Octave: Introduction

See "Octave Starting Guide" for how to launch Octave. In the sections below, >> indicates the Octave command prompt (an entry point for typing Octave commands in the command prompt window).

1. Simple Arithmetic

The basic arithmetic operators are `+, -, *, / and ^`(powers). Enter the following commands at the Octave prompt to learn the basic way that formulae can be evaluated in Octave.

```
>> 1+2
```

Press enter after you have typed in the command. If you want to save the answer from a calculation you can give it a name e.g. x,

```
>> x=(1+2)^2/3
```

This line creates a variable to save the answer in. If you want to find out what x is, type

```
>> x
```

Equally you can use it in a formula

```
>> 2*x/3+2
```

2. Variable Names

Octave distinguishes between lowercase and uppercase letters, so **a** and **A** are different variables in Octave. Only the first 19 characters in a variable name are important.

3. Vectors and Matrices

Octave is most used to work with vectors and matrices. Vectors are either row vectors or column vectors.

Row vectors have one row and several columns like $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

Column vectors have several rows in one column like $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

To enter a row vector into Octave, use a command like

```
>> v=[1 2 3 4]
```

To check this see what Octave has stored as v

```
>> v
```

To create column vector in Octave you must mark the end of a row with a semi-colon

```
>> a=[1; 2; 3; 4]
```

Another way is to use the dash (') for transposing

```
>> a=[1 2 3 4]'
```

or

```
>> v=[1 2 3 4]
>> a=v'
```

4. Creating Matrices

To define a matrix we use the symbol ; to indicate the end of row.

```
>> a=[1 2 ; 3 4; 5 6]
```

What output does this produce? how many rows and columns does it define?

5. Elements of Vectors and Matrices

Define a vector f

```
>> f=[ 2 3 4 ]
```

Try the following commands

```
>> f(1)
>> f(2)
>> f(3)
```

What did you obtain? You can define the same vector as

```
>> f(1)=2
>> f(2)=3
>> f(3)=4
```

Define a matrix p,

```
>> p=[1 2; 3 4; 5 6]
```

Try the following commands

```
>> p(1,2)
>> p(2,1)
>> p(2,:)
>> p(:,2)
```

Now use the information that you can get from these examples to write down the commands that will select

- the number in the 3rd row, and 2nd column of p
- the first row from p
- the third column from p

6. Evenly Spaced Integer Arrays

We can define an evenly spaced integer array as v=start : step : end
This array begins at start value, increase by step value and finish at or before end value. For example,

```
>> v=2:2:9
produces v=[2 4 6 8]
```

A special case only uses start and end and here step=1

```
>> i=1:10
produces i=[1 2 3 4 5 6 7 8 9 10]
>> v=2:5
produces v=[2 3 4 5]
```

An array with a non-integer step

```
>> v=0:0.1:1
produces v=[0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]
```

7. Vector Sizes

Check the length and size of a vector

```
>> v=[1:5]
>> length(v)
>> size(v)
```

Try the "whos" command

```
>> whos
```

It will list all the variables in the current workspace, together with information about their size, bytes, class, etc. You can clear all variables and functions in the workspace by using "clear", and type "whos" again.

```
>> clear
```

8. Basic Matrix Operations

The symbols for matrix addition, subtraction, multiplication and powers are `+, -, *, and ^` If `a` is a square matrix then `a^2` means a*a - the matrix product of a with itself. Write Octave commands that create the following matrices

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$b = \begin{bmatrix} 3 & 4 \\ 0 & 1 \end{bmatrix}$$

$$c = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$$

Now try to find

```
>> b+c
>> b-2*c
>> a+b
```

Octave also does matrix multiplication (provided that the matrices can be multiplied consistently).

```
>> b*a
```

Does the product a*b make sense?

```
>> a*b
```

More complicated algebraic expressions like `b+c-(b^2)*c` can also be found.

