MATLAB Basics

© C.R. Thomas 2006
What is MATLAB?

MATLAB is a powerful tool for mathematical computations

It has extensive capabilities for generating graphs

It is used routinely by many engineers for solving modelling problems
MATLAB can be used interactively or programs can be written for later execution

Error checking is very good – syntax errors are identified as code is written and there are very good diagnostic messages for program logic errors

Complete solutions to problems can be written very quickly
Starting MATLAB

In the clusters, you should find a shortcut on your desktop

When MATLAB opens, you are presented with the MATLAB Desktop
To get started, select MATLAB Help or Demo from the Help menu.
The Desktop should contain several embedded windows of which the most important is the Command Window.

This is where you can type commands i.e. instructions to the computer.
titlebar is blue meaning the window is active so now you can type commands
To get started, select MATLAB Help or Demos from the Help menu.

```
>> 2+2
```

**NB:** normally extra spaces are ignored
To get started, select MATLAB Help or Demos from the Help menu.

```
>> 2+2
ans =
    4
>> |
```

press Enter

and back comes the answer
There are lots of built in commands e.g.

```matlab
>> factor(23370569)
ans =
    37   191  3307
```

If you don’t give your answer a name, it is called `ans`

MATLAB assigns a new answer to `ans` with each calculation
If you want to keep your answer for later, give it a name

This is called assignment

CRT: Basic MATLAB
The Workspace window tells you what you have stored at present.
Once it has a name, you can use it again and again.
This is a history of what you have done - in the Command History window.
There is lots of on-line help available including video tutorials and demos

CRT: Basic MATLAB
You can also get help on a particular command

```
>> help factor
FACTOR Prime factors.
    FACTOR(N) returns a vector containing the prime factors of N.

    This function uses the simple sieve approach. It may require large
    memory allocation if the number given is too big. Technically
    it is possible to improve this algorithm, allocating less
    memory for most cases and resulting in a faster execution
    time. However, it will still have problems in the worst
    case, so we choose to impose an upper bound on the input number
    and error out for n > 2^32.

    See also primes, isprime.

    Overloaded functions or methods (ones with the same name in other directories)
       help sym/factor.m

    Reference page in Help browser
       doc factor
```

CRT: Basic MATLAB
Simple arithmetic
- just like a calculator!

+ add
- subtract
* multiply
/ divide
^ exponentiate (power)

Use brackets as necessary

CRT: Basic MATLAB
\[
\text{ans =}
\]

17
Making and fixing errors

If you make a syntax error in typing your command MATLAB will print an error message

\texttt{>> 2a}

???

\begin{verbatim}
| Error: Missing MATLAB operator.
\end{verbatim}

\texttt{>> 2*a} would be correct
Semicolon

In MATLAB, one use of a semicolon (;) is to suppress output to the screen (Command Window) e.g.

```matlab
>> x = 3
x =     
     \{output to screen\}
     3

but
```

```matlab
>> x = 3;   gives no output to the screen
```

CRT: Basic MATLAB
Note the result will still be stored in the Workspace for later use.

This use of the semicolon is common when writing programs, or if one is generating a large variable at the command line (see later).
Digression: scalars, vectors, arrays and matrices

A scalar quantity is one that is defined by a single number – its size or magnitude (with appropriate units)

Example: a speed of 100 km h\(^{-1}\)
A vector has magnitude and direction

Example:
a velocity of 100 km h\(^{-1}\) due South
If you think about direction in coordinates, you will realise that a vector can also be considered an ordered list of numbers e.g.

the direction is 1 unit along the $x$ axis, 2 units along the $y$ axis, and 2 unit along the $z$ axis
As long as we know what our base directions are (x, y and z) we could describe the vector \( \mathbf{r} \) as \( \mathbf{r} = [1, 2, 2] \)
Arrays

An array is a collection of objects (elements), of identical type, in a rectangular arrangement

An array of ?

CRT: Basic MATLAB
Matrices

A matrix is an array of numbers e.g.

\[
\begin{pmatrix}
-1 & 0 & 0 \\
1 & 1 & 0 \\
0 & -1 & 1
\end{pmatrix}
\]

... although not all arrays of numbers are matrices

MATLAB stands for MATrix LABoratory
A vector can be thought of as a matrix with only one row or one column

\[
\begin{pmatrix}
-1 & 0 & 0
\end{pmatrix}
\]

and a scalar as a matrix with only one “element”

\[
\begin{pmatrix}
-1 \\
1 \\
0
\end{pmatrix}
\]

CRT: Basic MATLAB
Assignment Statements

\[ x = 4 \quad (x \text{ is a scalar}) \]

Note that “=” in MATLAB is an assignment operator.

