has been chosen.

# 6 MATLAB Functions

In addition to the standard arithmetic operators, MATLAB provides an extensive collection of built-in functions. For example, most elementary mathematical functions (`sin, cos, log, sqrt,`...) are available.

```
>> cos(pi/4)

ans =
   7.0711e-01
```

`pi` is an example of a function that does not require parameters and simply returns a commonly used constant.

```
>> pi

ans =
     3.1416
```

Other functions are available in libraries of *M-files* grouped into *toolboxes*. Section 11 briefly describes how to create your own functions.

So far we have only seen functions that return a single matrix, but some functions return two or more matrices. To save these matrices, we surround the output variables by brackets [ ] and separate them by commas. For example,

```
>> [V,D] = eig(A)
```

returns the eigenvectors and eigenvalues of A in matrices `V` and `D`.

Several additional MATLAB functions are described in the remainder of this guide. For extensive lists of MATLAB functions consult the *MATLAB primer* or explore the list of topics provided by entering the command `help` with no arguments. In addition, the MATLAB demos provide examples of the use of many interesting functions.

# 7 Matrix Operations

We have already seen the matrix operator ' (prime) for transposing matrices. The arithmetic operators presented in Section 5 also operate on matrices. In each case the operator behaves in a manner consistent with standard linear algebra practises.

The operators $+$ and $-$ permit the addition and subtraction of matrices, and are defined whenever the matrices have the same dimension. For example, the following is a valid expression.

```
>> [1 2 3; 4 5 6] + [3 2 1; 1 1 1];
```

The exception to this rule is the addition (subtraction) of a scalar to (from) a matrix. In this case the scalar is added to or subtracted from each element of the matrix individually.

```
>> [1 2 3] + 1

ans =
      2     3     4
```

The multiplication of two matrices, denoted by `A*B`, is defined whenever the inner dimensions of the operands `A` and `B` are equal. For example, if

`C=[1 2 3; 4 5 6], D=[1 1 1; 2 2 2], x=[1 1 1]'`
then `C*x`, `x'*x` (an inner product), `x*x'` (an outer product) and `C*D'` are defined, but `C*D` is not. (Give these examples a try and be sure you understand how MATLAB interprets them.) In the special case when one of the operands is a scalar, each element of the matrix is multiplied by the scalar.

It is now time to explain the reason for MATLAB's two division operators. If `A` is a square nonsingular matrix then `A\B` and `B/A` formally correspond to the left and right multiplication of `B` by `A`$^{-1}$. (Note: MATLAB does not actually compute the inverse of A when evaluating these expressions.) These expressions are used to solve the following types of systems of equations.

**left division:** `x = A\B` solves $A * X = B$

**right division:** `x = A/B` solves $X * A = B$

In CPSC 303 we will typically use left division. Whenever `B` has as many rows as `A`, left division is defined. If `A` is a square nonsingular matrix and b is a vector with as many rows, MATLAB evaluates the expression `x = A\b` (the solution to `Ax = b`) by factoring `A` with Gaussian elimination and then solving two triangular systems to compute `x`. When $A$ is not square, MATLAB factors `A` using Householder orthogonalization and the factors are used to solve the under-determined or over-determined system of equations in the least squares sense. This can lead to surprising results if the wrong slash is used or if the dimensions of your matrices are wrong.

Finally, the expression `A^p` raises `A` to the `p`$^{th}$ power. This operation is only defined if `A` is square and `p` is an scalar. For example, `A^2` is equivalent to `A*A`, although MATLAB does not always compute powers with simple matrix multiplication.

# 8 Array Operations

The previous section described standard linear algebra matrix operations. Alternatively, element-by-element matrix arithmetic is provided by *array operations*. An array operator is formed by preceding one of the symbols +, −, *, \, or / by a period (.). Of course, the matrix and array operators for addition and subtraction are equivalent, and + and − are used in either case.

The operator `.*` denotes array multiplication and is defined whenever the two operands have the same dimensions. For example
`>> A=[1 2 3; 4 5 6] ; B = [2 2 2; 3 3 5]; C=A.*B`
results in

```
C =
     2     4     6
    12    15    30
```

Similarly, `A.\B` and `A./B` provide the left and right element-by-element division. Raising each element of a matrix to the same power is accomplished by the `.^` operator.

