1. **Introduction**
   1. **What is Octave?**

Octave is an open-source interactive software system for numerical computations and graphics. It is particularly designed for matrix computations: solving simultaneous equa- tions, computing eigenvectors and eigenvalues and so on. In many real-world engineering problems the data can be expressed as matrices and vectors, and boil down to these forms of solution. In addition, Octave can display data in a variety of different ways, and it also has its own programming language which allows the system to be extended. It can be thought of as a very powerful, programmable, graphical calculator. Octave makes it easy to solve a wide range of numerical problems, allowing you to spend more time experimenting and thinking about the wider problem.

Octave was originally developed as a companion software to a undergraduate course book on chemical reactor design4. It is currently being developed under the leadership of Dr. J.W. Eaton and released under the GNU General Public Licence. Octave’s usefulness is enhanced in that it is mostly syntax compatible with MATLAB which is commonly used in industry and academia.

* 1. **What Octave is not**

Octave is designed to solve mathematical problems *numerically*, that is by calculating values in the computer’s memory. This means that it can’t always give an exact solution to a problem, and it should not be confused with programs such as Mathematica or Maple, which give *symbolic* solutions by doing the algebraic manipulation. This does not make it better or worse—it is used for different tasks. Most real mathematical problems (particularly engineering ones!) do not have neat symbolic solutions.

* 1. **Who uses Octave?**

Octave and MATLAB are widely used by engineers and scientists, in both industry and academia for performing numerical computations, and for developing and testing math- ematical algorithms. For example, NASA use it to develop spacecraft docking systems; Jaguar Racing use it to display and analyse data transmitted from their Formula 1 cars; Sheffield University use it to develop software to recognise cancerous cells. It makes it very easy to write mathematical programs quickly, and display data in a wide range of different ways.

* 1. **Why not use a ‘normal’ highlevel language, e.g. C++**

C++ and other industry-standard programming languages are normally designed for writ- ing general-purpose software. However, solutions to mathematical problems take time to program using C++, and the language does not natively support many mathematical con- cepts, or displaying graphics. Octave is specially designed to solve these kind of problems, perform calculations, and display the results. Even people who will ultimately be writing

4[www.octave.org/history.html](http://www.octave.org/history.html)

software in languages like C++ sometimes begin by prototyping any mathematical parts using Octave, as that allows them to test the algorithms quickly.

Octave is available on the MDP Resource CD and can be downloaded from [www.octave.org](http://www.octave.org/) if required.

1. **Simple calculations**
   1. **Starting Octave**

If not already running start Octave, (see **start Programs Octave** on the MDP CD.) or type in a xterm window

*→ →*

octave

After a pause, a logo will briefly pop up in another window, and the terminal will display the header similar to this:

GNU Octave, version 2.1.57 (i386-pc-linux-gnu).

Copyright (C) 2004 John W. Eaton.

This is free software; see the source code for copying conditions. There is ABSOLUTELY NO WARRANTY; not even for MERCHANTIBILITY or FITNESS FOR A PARTICULAR PURPOSE. For details, type ‘warranty’.

Additional information about Octave is available at [http://www.octave.org.](http://www.octave.org/) Please contribute if you find this software useful.

For more information, visit <http://www.octave.org/help-wanted.html>

Report bugs to [<bug-octave@bevo.che.wisc.edu>](mailto:bug-octave@bevo.che.wisc.edu) (but first, please read <http://www.octave.org/bugs.html> to learn how to write a helpful report).

octave:1>

and you are now in the Octave environment. The octave:1> is the Octave prompt, asking you to type in a command.

If you want to leave Octave at any point, type quit at the prompt.

* 1. **Octave as a calculator**

The simplest way to use Octave is just to type mathematical commands at the prompt, like a normal calculator. All of the usual arithmetic expressions are recognised. For example, type

octave:##*>* 2+2

at the prompt and press return, and you should see

ans = 4

The basic arithmetic operators are + - \* /, and ^ is used to mean ‘to the power of’ (e.g.