9. The Dot Symbol: Element-by-element Operations

The dot symbol is special to Octave - it is not a standard mathematical notation/operation and should not be confused with the dot product of two vectors. It changes the meaning of `^`, * or / in a very important way. If a is a square matrix then `a^2` means a*a - the matrix product of a with itself. But `a.^2` means that we get a new matrix by squaring each element in it separately. Even if a is not square (so that a*a is not defined) we can still define `a.^2`. We have defined a, b and c in the previous section. Predict the values of `a.^2` and check that you are right using Octave

```
>> a.^2
```

Also see what `b.^2` is

```
>> b.^2
```

Check that this is different from matrix `b^2`

```
>> b^2
```

Check how `b.*c` is different from `b*c`

```
>> b.*c
>> b*c
```

10. Vector inputs for Functions

Functions in Octave (e.g. sin, cos, log, exp, sqrt and your own functions) accept vector inputs, and give vector outputs.

```
>> x=[1.0 3.0 7.5]
>> y=sin(x)
```

This gives a vector [0.8415 0.1411 0.9380].

11. Adding another entry to vectors

Another nice thing about Octave vector variables is that they are flexible. If you like to add another entry to a vector, its very easy to do so.

```
>> x=[1:5]
>> x=[x 6]
```

another example

```
>> a=[1 2 3]
>> b=[4 5 6]
>> c=[a b]
>> d=[a; b]
```

12. Useful Built-in Octave Functions for arrays

Use "help" for the details of the functions (e.g. >> help length) .

- length - length of vector
- sum - adds up entries in an array
- max - maximum entry in an array
- min - minimum entry in an array
- sort - sorts entries in an array into increasing order
- rand - producing random numbers

13. Using the find command and logical expressions

The "find" command lets you find the places in an array where the entries obey a specified condition.

```
>> a=rand(1,7)
>> index=find(a>0.5)
>> b=a(index)
```

where “rand” produces random numbers distributed between 0 and 1. The “find” command involves a logical condition - this typically involved comparison operators.

<	mean	less than
<=		less than or equal
>		greater than
>=		greater than or equal
==		equal
∼=		not equal
∼		NOT
&		AND
|		OR

14. Zeros array and Ones array

Special arrays are useful when you write a program. To create a row vector containing 4 zeros, check how the following command works

```
>> r=zeros(1,4)
```

What does the following command do

```
>> r=zeros(4,1)
```

To create a row vector containing 4 ones, check how following command works

```
>> r=ones(1,4)
```

15. Significant Figures

Octave works to 15 significant figures but usually show only 5 figures.

```
>> pi
3.1416
```

pi (the ratio of a circle's circumference to its diameter) is a built-in constant. You can change to the long format in which all 15 figures are displayed.

```
>> format long
>> pi
3.14159265358979
```

To go back short format

```
>> format short
>> pi
3.1416
```

Very large or small numbers use exponential format for outputs and inputs.

```
>> 4*10^33
4.0000e+23

>> 4e23
4.0000e+23

>> 1.67e-24          this is equivalent to 1.67*10^(-24)
1.6700e-24
```

16. The Consequences of Finite Accuracy on a Computer

Octave can only store it to 15 significant figures. Sometimes an answer that really ought to be exactly zero might appear as a small number like 10^{-15}. Type

```
>> sin(pi)
```

How much is this supposed to be?

17. Plotting Graphs

One very useful feature of Octave is its ability to plot graphs of functions quickly. For example suppose that we want to plot the graph of $y = x^2$ between 0 and 5.

Step1: Create a vector of evenly spaced points between 0 and 5

```
>> x=0:0.05:5;
```

The semicolon (;) is used to stop Octave printing all 100 numbers out.

Step2: Compute the y values for the curve $y = x^2$

```
>> y=x.^2;
```

The dot here is very important - it squares the element of x one at a time.