It is therefore perfectly OK to write

\[ \gg x = x + 1 \]

See A VERY, VERY, Brief Guide to MATLAB for a summary of MATLAB syntax.
\[ x = x + 1 \] would be incorrect in normal algebra but here means:
the (new) value of \( x \) becomes the (previous) value of \( x \) plus 1
or, more simply:
\[ x \text{ becomes } x + 1 \]
More Assignment Statements

\[ y = [2, 3] \quad y \text{ is a row vector} \]
\[ \text{i.e a matrix with only 1 row} \]

Creating a matrix:

\[ A = [-1, 0, 0; 1, 1, 0; 0, -1, 1] \]

\[ A \text{ is a } 3 \times 3 \text{ matrix} \]

\[ A = \begin{pmatrix}
-1 & 0 & 0 \\
1 & 1 & 0 \\
0 & -1 & 1
\end{pmatrix} \]

CRT: Basic MATLAB
Looking at the syntax more closely:

\[ A = [-1, 0, 0; 1, 1, 0; 0, -1, 1] \]

- A “,” is a divider, separating elements on a row.
- The “;” here separates rows i.e. starts a new row.
- (NB: a second use of “;”)

You can also use a space as an element divider e.g. \[ A = [-1 0 0; 1 1 0; 0 -1 1] \] which gives the same \( A \) as above.
This is a rectangular matrix with 3 rows and 2 columns. Its size is 3x2

In MATLAB: \( C = \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 2 & 5 \end{bmatrix} \)

\text{size}(C)\) is a MATLAB function that outputs the number of rows in \( nr \) and the number of columns in \( nc \) – can be very useful in handling matrices. Use it like: \( >> [nr \ nc] = \text{size}(C) \)

CRT: Basic MATLAB
\[ A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \]

\( b = A(3, 2) \) sets \( b \) to the element that is in the third row, second column of \( A \)

This is 8 in this case

You can also use this to assign values to elements e.g. \( A(3, 2) = 0 \) giving

\[ A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 9 \end{bmatrix} \]

The difference is that in the second case \( A(3, 2) \) is on the left hand side of “\( = \)”
Digression:
Matrix addition and subtraction

Matrix addition and subtraction behave as you might expect, as does multiplication by a scalar

\[
A = \begin{pmatrix} 2 & 3 \\ 1 & 7 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 3 \\ 0 & 1 \end{pmatrix}
\]

then \( A + B = \begin{pmatrix} 2+3 & 3+3 \\ 1+0 & 7+1 \end{pmatrix} = \begin{pmatrix} 5 & 6 \\ 1 & 8 \end{pmatrix} \)
Matrix multiplication does not work as you might expect

This is NOT how it is done:

\[
A = \begin{pmatrix} 2 & 3 \\ 1 & 7 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 3 \\ 0 & 1 \end{pmatrix}
\]

then \( A \times B \neq \begin{pmatrix} 2 \times 3 & 3 \times 3 \\ 1 \times 0 & 7 \times 1 \end{pmatrix} \)

Do not try to do this!

CRT: Basic MATLAB
Matrix multiplication is row $\times$ column

Each element of a row is multiplied by the corresponding element in a column, and the results are added to give one element of the new matrix.

This is located where the row and column intersect.

Hard to describe, easy to do ....
\[
A = \begin{pmatrix} 2 & 3 \\ 1 & 7 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 3 \\ 0 & 1 \end{pmatrix}
\]

then \( A \times B = \begin{pmatrix} (2 \times 3) + (3 \times 0) & (2 \times 3) + (3 \times 1) \\ (1 \times 3) + (7 \times 0) & (1 \times 3) + (7 \times 1) \end{pmatrix} \)

Similarly, if \( A = \begin{pmatrix} 1 & 3 & 2 \\ 1 & 2 & 4 \end{pmatrix} \) and \( B = \begin{pmatrix} 3 & 4 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \)

then \( A \times B = ? \)
A * B = \begin{pmatrix} 6 & 6 \\ 5 & 8 \end{pmatrix}

NB: we can only multiply matrices if the number of columns of the first matrix equals the number of rows of the second.