2^3=8). Brackets ( ) can also be used. The order *precedence* is the same usual i.e. brackets

are evaluated first, then powers, then multiplication and division, and finally addition and subtraction. Try a few examples.

* 1. **Built-in functions**

As well as the basic operators, Octave provides all of the usual mathematical functions, and a selection of these can be seen in Table 1. These functions are invoked as in C++ with the name of the function and then the function *argument* (or arguments) in ordinary brackets (), for example5

octave:##*>* exp(1) ans = 2.7183

Here is a longer expression: to calculate 1*.*2 sin(40*◦* + ln(2*.*42)), type

octave:##*>* 1.2 \* sin(40\*pi/180 + log(2.4^2)) ans = 0.76618

There are several things to note here:

An explicit multiplication sign is always needed in equations, for example between the 1.2 and sin.

*•*

The trigonometric functions (for example sin) work in *radians*. The factor *π/*180 can be used to convert degrees to radians. pi is an example of a named *variable*, discussed in the next section.

*•*

* + - The function for a natural logarithm is called ‘log’, not ‘ln’.

Using these functions, and the usual mathematical constructions, Octave can do all of the things that your normal calculator can do.

1. **The Octave environment**

As we can see from the examples so far, Octave has an *command-line* interface— commands are typed in one at a time at the prompt, each followed by return. Octave is an *interpreted* language, which means that each command is converted to machine code after it has been typed. In compiled languages, e.g. C++, language the whole program is typed into a text editor, these are all converted into machine code in one go using a *compiler*, and then the whole program is run. These compiled programs run more quickly than an interpreted program, but take more time to put together. It is quicker to try things out with Octave, even if the calculation takes a little longer.6

5A function’s arguments are the values which are passed to the function which it uses to calculate its response. In this example the argument is the value ‘1’, so the exponent function calculates the exponential of 1 and returns the value (i.e. *e*1 = 2*.*7183).

6Octave can call external C++ functions however the functionality is less than MATLAB.

|  |  |
| --- | --- |
| cos | Cosine of an angle (in radians) |
| sin | Sine of an angle (in radians) |
| tan | Tangent of an angle (in radians) |
| exp | Exponential function (*ex*) |
| log | Natural logarithm (NB this is log*e*, not log10) |
| log10 | Logarithm to base 10 |
| sinh | Hyperbolic sine |
| cosh | Hyperbolic cosine |
| tanh | Hyperbolic tangent |
| acos | Inverse cosine |
| acosh | Inverse hyperbolic cosine |
| asin | Inverse sine |
| asinh | Inverse hyperbolic sine |
| atan | Inverse tangent |
| atan2 | Two-argument form of inverse tangent |
| atanh | Inverse hyperbolic tangent |
| abs | Absolute value |
| sign | Sign of the number (*−*1 or +1) |
| round | Round to the nearest integer |
| floor | Round down (towards minus infinity) |
| ceil | Round up (towards plus infinity) |
| fix | Round towards zero |
| rem | Remainder after integer division |

Table 1: Basic maths functions

* 1. **Named variables**

In any significant calculation you are going to want to store your answers, or reuse values, just like using the memory on a calculator. Octave allows you to define and use named variables. For example, consider the degrees example in the previous section. We can define a variable deg to hold the conversion factor, writing

octave:##*>* deg = pi/180 deg =0.017453

Note that the *type* of the variable does not need to be defined, unlike most high level languages e.g. in C++. All variables in Octave are floating point numbers.7 Using this variable, we can rewrite the earlier expression as

octave:##*>* 1.2 \* sin(40\*deg + log(2.4^2)) ans =0.76618

which is not only easier to type, but it is easier to read and you are less likely to make a silly mistake when typing the numbers in. Try to define and use variables for all your common numbers or results when you write programs.

You will have already have seen another example of a variable in Octave. Every time you type in an expression which is *not* assigned to a variable, such as in the most recent example, Octave assigns the answer to a variable called ans. This can then be used in exactly the same way:

octave:##*>* new = 3\*ans new =2.2985

Note also that this is not the answer that would be expected from simply performing

3 0*.*76618. Although Octave *displays* numbers to only a few decimal places (usually five)8, it stores them internally, and in variables, to a much higher precision, so the answer given is the more accurate one.9 In all numerical calculations, an appreciation of the rounding errors is very important, and it is essential that you do not introduce any more errors than there already are! This is another important reason for storing numbers in variables rather than typing them in each time.