Step3: Plot just the graph of $y = x^2$

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

Step4: Add labels to the x and y axes on the graph as shown below.

```
>> xlabel('x values')
>> ylabel('y values')
```

18. Printout from Octave

Command to save the current image as a color eps (encapsulated postscript) file with the name "filename.eps".

```
>> print -depsc filename.eps
```

Type the command help for details.

```
>> help print
```

19. Putting Several Graphs in One Plot

Suppose that we want to compare the graphs of $y = x$, $y = x^2$, $y = x^3$ over the interval from -2 to +2. First we set up the x vector.

```
>> x= -2:0.05:2;
```

Next we need to calculate the coordinates on these three graphs

```
>> y1=x; y2=x.^2; y3=x.^3;
```

An extended version of plot can take more pairs of x,y arrays. Each x, y pair is added to the graph.

```
>> plot(x,y1,x,y2,x,y3); grid; shg
```

Here we put three commands on the one line - grid puts lines on the graph to make it easier to interpret. The shg command means "show graph window" (it brings the current figure window forward). Octave uses a default color scheme (i.e. using colors in the order :blue, green, red, cyan,...). You can control the color scheme yourself, e.g. the following commands makes the graphs blue, black and green.

```
>> plot(x,y1,'b',x,y2,'k',x,y3,'g')
```

You can change the colors or the style of the three plots yourself by adding a third 'style parameter' to any (or all) of the x,y pairs. This is typical of Octave - to make plots you have to give the x and y data and then there are optional things that you do the plot. To see available options, type the help command

```
>> help plot
```

Examples

```
>> plot(x,y,'bo')              plots graph as blue circles
>> plot(x,y1,'k',x,y2,'b--')   1st curve: black line and 2nd: blue dashed
```

the latter can be written in a different way

```
>> plot(x,y1,'k')
>> hold on
>> plot(x,y2,'b--')
```

"hold on" holds the current plot and all axis properties so that subsequent graphing commands add to the existing graph. "hold off" return to the default mode whereby plot commands erase the previous plots and reset all axis properties before drawing new plot. "clf" also clear the current figure.

```
>> clf
```

20. Subplot

Use the subplot command to generate multiple plots in one figure. subplot(m,n,p) breaks the figure window into an m-by-n matrix of small axes, selecting the p-th axes for the current plot.

```
>> subplot(2,1,1)
>> plot(x,y1)
>> subplot(2,1,2)
>> plot(x,y2)
```

21. Loading data from external files

```
load fileName
```

Example: first create a matrix "results" and save it into a data file "datafile" in the ascii format (text file). then, load the data from the text file and plot the data.

```
>> results=rand(10,2);
>> save datafile results -ascii
```

```
>> load datafile;
>> x=datafile(:,1);
>> y=datafile(:,2);
>> plot(x,y)
```

22. Octave Programming

Octave has an interactive interface as discussed in the previous sections, the feature is helpful for quick calculations or developing codes efficiently. It is also possible to execute commands written in files. This is useful for automating a series of steps you need to perform many times, or extending the Octave language for your application.

23. Script m-files

Scripts are collection of Octave commands stored in plan text files. When you type the name of the script file at the Octave prompt, the script file are executed as if you had typed them in from the keyboard. Script files must end with the extension ".m", and often these files are referred to as **m-files**.

Create a file "myScript.m". Octave provides a built in editor, though you can use any text editor you like.

```
>> edit myScript.m
```

and then type the following lines into the editor window.

```
x=0:.01:10;
y=exp(-x/2).*cos(5*x);
plot(x,y,'o-')
```

after saving the script, type myScript (without .m extension) at the Octave prompt.

```
>> myScript
```

24. Function m-files

Another kind of m-files are called Function m-files. Octave Functions accept input arguments and return output arguments, and help us to achieve complicated tasks. Some examples of functions are shown below.

Example 1: Create M-file "test.m" using a text editor.

```
>> edit test.m
```

and then type the following two lines in the editor window.

```
function c = test(a,b)
c=a*b;
```

The first line of a function m-file contains the syntax definition: function name (test), input (a,b) and output (c) arguments. The name of a function, as defined in the first line, should be the same as the name of the file without the .m extension.