For example, we cannot evaluate \( A * B \) if 
\[
A = \begin{pmatrix} 1 & 3 & 2 \\ 1 & 2 & 4 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 3 & 4 \\ 1 & 0 \end{pmatrix}
\]
If A is a $n \times m$ matrix, and B is a $p \times q$ matrix, $A \times B$ only exists if $m = p$

If $m = p$, then the resulting matrix has dimensions $n \times q$

$n \times m, m \times q \rightarrow n \times q$

Remember from earlier:

\[
A = \begin{pmatrix}
1 & 3 & 2 \\
1 & 2 & 4
\end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix}
3 & 4 \\
1 & 0 \\
0 & 1
\end{pmatrix} \quad \text{then} \quad A \times B = \begin{pmatrix}
6 & 6 \\
5 & 8
\end{pmatrix}
\]

$2 \times 3, 3 \times 2 \rightarrow 2 \times 2$
A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
B = [1 0 1; 0 1 0; 1 1 0]

A*B in MATLAB is matrix multiplication

In this case
A*B = [4, 5, 1; 10, 11, 4; 16, 17, 7]

Note that in general A*B ≠ B*A, and
A*B = 0 does not imply either A or B is necessarily 0
A.*B in MATLAB is multiplication element by element

\[ A \cdot B = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 4 & 0 \\ 7 & 8 & 0 \end{pmatrix} \]

Similarly, A.^2 means square each element of A, but A^2 equals A*A
A = A' transposes A

Transpose means swap rows and columns

\[
A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 9 \end{pmatrix} \quad A' = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 0 \\ 3 & 6 & 9 \end{pmatrix}
\]

If \(A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 9 \end{pmatrix}\) then

\(A' = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 0 \\ 3 & 6 & 9 \end{pmatrix}\)

If \(y = [2, 3]\), then \(y' = [2; 3]\)
A = inv(A) gives the inverse of the matrix
inv(A)*A = I
where I is the “identity matrix”
The identity matrix behaves like the number 1 in arithmetic but might look like

\[
I = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

The size is variable and here would be the same as A
To find the inverse $A$ has to be "square" i.e. the same number of rows as columns

Also its determinant must not equal zero

Say $A \times x = b$, then

$\text{inv}(A) \times A \times x = I \times x = x = \text{inv}(A) \times b$

This could be used to solve systems of linear equations (for $x$ here), but it is usually more efficient for a computer to do Gaussian elimination
A\b is matrix division in MATLAB, used for solving sets of linear equations by Gaussian elimination

Say \( A \, x = b \)

Then in MATLAB: \( x = A\backslash b \)

e.g. \( >> A = [1 \, 1; \, 1 \, 4] \)
\( >> b = [1; \, 2.5] \)
\( >> A\backslash b \) gives \([0.500 \, 0.500]\)
Colon Operator
If a colon is used to separate two integers, it generates all the integers between them e.g.
>> c = 1:8
creates a vector c =

The step size can be defined e.g.
>> b = 0:2:8
creates a vector b = [0 2 4 6 8]
The step size can be negative for a countdown e.g.

```matlab
>> d = 2:-0.2:1
```

creates a vector `d` containing numbers dropping in steps of 0.2 from 2 to 1 inclusive

```
d = ?
```
Concatenation

Concatenation means creating larger matrices from smaller ones - not addition e.g. if

\[ A = \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix} \]

Then \( C = [A \ B] \) gives \( C = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 4 & 3 & 0 \end{bmatrix} \)

\[
C = \begin{pmatrix}
1 & 1 & 1 & 2 \\
1 & 4 & 3 & 0
\end{pmatrix}
\]

CRT: Basic MATLAB
On the other hand, with the same
$A = [1 \ 1; 1 \ 4]$ and $B = [1 \ 2; 3 \ 0]$

$C = [A; B]$ gives $C = [1 \ 1; 1 \ 4; 1 \ 2; 3 \ 0]$

CRT: Basic MATLAB
Special constants and values are often available in MATLAB

e.g. \texttt{pi} represents $\pi$

\begin{tabular}{l l}
  Inf & infinity \\
  NaN & not a number \\
\end{tabular}
Strings

MATLAB can handle strings i.e. bits of text. It does this by treating text as a matrix of characters. Use single quotes to show you are dealing with text e.g.