*×*

When defining and using variables, the capitalisation of the name is important: a is a different variable from A. There are also some variable names which are already defined and used by Octave. The variable ans has also been mentioned, as has pi, and in addition i

and j are also defined as *√−*1 (see Section 14). Octave won’t stop you redefining these

7Or strings, but those are obvious from the context. However, even strings are stored as a vector of character ID numbers.

8MATLAB normally displays to 4 or 5 decimal places

9Octave stores all numbers in IEEE floating point format to double (64-bit) precision. The value of

ans here is actually 0.766177651029692 (to 15 decimal places).

as whatever you like, but you might confuse yourself, and Octave, if you do! Likewise, giving variables names like sin or cos is allowed, but also not to be recommended.

If you want to see the value of a variable at any point, just type its name and press return. If you want to see a list of all the named functions, variables10 you have created or used, type

octave:##*>* who

\*\*\* dynamically linked functions: dispatch

\*\*\* currently compiled functions: rem

\*\*\* local user variables: deg new

You will occasionally want to remove variables from the workspace, perhaps to save memory, or because you are getting confusing results using that variable and you want to start again. The clear command will delete all variables, or specifying

clear *name*

will just remove the variable called *name* .

* 1. **Numbers and formatting**

We have seen that Octave usually displays numbers to five significant figures. The format command allows you to select the way numbers are displayed. In particular, typing

octave:##*>* format long

will set Octave to display numbers to fifteen+ significant figures, which is about the accuracy of Octave’s calculations. If you type help format you can get a full list of the options for this command. With format long set, we can view the more accurate value of deg:

octave:##*>* deg

deg =.0174532925199433

>> format short

10Normally MATLAB only displays the user (Your) variables + ans

The second line here returns Octave to its normal display accuracy.

Octave displays very large or very small numbers using *exponential notation*, for example: 13142*.*6 = 1*.*31426 104, which is displayed by Octave as 1.3143e+04. You can also type numbers into Octave using this format.

*×*

There are also some other kinds of numbers recognised, and calculated, by Octave:

**Complex numbers** (e.g. 3+4i) Are fully understood by Octave, as discussed further in Section 14

**Inftnity** (Inf) The result of dividing a number by zero. This is a valid answer to a calculation, and may be assigned to a variable just like any other number

**Not a Number** (NaN) The result of zero divided by zero, and also of some other oper- ations which generate undefined results. Again, this may be treated just like any other number (although the results of any calculations using it are still always NaN).

* 1. **Number representation and accuracy**

Numbers in Octave, as in all computers, are stored in binary rather than decimal. In decimal (base 10), 12.25 means

12*.*25 = 1 *×* 101 + 2 *×* 100 + 2 *×* 10*−*1 + 5 *×* 10*−*2

but in binary (base 2) it would be written as

1101*.*01 = 1 *×* 23 + 1 *×* 22 + 0 *×* 21 + 1 *×* 20 + 0 *×* 2*−*1 + 1 *×* 2*−*2 = 12*.*25

Using binary is ideal for computers since it is just a series of ones and zeros, which can be represented by whether particular circuits are on or off. A problem with representing numbers in a computer is that each number can only have a finite number circuits, or *bits*, assigned to it, and so can only have a finite number of digits. Consider this example:

octave:##*>* 1 - 0.2 - 0.2 - 0.2 - 0.2 - 0.2 ans = 5.5511e-017

The result is very small, but not exactly zero, which is the correct answer. The reason is that 0.2 cannot be *exactly* represented in binary using a finite number of digits (it is 0*.*0011001100 *. . .* ). This is for exactly the same reasons that 1/3 cannot be exactly written as a base 10 number. Octave (and all other computer programs) represent these numbers with the closest one they can, but repeated uses of these approximations, as seen here, can cause problems. For this reason, you should think very carefully before checking that a number is *exactly equal* to another. It is usually best to check that it is the same to within a tolerance. We will return to this problem throughout this tutorial.