Most standard MATLAB functions operate on a matrix element-by-element. For example
`>> cos(C)`

```
ans =
  -4.1615e-01  -6.5364e-01   9.6017e-01
   8.4385e-01  -7.5969e-01   1.5425e-01
```

If you create your own functions (See Section 11.) you should keep in mind that MATLAB assumes that a matrix (or a vector) can be passed to it.

# 9 Vector and Matrix Manipulation

Vectors are easily generated with MATLAB's colon ":" notation. For example, the expression `1:5` creates the following row vector.

```
     1      2      3      4      5
```
You can also create a vector using an increment other than one. For example, `1:2:7` results in the vector
```
     1      3      5      7
```
The increment may be negative and need not be an integer.

It is very easy to create a table using the colon notation. Experiment with the commands
```
>> x=(0:pi/4:pi)'; y=cos(x); AA=[x y]
```
for a demonstration of this technique.

MATLAB permits users to easily manipulate the rows, columns, submatrices and individual elements of a matrix. The subscripts of matrices can be vectors themselves. If `x` and `v` are vectors then `x(v)` is equivalent to the vector `[x(v(1)), x(v(2)), ...]` . Subscripting a matrix with vectors extracts a submatrix from it. For example, suppose `A` is an $8 \times 8$ matrix. `A(1:4, 5:8)` is the $4 \times 3$ submatrix extracted from the first 4 rows and last 3 columns of the matrix. When the colon operator is used by itself, it denotes all of the rows or columns of a matrix. Using the result of the table above
```
>> AA(:,1)
```
produces the first column of matrix `AA`.

```
ans =
            0
   7.8540e-01
   1.5708e+00
   2.3562e+00
   3.1416e+00
```

# 10 For Loops, While Loops, and Conditional Statements

MATLAB also provides a programming language that includes looping statements (`for` and `while`), conditional statements (`if`), and relational and logical operators for constructing conditions.

The execution of an `if` or `while` depends upon the evaluation of a *condition*. To construct conditions, MATLAB provides six relational operators

| | | | | | |
|---|---|---|---|---|---|
| $<$ | less than | $\leq$ | less than or equal | $==$ | equal |
| $>$ | greater than | $\geq$ | greater than or equal | ~= | not equal |

and three logical operators.

| | | | | | |
|---|---|---|---|---|---|
| & | and | \| | or | ~ | not |

Note that the relational operator $==$ compares two arguments, while $=$ is used to assign a value to a variable.

When a condition compares scalar quantities it evaluates to 1 if true and 0 if false. Relational operators can also be applied to matrices of the same dimension. In this case a condition evaluates to a matrix of 0s and 1s representing the result of applying the operator to the individual elements of the matrix operands. A condition evaluating to a matrix is interpreted by `if` or `while` to be true if each element of the matrix is nonzero. (The MATLAB function `any` is useful when you want a different interpretation of the matrix resulting from the evaluation of a condition.)

The simplest form of an `if` statement is

```
if condition
  statements
end
```

For example,

```
>> A = [6 3 9];
>> n = 3;
>> if n ~= 0
```

```
      A = A/n;
   end
>> A
```

results in the output

```
  A =
      2     1     3
```

(Each semicolon in the previous example suppresses the output of a statement's result.)

As in other programming languages, `if` statements can also be used to choose between 2 or more alternatives. For example, the statement

```
>> if n < 0
      x = [x,abs(n)];
   elseif (rem(n,2) == 0) & (n ~= 0)
      x = [x,n/2];
   else
      x = [x,n+1];
   end
```

adds one of three possible elements to the end of the existing row vector `x`, depending on the value of scalar `n`. Note that `elseif` is one word. Writing `else` and `if` separately would create a second `if` statement requiring its own `end`.

When working with matrices and vectors, the ability to repeat statements with `for` and `while` statements is essential. The general form of the `for` statement given below permits the statements in its body to be repeated a specific number of times.

