WIKIBOOKS
Open books for an open world

Octave Programming Tutorial/Getting started

The aim of this tutorial is to give you a quick introduction to basic Octave and to show that you know a lot of it already. If you should ever get stuck or need more information on an Octave function or command, type

```
help command
```

at the Octave prompt. `command` is the name of the Octave command or function on which to find help. Be warned though that the descriptions can sometimes be a bit technical and filled with jargon.

Starting Octave

Type `octave` in a terminal window to get started. You should see something like the following, depending on the version that you have installed:

```
GNU Octave, version 3.6.4.0
Copyright (C) 2013 John W. Eaton and others.
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.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

octave:1>
```

To exit Octave, type quit or exit.

Entering commands

The last line above is known as the Octave prompt and, much like the prompt in Linux, this is where you type Octave commands. To do simple arithmetic, use `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), and `^` (exponentiation).

Many mathematical functions are available and have obvious names (with many of those similar to other programming languages). For example, we have:

- trigonometric functions[1]: `sin`, `cos`, `tan`
- inverse trigonometric functions: `asin`, `acos`, `atan`

- natural and base 10 logarithms: `log`, `log10`
- exponentiation: `exp`
- absolute value: `abs`

Various constants are also pre-defined: `pi`, `e` (Euler's number), `i` and `j` (the imaginary number sqrt(-1)), `inf` (Infinity), `NaN` (Not a Number - resulting from undefined operations, such as `Inf/Inf`.)

Here are some examples showing the input typed at the prompt and the output returned by Octave.

$2 + 3$

```
octave:1> 2 +
3
ans = 5
```

$\frac{\log_{10} 100}{\log_{10} 10}$

```
octave:2>
log10(100)/log10(10)
ans = 2
```

$\left\lfloor \frac{1 + \tan 1.2}{1.2} \right\rfloor$

```
octave:3> floor((1+tan(1.2)) /
1.2)
ans = 2
```

$\sqrt{3^2 + 4^2}$

```
octave:4> sqrt(3^2 +
4^2)
ans = 5
```

$e^{i \cdot \pi}$

```
octave:5> e^(i*pi)
ans = -1.0000e+00 + 1.2246e-16i
octave:6> # Comment: From Euler's famous formula
octave:6> # extremely close to the correct value of -1
```

Some things to note:

- Octave requires parentheses around the input of a function (so, `log(10)` is fine, but (`log 10`) is not).
- Any spacing before and after arithmetic operators is optional, but allowed.
- Not all Octave functions have obvious names (e.g. `sqrt` above). Don't panic for now. You will get to know them as we go along.

Plotting

You are going to plot the following pictures using Octave:

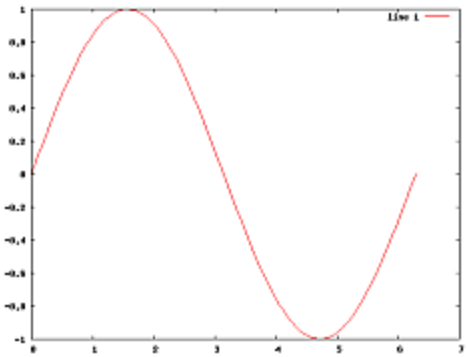


Figure 1

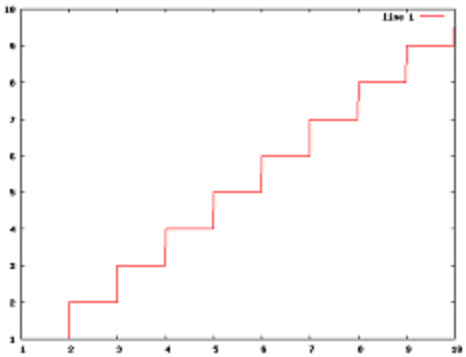


Figure 2

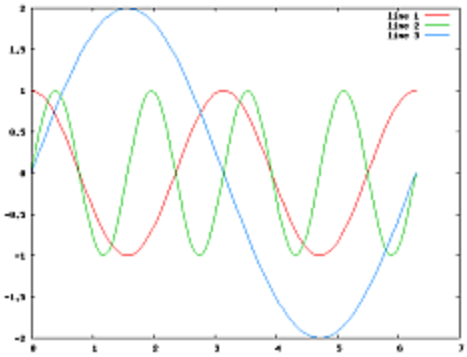


Figure 3

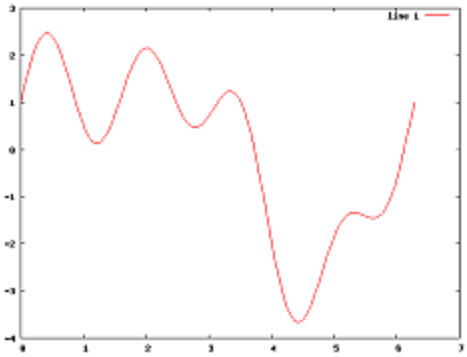


Figure 4

Figure 1 contains a plot of sin *x* vs *x* and is generated with the following commands. While this is a simple example, it is meant to illustrate the basic functionality. We will see more elaborate examples later.

```
x = linspace(0, 2*pi, 100);
y = sin(x);
plot(x, y);
figure;
```

The command that actually generates the plot is, of course, `plot(x, y)`. Before executing this command, we need to set up the variables, *x* and *y*. The `plot` function simply takes two vectors of equal length as input, interprets the values in the first as *x*-coordinates and the second as *y*-coordinates and draws a line connecting these coordinates.

The first command above, `x = linspace(0, 2*pi, 100)`, uses the `linspace` function to make a vector of equally spaced intervals (sometimes also called "linearly spaced values"). The first value in the vector is 0, the final value is 2π and the vector contains 100 values. This vector is assigned to the variable named `x`.

The second command computes the sin of *each* value in the vector variable, `x`, and stores the resulting vector in the variable `y`.

(As an aside: the *name of a variable* can be any sequence of letters, digits and underscores that does not start with a digit. There is no maximum length for variable names, and the case of alphabetical characters is important, i.e. `a` and `A` are two different variable names.)

Exercise

Plot the function $y = \lfloor x \rfloor$ for $x \in [0, 15]$. (This is Figure 2).
Note : The graph may appear slightly inaccurate if the length(3rd) parameter of linspace is not sufficiently large. See the next heading for more information.

More on commands

The following commands and functions are useful for setting up variables for plotting 2D graphs.

- `linspace` creates a vector of evenly (linearly) spaced values.

Usage: `linspace(start, stop, length)`. The `length` parameter is optional and specifies the number of values in the returned vector. If you leave out this parameter, the vector will contain 100 elements with `start` as the first value and stop as the last.

- `plot` plots a 2-dimensional graph.

Usage: `plot(x, y)` where `x` and `y` are vectors of equal length.

- `figure` creates a new plotting window.

This is useful for when you want to plot multiple graphs in separate windows rather than replacing your previous graph or plotting on the same axes.

- `hold on` and `hold off` sets whether you want successive plots to be drawn together on the same axes or to replace the previous plot.
- The `;` at the end of a line suppress the display of the result. Try to remove it, and see !

Example

We are going to plot Figures 3 and 4. Figure 3 contains the 3 trigonometric functions

- $\cos 2x$,
- $\sin 4x$, and
- $2 \sin x$

on one set of axes. Figure 4 contains the sum of these 3 functions.

Firstly, we use `linspace` to set up a vector of *x*-values.

```
octave:1> x = linspace(0, 2*pi);
```

Then, we compute the *y*-values of the 3 functions.

```
octave:2> a = cos(2*x);
octave:3> b = sin(4*x);
octave:4> c = 2*sin(x);
```

The following plots the first graph.

```
octave:5> figure;
octave:6> plot(x, a);
octave:7> hold on;
octave:8> plot(x, b);
octave:9> plot(x, c);
```

We use line 5 (`figure`) to tell Octave that we want to plot on a new set of axes. It is good practice to use `figure` before plotting any new graph. This prevents you from accidentally replacing a previous plot with the new one.

Note that on line 7, `hold on` is used to tell Octave that we *don't* want to replace the first plot (from line 6) with subsequent ones. Octave will plot everything after `hold on` on the same axes, until the `hold off` command is issued.