* 1. **Loading and saving data**

When you exit Octave, you lose all of the variables that you have created. If you need to quit Octave when you are in the middle of doing something, you can save your current session so that you can reload it later. If you type

octave:##*>* save anyname

it will save the entire workspace to a file called anyname.mat in your current directory (note that Octave automatically appends .mat to the end of the name you’ve given it).

You can then quit Octave, and when you restart it, you can type

octave:##*>* load anyname

which will restore your previous workspace, and you can carry on from where you left off.

You can also load and save specific variables. The format is

save *filename var1 var2* ...

For example, if you wanted to save the deg variable, to save calculating it from scratch (which admittedly is not very difficult in this case!), you could type

octave:##*>* save degconv deg

This will save the variable into a file called degconv.mat You can reload it by typing

octave:##*>* load degconv

Octave will also load data from text files, which is particularly useful if you want to plot or perform calculations on measurements from some other source. The text file should contain rows of space-separated numbers.

* 1. **Repeating previous commands**

Octave keeps a record of all the commands you have typed during a session, and you can use the cursor keys and to review the previous commands (with the most recent first). If you want to repeat one of these commands, just find it using the cursor keys, and press return.

*↑ ↓*

Once a command has been recalled, you can edit it before running it again. You can use and to move the cursor through the line, and type characters or hit Del to change the contents. This is particularly useful if you have just typed a long line and then Octave finds an error in it. Using the arrow keys you can recall the line, correct the error, and try it again.11

*← →*

* 1. **Getting help**

If you are not sure what a particular Octave command does, or want to find a particular function, Octave contains an integrated help system. The basic form of using help is to type

help *commandname*

11On some systems the emacs key bindings are acknowledged, e.g. *<*ctrl*>*p =*↑*, *<*ctrl*>*n=*↓*, *<*ctrl*>*f=*→*,

*< c*trl*>*b=*←*

For example:

octave:1> help sqrt

sqrt is a built-in function

- Mapping Function: sqrt (X)

Compute the square root of X. If X is negative, a complex result is returned. To compute the matrix square root, see

\*Note Linear Algebra::.

Overloaded function gsqrt(galois,...)

Additional help for built-in functions, operators, and variables

is available in the on-line version of the manual. Use the command ‘help -i <topic>’ to search the manual index.

Help and information about Octave is also available on the WWW

at [http://www.octave.org](http://www.octave.org/) and via the [help-octave@bevo.che.wisc.edu](mailto:help-octave@bevo.che.wisc.edu) mailing list.

12If you don’t know the name of the command you want, there are a number method to find if it exists. Typing help -i at the prompt will give a list of all the main areas of help.

More detailed information on a topic can be obtained by moving the cursor of the item of interest and pressing *<*return*>*. The list can be navigated using the key bindings **U**p, **N**ext, **P**revious. etc. Or directly from the prompt, e.g. to find out about arithmetic functions type;

octave:##*>* help -i arithmetic

Use the letter **‘q’ to quit from the help system** and return to the Octave command prompt.

Note that often the help information gives an indication of which area to search for fur- ther information, in the help sqrt example it is suggested to search the ‘Linear Algebra’ area, e.g.

octave:##*>* help -i linear algebra

Take some time browsing around the help system and anyone line help/manuals to get an idea of the commands that Octave provides.13

12MATLAB help messages tend to be shorter, but always indicate the command in UPPER CASE. Don’t copy these exactly—Octave and MATLAB commands are almost always in lower case, e.g. the square root function is sqrt(), *not* SQRT()

13MATLAB has some additional search facilities including the function lookfor which allows the user to search the help database

* 1. **Cancelling a command**

If you find yourself having typed a command which is taking a long time to execute (or perhaps it is one of your own programs which has a bug which causes it to repeat endlessly), it is very useful to know how to stop it. You can cancel the current command by typing

Ctrl-C

which should (perhaps after a small pause) return you to the command prompt.