After saving the m-file, use the "test" function

```
>> test(2,3)
```

Example 2: Create m-file "fact.m"

```
function f = fact(n)
% FACT Factorial
% FACT(N) returns the factorial of N
% usually denoted by N!
f=prod(1:n); % function body
```

Comment lines begin with a percent sign. Comment lines can appear anywhere in an m-file. The comment lines immediately following the function definition line constitute the help entry for the file.

```
>> help fact
```

Use the fact function

```
>> fact(10)
```

Example 3: If the function has multiple output values, enclose the output argument list in square bracket. Input argument, if present, are enclosed in parentheses. Use commas to separate multiple input or output argument. Here is an example.

```
function [x,y] = foo(a,b)
% x,y: output,  a,b: input
x=a+b;
y=a-b;
```

Use the "foo" function

```
>> [x,y]=foo(1,2)
```

25. Scripts and Functions

Files that contain Octave language code are called m-files, and there are two kinds of m-files. The differences are summarized below.

Script M-Files	Function M-Files
• Do not accept input arguments or return output arguments	• Can accept input arguments and return output arguments
• Operate on data in the workspace	• Internal variables are local to the function by default
• Useful for automating a series of steps you need to perform many times	• Useful for extending the Octave language for your application

26. Commenting

Commenting involves placing Human Readable Descriptions inside of computer programs detailing what the Code is doing. Proper use of commenting can make code maintenance much easier, as well as helping make finding bugs faster. Further, commenting is very important when writing functions that other people will use. The comment character in Octave is %. Everything from the % to the end of that line is ignored by the program and is only for use by the human reader of the code. For example,

```
% Add up all the vector elements.
y = sum(x);   % Use the sum function.
```

27. Flow Control

We need a way to make the computer do repetitive processes, and give it some logic so it can decide when to stop iterating. In Octave the repetitive processes are done using "for"and "while" commands to set up loops. There are several ways to handle logic, the most popular is the "if" construction. Some of useful flow control statements in Octave are

- for
- while
- if
- break
- return

27.1. for-loop

The for-loop repeats a group of statements a fixed, predetermined number of times. A matching end delineates the statements. For example,

```
for i=2:6
    x(i)=2*x(i-1);
end
```

Example: A loop with an index incremented by two

```
for i=1:2:n
    ...
end
```

Example: A loop with an index that counts down

```
for i=n:-1:1
    ...
end
```

Example: A loop with non-integer increments

```
for x=0:pi/5:pi
    ...
end
```

Example: You can nest multiple for-loops. It is a good idea to indent the loops for readability.

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

27.2. while-loop

The while-loop executes a statement or group of statements repeatedly as long as the controlling expression is true. The relational and logical operators (e.g. $>, <, ==$, &) are listed in section 13.

Example

```
n=1;
while prod(1:n) < 1000
      n=n+1;
end
n       % this displays the final result
```

27.3. if-statement

The if-statement evaluates a logical expression and executes a group of statements based on the value of the expression. For example,

```
if a < 0
   disp('a is negative')
end
```

Example

```
if rem(a,2)==2
    disp('a is even')
else
    disp('a is odd')
end
```

Example

```
if (attendance >= 0.8) & (grade_average >= 60)
   disp('pass')      % this is not about this couse.
end
```

Example

```
if n < 0             % If n negative, display error message.
    disp('Input must be positive');
elseif rem(n,2) == 0 % If n positive and even, divide by 2.
    a = n/2;
else                 % If n positive and odd, increment and divide.
    a = (n+1)/2;
end
```

27.4. break and return

The "break" and "return" statements provide an alternative way to exit from a loop construct.

The **break** statement terminates the execution of a for-loop or while-loop. When a break statement is encountered, execution continues with the next statement outside of the loop. In nested loops, break exits from the innermost loop only.