The figure you see shows all three plotted functions in the same color. To let Octave assign different colors automatically plot all functions in one step.

```
octave:10> plot(x, a, x, b, x, c);
```

Finally, we plot the second graph.

```
octave:11> figure;
octave:12> hold off;
octave:13> plot(x, a+b+c);
```

Line 11 creates a new graph window and line 12 tells Octave that any subsequent plots should simply replace previous ones. Line 13 generates the plot of the sum of the 3 trigonometric functions. A small note for mac users, if plot figure commands do not work first use this command: > setenv ("GNUTERM", "x11")

Exercises

- Plot the absolute value function for $x \in [-5, 5]$.
- Plot a circle of radius 1, centered at the origin. (This is not so easy.)
- Plot your favourite function(s) like sin and cos

Warning

If you try (or have tried) to plot something like x^2 or $\sin x \times \cos x$, you will run into trouble. The following error messages are common. In the case of `x^2`:

```
error: for A^b, A must be square
```

In the case of `sin(x)*cos(x)`:

```
error: operator *: nonconformant arguments (op1 is 1x100, op2 is 1x100)
```

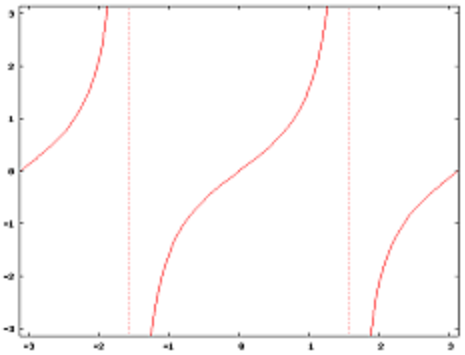
This error occurs whenever you try multiply or divide two vector variables (remember that `x` and `y` are vectors). For now, you can do one of two things.

1. use plot(sin(x).*cos(x)) - notice the '.' between sin(x) and *. For x^2 plot(x.*x)
2. Read the section on vectors and matrices.

Challenge

Since Octave is a numerical (and not symbolic) mathematics package, it does make numerical errors and does not handle some operations well. To confirm this, make a plot of tan *x*, for *x* between -π and π. What is wrong with the resulting picture?

Your task is to generate the (much better looking) graph below using what you have learned so far and the `axis` function. `axis` can be used to adjust which part of the plot is actually displayed on screen. Use the command `help axis` to determine how this function works.



It might take some thinking to get the asymptote lines at $x = \pm\pi/2$ right. You can use `help plot` to find out how to plot dotted lines. (Try to make the plot yourself before looking at the solution below.)

Following commands could be used to generate the plot shown above.

```
octave:1> x_part_left  = linspace(-pi,  -pi/2-0.001, 100);
octave:2> x_part_mid   = linspace(-pi/2, pi/2, 100);
octave:3> x_part_right = linspace( pi/2+0.001, pi,   100);
octave:4> plot(x_part_left, tan(x_part_left));
octave:5> hold on;
octave:6> plot(x_part_mid, tan(x_part_mid));
octave:7> plot(x_part_right, tan(x_part_right));
octave:8> y_limit = 4;
octave:9> axis([-pi, pi, -y_limit, y_limit]);
octave:10> plot(linspace(-pi/2, -pi/2, 100), linspace(-y_limit, y_limit, 100), '.');
octave:11> plot(linspace( pi/2,  pi/2, 100), linspace(-y_limit, y_limit, 100), '.');
octave:12> hold off;
```

The horizontal plot range -π to π is split into three vectors such that singular points are skipped for the plot. In lines 4-7 the separate parts of the tan function are plotted. Thereafter, in line 8, we choose a limit for the y-axis and use it to constrain the vertical plot range (using the `axis` command [line 9]). Finally we add the asymptote lines at $x = \pm\pi/2$ in a dotted style (lines 10 & 11).

Script files

It is useful to be able to save Octave commands and return them later on. You might want to save your work or create code that can be reused (by yourself or somebody else). Such files are known as Octave script files. They should be saved with a `.m` extension so that Octave can recognise them. (The `.m` extension is used because MATLAB calls its script files M-files and Octave is based on MATLAB.)

To run an existing script in Octave, you have to be in the same directory as the script file and type in the name of the file **without the** `.m` in Octave. For example, if I have a script called `myscript.m` in an `octave` directory, the following two commands will execute the script.

```
chdir('~/octave'); % This changes to the octave directory
myscript;
```

Note that the `chdir('~/octave')` command is necessary only if you are not already inside that directory when running Octave.

In the following section you will be shown a number of new statements that you can use to make your Octave code much more powerful. A number of example script files are provided and you should save these into a directory for later use. A good idea is to create a directory called `octave` in your home directory and store all your Octave files in there.

Return to the Octave Programming tutorial index

References

1. As usual in Mathematics, the trigonometric functions take their arguments in radians.