* 1. **Semicolons and hiding answers**

Semicolons ‘;’ are often used in programming languages to separate functions, denote line- ends, e.g. C++ where ’;’ are added to almost every line. Semicolons are not required in Octave, but do serve a useful purpose. If you type the command as we have been doing so far, without a final semicolon, Octave always displays the result of the expression. However, if you finish the line with a semicolon, it stops Octave displaying the result. This is particularly useful if you don’t need to know the result there and then, or the result would otherwise be an enormous list of numbers:

1. **Arrays and vectors**

Many mathematical problems work with sequences of numbers. In many languages they are called *arrays*; in Octave they are just examples of *vectors*. Vectors are commonly used to represent the three dimensions of a position or a velocity, but a vector is really just a list of numbers, and this is how Octave treats them. In fact, vectors are a simple case of a *matrix* (which is just a two-dimensional grid of numbers). A vector is a matrix with only one row, or only one column. We will see later that it is often important to

. Σ

distinguish between *row* vectors ( *· · ·* ) and *column* vectors *··* , but for the moment that

won’t concern us.

*·*

* 1. **Building vectors**

There are lots of ways of defining vectors and matrices. Usually the easiest thing to do is to type the vector inside *square* brackets [], for example

octave:##*>* a=[1 4 5] a =

1 4 5

octave:##*>* b=[2,1,0] b =

2 1 0

octave:##*>* c=[4;7;10] c =

4

7

10

A list of numbers separated by spaces or commas, inside square brackets, defines a row vector. Numbers separated by semicolons, or carriage returns, define a column vector.

You can also construct a vector from an existing vector by including it in the definition, for example

octave:##*>* a=[1 4 5] a =

1 4 5

|  |  |  |
| --- | --- | --- |
| octave:##*>* | d=[a 6] |  |
| d =  1 4 | 5 | 6 |

* 1. **The colon notation**

A useful shortcut for constructing a vector of counting numbers is using the colon symbol ‘:’, as in this example

octave:##*>* e=2:6 e =

2 3 4 5 6

The colon tells Octave to create a vector of numbers starting from the first number, and counting up to (and including) the second number. A third number may also be added between the two, making *a* : *b* : *c*. The middle number then specifies the increment between elements of the vector.

octave:##*>* e=2:0.3:4 e =

2.0000 2.3000 2.6000 2.9000 3.2000 3.5000 3.8000

Note that if the increment step is such that it can’t exactly reach the end number, as in this case, it generates all of the numbers which do not exceed it. The increment can also be negative, in which case it will count down to the end number.

* 1. **Displaying large vectors and matrices**

If you ask Octave to display a matrix or vector that will not fit onto a single screen that it will present the values one page at a time. Try

octave:##*>* v = 1:1000

zeros(*M, N* )

ones(*M, N* )

Create a matrix where every element is zero. For a row vector of size *n*, set *M* = 1*, N* = *n* Create a matrix where every element is one. For a row vector of size *n*, set *M* = 1*, N* = *n*

linspace(*x*1*, x*2*, N* ) Create a vector of N elements, evenly spaced

between *x*1 and *x*2

logspace(*x*1*, x*2*, N* ) Create a vector of N elements, logarithmically

spaced between 10*x*1 and 10*x*2

Table 2: Vector creation functions

Press the space bar to see the next page of values and use the ‘q’ key to quit the display and return to the Octave command prompt. You can also use the ‘b’ key to scroll backwards up throught the values being displayed.

It is sometimes convenient to turn off this pagination facility, for example when dis- playing intermediate values during a long calculation. This can be achieved using the

octave:##*>* more off

command. As you would expect, pagination can be turned back on again using

octave:##*>* more on

* 1. **Vector creation functions**

Octave also provides a set of functions for creating vectors. These are outlined in Table

2. The first two in this table, zeros and ones also work for matrices, and the two function arguments, *M* and *N* , specify the number of *rows* and *columns* in the matrix respectively. A row vector is a matrix which has one row and as many columns as the size of the vector. Matrices are covered in more detail in Section 9.