`>> message = 'Hello world'`
You can use concatenation to build more complex text e.g.

```matlab
>> big_message = [message; 'from Fred']
```

You can display your text on the screen using the function `disp`

```matlab
>> disp(big_message)
```

There are many ways to control screen output e.g. `fprintf`
MATLAB Functions

As in Excel, MATLAB provides lots of built-in functions for you to use
e.g. sqrt, exp, log, sin, cosh ....

```matlab
>> y=[1 2 3 4 5];
>> z=sqrt(y)
z =
    1.0000    1.4142    1.7321    2.0000    2.2361
```

This is “vectorisation” is one reason for the power of MATLAB
Plots

plot(x,y) produces a graph of y against x, where x and y are vectors

plot takes in data sets and outputs a plot or “figure”

e.g.  >> x = -pi:0.1:pi;
        >> y = sin(x);
        >> plot(x,y)
There are many ways to improve the look of your plots!

CRT: Basic MATLAB
Digression: Computer representation of numbers

Decimal: 123.45 means $1 \times 10^2$
+ $2 \times 10^1$
+ $3 \times 10^0$
+ $4 \times 10^{-1}$
+ $5 \times 10^{-2}$

Using scientific notation, this is written as $1.2345 \times 10^2$
Computers use an adaptation of scientific notation called “floating point” representation.

For example, in MATLAB:
123.45 becomes
1.2345e+002

*e+002 means \(10^2\)*
Computers represent numbers as a string of bits e.g. 53 binary digits. Only some (decimal) numbers can be represented exactly in a computer. Of course, internally computers work in binary i.e. powers of 2, not 10. In common "double precision" representation, consecutive numbers differ by about 1 part in $10^{16}$. The true mathematical result of a calculation might not be one of these.
This can result in numerical errors e.g.

```
>> 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2
ans =
  5.5511e-017
```

```
>> sin(pi)
ans =
  1.2246e-016
```

Most of the time, such errors in numerical calculations in MATLAB will be unimportant

CRT: Basic MATLAB
Matlab Script Files

Although a lot can be done from the command line, it is often useful to write a MATLAB program or “script”

A script is stored in a text file, with the extension .m - hence “m-files”

When you invoke a script by typing its name in the command line, it simply executes the commands in the file
Example: simplified version of `magicrank.m` from the “Getting Started“ tutorial

`magic(n)` makes a magic square of size `n`

```matlab
% investigate the rank of magic squares
for n = 3:32
    r(n) = rank(magic(n));
end
bar(r)
```

Loops through values of `n` from 3 to 32 making magic squares and storing the rank in a vector `r`

Bar chart of ranks

Comment ignored by the program

CRT: Basic MATLAB
>> magicrank ← at command line

CRT: Basic MATLAB
For simple problems, the command line is fast and efficient.

For larger problems, or if you wish to change variable values, or have loops or branches, or modify the commands, use script files.

Note that you can store your script files and reuse them in other work.
### Useful functions for script files:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disp(ans)</td>
<td>Displays results without identifying variable names</td>
</tr>
<tr>
<td>echo</td>
<td>Turning echo on displays the script commands as they are executed - good for “debugging”</td>
</tr>
<tr>
<td>input</td>
<td>Prompts user for input</td>
</tr>
<tr>
<td>pause</td>
<td>Pause until user presses any keyboard key</td>
</tr>
<tr>
<td>pause(n)</td>
<td>Pause for n seconds</td>
</tr>
<tr>
<td>waitForbuttonpress</td>
<td>Pause until user presses mouse button or keyboard key</td>
</tr>
</tbody>
</table>

CRT: Basic MATLAB
If you ever need to stop execution of a command or script file, press Ctrl-C i.e. the Control and C keys simultaneously e.g.

```
for p = 1:1000
    for q = 1:1000
        A(p,q) = p*q
    end
end
```

“for” loops are discussed later
A better approach might be

```matlab
A = zeros(1000,1000);
for p = 1:1000
    for q = 1:1000
        A(p,q) = p*q;
    end
end
```

greatly reduces memory footprint and speeds up operations.
Polynomials in MATLAB

In MATLAB, polynomials are represented by a row vector of the coefficients

e.g. a polynomial $f = 3x^3 - x^2 - 1$ is specified by the coefficient vector
$a = [3 \ -1 \ 0 \ -1]$
Polynomial Functions