`for` *variable* = *expression*
  statements
`end`

The columns of *expression* are assigned to *variable* one by one and then the statements are executed. As an example, the following statements construct a vector containing the squares of the integers 2 through 5.

```
>> x = [];
>> low = 2;
>> hi = 5;
>> for i = low:hi
      x = [x, i*i];
   end
```

Of course, you could also write this `for` statement on a single line by separating its components by commas,

```
>> for i = low:hi, x = [x,i*i], end
```

but using separate lines with appropriate indentation usually improves readability. The following statement produces a vector with the same elements as `x`, but they are arranged in the reverse order.

```
>> y = [];
>> for i = hi:-1:low
      y = [y, i*i];
   end
```

You can also nest `for` statements. The following statements create an $m \times n$ Hilbert matrix, `H`.

```
>> for i = 1:m
     for j = 1:n
       H(i,j) = 1/(i+j-1)
     end
   end
```

Finally, MATLAB also has its own version of the while loop, which repeatedly executes a group of statements while a condition remains true.

`while` *condition*
  statements
`end`

Given a positive number `val` the following statements compute and display the even powers of 2 less than or equal to `val`.

```
>> n = 0;
>> while 2^n <= val
     if rem(n,2) == 0
       2^n
       n = n + 1;
     else
       n = n + 1;
     end
   end
```

(The previous example increments `n` by 1 each iteration (instead of by 2) and exhibits other elements of poor programming style so that we can illustrate the nesting of an `if` statement inside a `while`.)

The commands discussed in this section, together with the discussion of scripts and functions in the following section, permit students to do most of their 303 work within MATLAB.

# 11 M-Files: Creating Your Own Scripts and Functions

Users are able to tailor MATLAB by creating their own functions and scripts of MATLAB commands and functions. Both scripts and functions are ordinary ASCII text files external to MATLAB. The name of each file must end in "`.m`" and must be found on MATLAB's search path. (It is easiest to start MATLAB from the directory containing your M-files. See also the command `path`.) By default while an M-file is executing its statements are not displayed. This default behavior can be changed using the `echo` command.

A script may contain any sequence of MATLAB statements, including references to other M-files. It is invoked like any other command without arguments and acts on the variables of the workspace globally. Each command in the file is executed as though you had typed it into MATLAB. Most of the demos provided by MATLAB are simply scripts.

The first line of a function M-file starts with the word *function* and declares the name of the function and its input and output parameters. All input parameters and variables defined within the file are local to the function. Figure 1 provides a simple example from the *MATLAB User's Guide*. If we type this function into a file called mymean.m, then we can call `mymean` like any other MATLAB function.

```
>> mymean(1:99)

ans =
     50
```

The parameter supplied to mymean is passed by value (not reference) to the local variable `x`. The variables `m,n,y` are also local to `mymean` and do not exist after it returns. An M-file function does not alter the value of variables in the workspace unless explicit *global* declarations are made.

```
function y = mymean(x)
% mymean:  Average or mean value.
% For vectors, mymean(x) returns the mean value.
% For matrices, mymean(x) is a row vector containing the mean value of
each column.
[m,n] = size(x);
if m == 1
      m = n;
end
y = sum(x)/m;
```

Figure 1: A User Defined Function.

The MATLAB function `eval` accepts a single string argument, which it interprets as a MATLAB expression or statement. The following example demonstrates how you can use `eval` to pass one function to second function as an argument and have the second function call the first function within its body.

Suppose the following function is in the M-file `goo.m`

```
function val = goo(x)
val = x.^2;
```

and the following function is in the M-file `foo.m`.

```
function val = foo(fname, a)
% calls another function, whose name is passed as a string to
% argument fname, with the parameter a
val = eval([fname,'(a)']);
%          ^^^^^^^^^^^  Constructs a string interpreted as an expression
%                       or statement by eval.
```

Using these functions, the following command

```
>> foo('goo', 3)
```

evaluates to

```
ans =
     9
```

# 12 Graphics and Related Issues

MATLAB provides users with a powerful array of functions for creating sophisticated graphics. In CPSC 303 we will probably only use the 2-D plotting features, but you should familiarize yourself with the breadth of MATLAB's capabilities.

For our work the two plotting functions `plot` and `fplot` will be especially useful. `plot` permits the plotting of vectors and matrices, while `fplot` simplifies the plotting of functions.