* 1. **Extracting elements from a vector**

Individual elements are referred to by using normal brackets (), and they are numbered starting at *one*, not zero as in C++. If we define a vector

octave:##*>* a=[1:2:6 -1 0] a =

1 3 5 -1 0

then we can get the third element by typing

octave:##*>* a(3) ans =

5

The colon notation can also be used to specify a range of numbers to get several elements at one time

octave:##*>* a(3:5) ans =

5 -1 0

octave:##*>* a(1:2:5) ans =

1 5 0

* 1. **Vector maths**

Storing a list of numbers in one vector allows Octave to use some of its more powerful features to perform calculations. In C++ if you wanted to do the same operation on a list of numbers, say you wanted to multiply each by 2, you would have to use a for loop to step through each element. This can also be done in Octave (see Section 7), but it is much better to make use of Octave’s vector operators.

Multiplying all the numbers in a vector by the same number, is as simple as multiplying the whole vector by number. This example uses the vector a defined earlier:

octave:##*>* a \* 2 ans =

2 6 10 -2 0

The same is also true for division. You can also add the same number to each element by using the + or - operators, although this is not a standard mathematical convention.

Multiplying two vectors together in Octave follows the rules of matrix multiplication (see Section 9), which doesn’t do an element-by-element multiplication.14 If you want to do this, Octave defines the operators .\* and ./, for example

*a*1 *b*1 *a*1*b*1

*a*2 .\* *b*2 = *a*2*b*2

*a*3

*b*3

*a*3*b*3

*a*3

*b*3

*a*3*b*3

Note the ‘.’ in front of each symbol, which means it’s a element-by-element operation. For example, we can multiply each of the elements of the vector a, defined earlier, by a different number:

octave:##*>* b=[1 2 3 4 5]; octave:##*>* a.\*b

ans =

14Recall that the only vector products mathematically defined are the dot and cross product, both of which represent particular operations on two vectors, neither of which just multiply the elements together and return another vector.

1 6 15 -4 0

The element-by-element ‘to the power of’ operator .^ is particularly useful. It can be used to raise a vector of numbers to a power, or to raise a number to different powers, depending on how it is used:

octave:##*>* b .^ 2 ans =

1 4 9 16 25

octave:##*>* 2 .^ b ans =

2 4 8 16 32

The first example squares each element of b; the second raises 2 to each of the powers given in b.

All of the element-by-element vector commands (+ - ./ .\* .^) can be used between two vectors, as long as they are the *same size and shape*. Otherwise corresponding elements cannot be found and an error is given.

Most of Octave’s functions also know about vectors. For example, to create a list of the value of sine at 60-degree intervals, you just need to pass a vector of angles to the sin function:

octave:##*>* angles=[0:pi/3:2\*pi] angles =

0 1.0472 2.0944 3.1416 4.1888 5.2360 6.2832

octave:##*>* y=sin(angles) y =

0 0.8660 0.8660 0.0000 -0.8660 -0.8660 -0.0000

1. **Plotting graphs**

Octave has powerful facilities for plotting graphs via a second open-source program GNUPLOT15, however some of the range of plotting options are restricted compared with MATLAB The basic command is plot(x,y), where x and y are the co-ordinates. If given just one pair of numbers it plots a point, but usually you pass *vectors*, and it plots all the points given by the two vectors, joining them up with straight lines.16 The sine curve defined in the previous section can be plotted by typing

octave:##*>* plot(angles,y)

15[www.gnuplot.org](http://www.gnuplot.org/)

16The two vectors must, naturally, both be the same length.

1

0.8

0.6

0.4

0.2

0

−0.2

−0.4

−0.6

−0.8

−1

0 1 2 3 4 5 6 7

Figure 1: Graph of *y* = sin(*x*), sampled every 60*◦*.

A new window should open up, displaying the graph, shown in Figure 1. Note that it automatically selects a sensible scale, and plots the axes.

At the moment it does not look particularly like a sine wave, because we have only taken values one every 60 degrees. To plot a more accurate graph, we need to calculate y at a higher resolution:

octave:##*>* angles=linspace(0,2\*pi,100); octave:##*>* y=sin(angles);

octave:##*>* plot(angles, y);

The linspace command creates a vector with 100 values evenly spaced between 0 and 2*π* (the value 100 is picked by trial and error). Try using these commands to re-plot the graph at this higher resolution. Remember that you can use the arrow keys and to go back and reuse your previous commands.