See the VERY, VERY Brief Guide to MATLAB for the polynomial functions

polyval(a, x) : to evaluate a polynomial with coefficient matrix a at x
\[ f = 3x^3 - x^2 - 1 \]

\[ \gg a = [3 \ -1 \ 0 \ -1] \]

\[ \gg \text{polyval}(a, 1) \]

\[ \text{ans} = \]

\[ 1 \]
Polynomial Functions

roots(a) : to find the roots of a polynomial

poly(r) : to find the coefficient matrix from the roots
\[ f = 3x^3 - x^2 - 1 \]

\[ a = \begin{bmatrix} 3 & -1 & 0 & -1 \end{bmatrix} \]

\[ r = \text{roots}(a) \]

\[ r = \]

\[ 0.8241 \]

\[ -0.2454 + 0.5867i \]

\[ -0.2454 - 0.5867i \]
\[ f = 3x^3 - x^2 - 1 \]

\[
\begin{align*}
>> r &= \quad 0.8241 \\
&\quad -0.2454 + 0.5867i \\
&\quad -0.2454 - 0.5867i
\end{align*}
\]

\[
>> \text{poly}(r)
\]

\[
\text{ans} =
\begin{pmatrix}
1.0000 & -0.3333 & -0.0000 & -0.3333
\end{pmatrix}
\]
Polynomials and Regression

\[ p = \text{polyfit}(x, y, n) \]

- vector of x values
- vector of y values
- order of fitted polynomial
- coefficients of polynomial that fits data best on least square basis

CRT: Basic MATLAB
Flow Control

If you want to loop e.g. do something lots of times, with a different value of a variable each time or if you want your program to make decisions while it is running, you need “flow control”
MATLAB has five “constructs” for flow control

• if
• switch
• for
• while
• break
if
  if <logical condition>
    <statements for first case>
  elseif <logical condition>
    <statements for second case>
  else
    <otherwise>
end
if

if mark >= 69.5
  firstclass
elseif mark >= 40
  pass
else
  fail
end

If if finds a condition is satisfied, it executes the statement(s) that follow immediately, and then goes to end

CRT: Basic MATLAB
switch

switch <variable or expression>

case  <some value(s)>
    <statements for first case(s)>

case  <some value(s)>
    <statements for second case(s)>

case  <some value(s)>
    <statements for third case(s)>

otherwise
    <statements for other case(s)>

end
switch

switch lower(input('What day is it? ', 's'))
    case {'saturday', 'sunday'}
        disp('Weekend - hurrah!')
    case {'monday','friday'}
        disp('More weekend - cool')
    case {'tuesday', 'wednesday', …
        'thursday'}
        disp('Rest day - wicked')
    otherwise
        disp('Not a day')
end
switch works down the cases

When it finds a true condition, it executes the statement(s) that follow immediately, then goes to end
for

for n = 3:32
    r(n) = rank(magic(n));
end

Executes the statements the stated number of times

Note: you can have steps other than 1
e.g. n = 2:2:100 - even numbers up to 100
n = 10:-1:0 - countdown
while

Repeats statements until some logical condition is met

```
n = 1;
while n <= 500
    disp(n)
    n = n^2+ 1;
end
```

Note the use of indenting in loops – helps make the code much easier to read
break

Useful if you need to exit early from a loop

n = 1;
while n <= 5000
    disp(n)
    n = n^2 + 1;
    if n == 26 break
end % if
end
MATLAB Functions

Functions are m-files that can accept input “arguments” and return “output arguments”

The function m-file “blanks.m” is a simple example
function b = blanks(n)
%BLANKS String of blanks.
% BLANKS(n) is a string of n blanks.
% Use with DISP, eg. DISP(['xxx' BLANKS(20) 'yyy']).
% DISP(BLANKS(n)') moves the cursor down n lines.
%
% See also CLC, HOME, FORMAT.

% Copyright 1984-2002 The MathWorks, Inc.
% $Revision: 5.10$ $Date: 2002/04/15 03:53:35$

space = ' ';
b = space(ones(1,n));
function b = blanks(n)

The first line starts with the word function
It gives the function name, and the order of the “arguments”
Here there is only one input: n
This is the number of blanks required
There is one output b, a string of n blanks
function b = blanks(n)
    \% BLANKS String of blanks.
    \% BLANKS(n) is a string of n blanks.
    \% Use with DISP, eg. DISP(["xxx" BLANKS(20) 'yyy']).
    \% DISP(BLANKS(n)') moves the cursor down n lines.
    \%
    \% See also CLC, HOME, FORMAT.