Let's start by plotting sin between 0 and pi using `plot`. First we must create two vectors containing the x and the y coordinates to the points we will use to create our plot.

```
>> x=0:pi/100:2*pi;
>> y=sin(x);
```

To create our plot of sin we enter the following statement.

```
>> plot(x,y,'-')
```

The minus sign in single quotes tells MATLAB we want a solid line. Other line types (and colors) can be specified in this fashion. Enter `help plot` for more information.

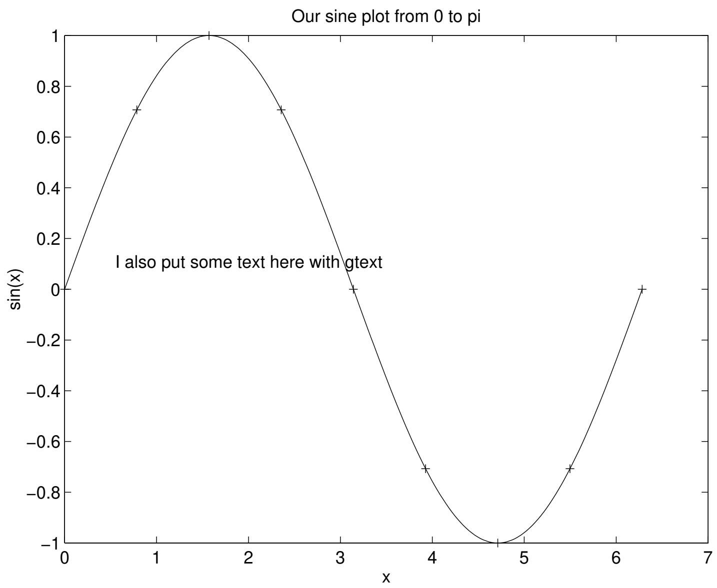


Figure 2: A Plotting Example.

MATLAB permits the addition of another plot to the same graph by entering the command
`>> hold on`
Let's also plot the value of sin at 0,pi/4,pi/2,. . .,2∗pi on our graph, but this time instead of a solid line we will just mark each point with a plus sign.
`>> plot(0:pi/4:2*pi,sin(0:pi/4:2*pi),'+')`
Perhaps we also want to add a title and axis labels to the graph.

```
>> title('Our sine plot from 0 to pi')
>> xlabel('x')
>> ylabel('sin(x)')
```

Text can also be added to a plot at user specified coordinates with the commands `text` and `gtext`. Your final graph should look something like Figure 2. When you are finished with the graph enter the command
`>> hold off`
so that subsequent graphics do not include the current plot.

The MATLAB command `fplot` is similar to `plot` but accepts the name of a function in single quotes and an abscissa range. It adaptively samples the function at enough points to give a representative graph and then plots the results. As an example, let's plot sin again.
`>> fplot('sin', [0,2*pi])`

To create a hardcopy of the current figure we use the MATLAB command `print`. For example, to create an PostScript version of our figure enter the following command.
`>> print file_name.ps -dps`
The resulting file can be printed on any PostScript printer using the Unix command *lpr* or previewed using *ghostview*. To create encapsulated PostScript suitable for inclusion into documents substitute `-deps` for `-dps` and `file_name.eps` for `file_name.ps`. (WARNING: PostScript files can be quite large and eat up a lot of your disk quota.)

In CPSC 303 we may often create data outside MATLAB and want to import it for plotting. This is accomplished by placing the data pairs (or triples) into an ASCII file with each pair (or triple) of data points on a separate line separated by blanks. (In other words, each vector of data to be imported is placed in a separate column of the file.) Be sure that numbers in scientific notation are compatible with MATLAB's format. (See Section 5.)

To import the data enter the following command.

```
>> load file_name
```

Variable `file_name` is now a matrix containing your data. To check if it was properly imported, simply enter `file_name`. To extract the vectors of data from the columns of the matrix use the colon notation as previously discussed.

# 13 Additional Features

This brief guide has not touched upon many interesting MATLAB features. MATLAB provides many additional functions that you can explore in the MATLAB manuals or using MATLAB's help facility. If you intend to create complex MATLAB functions check out the description of MATLAB's debugger in the User's Manual.