*↑ ↓*

* 1. **Improving the presentation**

You can select the colour and the line style for the graph by using a third argument in the plot command. For example, to plot the graph instead with red circles, type

octave:##*>* plot(angles, y, ’ro’)

The last argument is a string which describes the desired styles. Table 3 shows the possible values (also available by typing help plot in Octave).

To put a title onto the graph, and label the axes, use the commands title, xlabel

and ylabel:

octave:##*>* title(’Graph of y=sin(x)’) octave:##*>* xlabel(’Angle’)

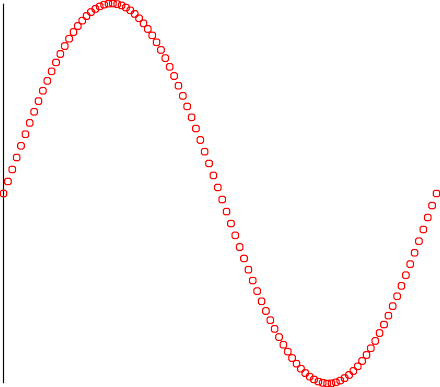
octave:##*>* >> ylabel(’Value’)

Strings in Octave (such as the names for the axes) are delimited using apostrophes (’).

|  |  |  |
| --- | --- | --- |
| w whitew  m magenta  c cyan  r red  g green  b blue  y yellow*†*  k black*†* | . point  o circle  x x-mark  + plus  \* star  s square*†*  d diamond*†*  v triangle (down)*†*  ^ triangle (up)*†*  < triangle (left)*†*  > triangle (right)*†*  p pentagram*†*  h hexagram*†* | - solid  : dotted*†*  -. dashdot*†*  -- dashed*†* |

Table 3: Colours and styles for symbols and lines in the plot command (see help plot).. (*†*N.B. Only available in MATLAB)

Graph of y=cos(x)

1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

0.8

0.6

0.4

0.2

0

Value

−0.2

−0.4

−0.6

−0.8

−1

0 1 2 3 4 5 6 7

Angle

Figure 2: Graph of *y* = sin(*x*), marking each sample point with a red circle.

In some circumstances the command replot has to called to enable the graph to update.

A grid may also be added to the graph, by typing

octave:##*>* grid on

Figure 2 shows the result. You can resize the figure or make it a different height and width by dragging the corners of the figure window.

* 1. **Multiple graphs**

Several graphs can be drawn on the same figure by adding more arguments to the plot command, giving the x and y vectors for each graph. For example, to plot a cosine curve as well as the previous sine curve, you can type

octave:##*>* plot(angles,y,’:’,angles,cos(angles),’-’)

1

Sine

Cosine

0.8

0.6

0.4

0.2

0

−0.2

−0.4

−0.6

−0.8

−1

0 1 2 3 4 5 6 7

Figure 3: Graphs of *y* = sin(*x*) and *y* = cos(*x*).

where the extra three arguments define the cosine curve and its line style. You can add a legend to the plot using the legend command:

octave:##*>* legend(’Sine’, ’Cosine’)

where you specify the names for the curves in the order they were plotted. If the legend doesn’t appear in a sensible place on the graph, you can pick it up with the mouse and move it elsewhere. You should be able to get a pair of graphs looking like Figure 3.

Thus far, every time you have issued a plot command, the existing contents of the figure have been removed. If you want to keep the current graph, and overlay new plots on top of it, you can use the hold command. Using this, the two sine and cosine graphs could have been drawn using two separate plot commands:

octave:##*>* plot(angles,y,’:’) octave:##*>* hold on

octave:##*>* plot(angles,cos(angles),’g-’) octave:##*>* legend(’Sine’, ’Cosine’)

If you want to release the lock on the current graphs, the command is (unsurprisingly)

hold off.

* 1. **Multiple figures**

You can also have multiple figure windows. If you type