The comment lines that follow are the help text you see when you type

>> help blanks

If you write your own, this will work for your functions too!

CRT: Basic MATLAB
The rest of the code is what the function does

```
space = ' ';  
b = space(ones(1,n));
```

Note that one line, often the last, gives a value for the output, here b
You can “call” the function from the command line or from another m-file

```matlab
>> myblanks = blanks(6)
myblanks =

>> xxblanks = ['x' blanks(6) 'x']
xxblanks =
x  x
```

Note the concatenation here
>> myblanks = blanks(6)

function b = blanks(n)
    varargin = ' ';
    b = varargin*ones(1,n);

% BLANKS String of blanks.
% BLANKS(n) is a string of n blanks.
% Use with DISP, eg. DISP(['xxx' BLANKS(20) 'yyy']).
% DISP(BLANKS(n))' moves the cursor down n lines.

% See also CLC, HOME, FORMAT.

% Copyright 1984-2002 The MathWorks, Inc.
% $Revision: 5.10 $  $Date: 2002/04/15 03:53:35 $
Note that everything inside a function is hidden from the outside

If we call `blanks` from the command line, the value of b and n are not defined (known) outside the function

```
>> blanks(6)
an =

>> b
??? Undefined function or variable 'b'.
```
This means we don’t have to worry about the function altering the values of variables we have defined

```matlab
>> b=6;
>> blanks(6)
ans =

>> b
b =
   6
```

not 6 blanks!
If we want to share a variable between the inside of a function and outside, we might declare the variable as “global”.

However, it is better practice to pass all variables in and out as arguments.
User-defined functions

MATLAB has lots of functions to play with, but you may want to write your own – as a function m-file.

For example, you may want a function which changes £ into $
function dollars = convert(pounds)
% CONVERT changes a given amount of
%pounds sterling into US dollars, using a global value
% for the exchange rate. It rounds down to a whole
% number of dollars.

global exchange_rate
dollars = floor(exchange_rate*pounds);

This is stored on the path as an m-file
called convert.m

It can then be called from the command
line or another m-file
For example

```matlab
>> global exchange_rate
>> exchange_rate = 1.5;
>> pounds = 200;
>> mydollars = convert(pounds)

mydollars =

300
```

An advantage of such files is that you can re-use them
Simple Numerical Analysis in MATLAB

“Function functions” are functions that have other functions as inputs.

Examples are finding minima, finding roots, quadrature, and solving ODEs numerically.
MATLAB’s favourite function is humps; a curve generated by the equation

\[ y = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04} - 6 \]
strong maxima near $x = 0.3$ and $x = 0.9$

note: humps does not intercept x-axis in this range
MATLAB’s favourite function is humps
Here is a modified version: newhumps

function y = newhumps(x)
%NEWHUMPS  A modified simple version of MATLAB’s humps.
%   Y = HUMPS(X) is a function with strong maxima near x = .3
%   and x = .9.
% Y = NEWHUMPS(X) subtracts 15 from HUMPS to ensure
% some roots in the range 0 <= x <= 1.

y = (1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6) -15;

If we try

>> x = 0:0.002:1;
>> y = newhumps(x);
>> plot(x,y)

we get ...

CRT: Basic MATLAB
now there are some roots!
fminbnd('newhumps', 0.5, 0.7) will find the minimum in the function newhumps between $x = 0.5$ and $x = 0.7$

fzero('newhumps', 0.5) will try to find a root near $x = 0.5$

feval('newhumps', 0.5) will compute the value of newhumps at $x = 0.5$
fminbnd

0.637
feval
quad(‘newhumps’, 0.2, 0.4) will numerically integrate newhumps between $x = 0.2$ and $x = 0.4$

quad uses a version of Simpson’s Rule
All these work as well on other functions

\[ fzero(@sin, 0.9\pi) \]

will try to find a root of \( \sin x \) near \( x = 0.9\pi \)

\@ is a function ‘‘handle’’
- can use instead of quotes

Returns

\[ \text{ans} = 3.14159265358979 \sim \pi \text{ as expect} \]
Key point

MATLAB is a powerful programming tool for Engineers, which is worth learning and using