The **return** statement is used to force an exit from a function. This can have the effect of escaping from a loop. Any statement following the loop that are in the function body are skipped.

28. Text Outputs to the Command Window

Output to the command window is achieved with either the "disp" function or "fprintf" function.

- disp - Simple to use. Provides limited control over appearance of output.

```
>> disp(5)
   5

>> x=1:3; disp(x)
   1  2  3

>> disp('You will Never Walk Alone!')
   You will Never Walk Alone!

>> s='Planet Nine'; disp(s)
   Planet Nine
```

- fprintf - Slightly more complicated than disp. Provides total control over appearance of output.

```
>> x=3;
>> fprintf('Square root of %g is %8.6f\n',x,sqrt(x));
   The Square root of 3 is 1.732051

>> x=1:4; y=sqrt(x);
>> fprintf('%9.4f\n',y)
   1.0000
   1.4142
   1.7321
   2.0000

>> x=1:4; y=sqrt(x);
>> fprintf('y = %9.4f\n',y)
   y =    1.0000
   y =    1.4142
   y =    1.7321
   y =    2.0000
```

The conversion specifiers control the output of array elements.

Code	Conversion instruction
`%s`	format as a string
`%d`	format with no fractional part (integer format)
`%f`	format as a floating-point value
`%e`	format as a floating-point value in exponential notation
`%g`	format in the most compact form of either `%f` or `%e`
`\n`	insert newline in output string
`\t`	insert tab in output string

In addition to specifying the type of conversion (e.g. `%d`,`%f`,`%e`), one can also specify the width and precision of the result of the conversion.

Syntax:

```
%wd       example  %3d
%w.pf     example  %14.5f
%w.pe     example  %12.3f
```

`%14.5f` means
use floating point format to convert a numerical value to a string 14 characters wide with 5 digits after the decimal point.

`%12.3e` means
use exponential notation format to convert numerical value to a string 12 characters wide with 3 digits after the decimal point. The 12 characters for the string include the `e+00` or `e-00` (or the `e+000` or `e-000` on Windows)

29. Text Outputs to Files

Output to a file requires the fprintf function.
Example: Writing contents of a vector to a file.

```
x= ...                                          % content of x
fout = fopen('myfile.dat','wt');                % open myfile.dat
for k=1:length(x)
   fprintf(fout,'%4d     %5.2f\n',k,x(k));  % format:%4d and %5.2f
```

```
end
fclose(fout);                                   % close myfile.dat
```

Files are opened with the fopen command and closed with the fclose command. Before reading or writing a text or binary file you must open it with the fopen command.

```
fid = fopen('filename','permission')
```

The permission string specifies the kind of access you require. Possible permission strings include:

- r for reading only
- w for writing only
- a for appending only
- r+ for both reading and writing
- t for text mode
- b for binary mode

When you finish reading or writing, use fclose to close the file.

```
fclose(fid);
```

This tutorial is written based on the Matlab manual (MathWorks) and documents by Dr. Anthony O'Connor at Griffith University and Dr. Tom Huber at Gustavus Adolphus College.

Summary

1. Simple Arithmetic with Octave
2. Variable Names
3. Vectors and Matrices
4. Creating Matrices
5. Elements of Vectors and Matrices
6. Evenly spaced real/integer arrays
7. Vector Sizes
8. Basic Matrix Operations
9. The dot symbol: element-by-element operations
10. Vector inputs for Functions
11. Adding another entry to vectors
12. Useful built-in Octave functions for arrays
13. Using the find command and logical expressions
14. Zeros array and Ones array
15. Significant figures
16. The consequences of finite accuracy on a computer
17. Plotting Graphs
18. Printout from Octave
19. Putting several graphs of the one plot
20. Subplot
21. Loading data from external files
22. Octave Programming
23. Script m-files
24. Function m-files
25. Scripts and Functions
26. Commenting %
27. Flow control: the for and while-loops and the if-statement
28. Text outputs to the command window
29. Text outputs to files