

```
% GNU Octave is a (programmable) calculator and is very good at performing
% matrix operations. The basic syntax is the same as MATLAB's. At Octave's
% command prompt, a command can be entered. If you end a line with a semicolon,
% the output is suppressed. If the output is longer than one screen, you might
% have to press 'q' to get back to the prompt. Everything you enter at the
% prompt can as well be written into a script file with extension .m (like this
% one). Scripts can be executed by calling its name. Comments are done with the
% '%' sign.
```

```
%%%%%%%% GETTING HELP
```

```
% The command 'help <command>' displays the help text for the desired
% command.
```

```
help rand
```

```
% Search for the given string in the help text of all functions.
```

```
lookfor eigenvalues
```

```
% List all currently defined variables
```

```
who
```

```
% Delete all variables defined until now
```

```
clear
```

```
% Clear screen
```

```
clc
```

```
%Turn off output pagination
```

```
more off
```

```
%%%%%%%% DATA ENTRY
```

```
% Vectors and matrices are entered using square brackets [ ].
```

```
% Elements are separated by a space or a comma, a new row is
```

```
% started with a semicolon:
```

```
% A 1x4 row vector
```

```
a = [ 1, 2, 3, 4 ]
```

```
a2 = [ 1 2 3 4 ]
```

```
% A 2x2 matrix
```

```
A = [ 1, 2; 3, 4 ]
```

```
A2 = [ 1 2; 3 4 ]
```

```
% Get the size of a matrix
```

```
size(A)
```

```
size(A,1)
```

```
size(A,2)
```

```
%%%%%%%% DATA GENERATION
```

```
% Generate a row vector with elements 1, ..., 10
```

```
b = [1:10]
```

```
% Generate a row vector with elements 1, 1.1, 1.2, ..., 10
```

```
c = [1:0.1:10]
```

```
% Get the length of a vector
```

```
length(c)
```

```
% Create a 2x3 matrix filled with zeros or ones respectively
```

```
C = zeros(2,3)
```

```
D = ones(2,3)
```

```
% Create a 2x2 identity matrix
```

```
E = eye(2)
```

```
%Create matrix from other matrices/vectors (dimensions must agree)
```

```
X = [c;c]
```

```
Y = [A2 A2]
```

```
help repmat
```

```
Z = repmat(A,2,3)
```

```
% Create a column vector of 10 uniformly distributed random numbers  
% between 5 and 15.
```

```
u = unifrnd(5, 15, 10, 1)
```

```
% Create a 5x5 matrix with normally distributed random variables with a  
% mean of 2.5 and a sigma of 5.0.
```

```
N = normrnd(2.5, 5.0, 5, 5)
```

```
%%%%%%%% DATA ACCESS
```

```
% All indices in Octave start with 1, as opposed to 0 as usual in other  
% programming languages.
```

```
% Retrieve the element in row 1 and column 2
```

```
A(1,2)
```

```
% Retrieve all elements of row 1 in the matrix
```

```
A(1,:)
```

```
% Retrieve all elements of column 2 in the matrix
```

```
A(:,2)
```

```
% Retrieve a submatrix
```

```
Z2 = Z(1:2,3:6)
```

```

% Retrieve every third element of a vector
x = [1:20]
x2 = x(1:3:length(x))

% Saving and loading data
save A
clear A
load A

%%%%%% MATRIX OPERATIONS

% Transpose
A'

% Matrix addition, subtraction, multiplication and inversion
F = A + E + C * D'
G = F * inv(F)

% Element-wise operations
H = A * 2 + A .* E + A .^ 2

% Matrix-scalar addition/multiplication
threes = 3 + zeros(3)
tens = 10*ones(3)

%%%%%% OTHER FUNCTIONS
% Can be used on scalars as well as matrices. When applied to matrices the
% operations are performed elementwise.
a = 2
b = 3
v = [2 4 6]
w = [3 5 7]
sin(a)
sin(v)
cos(a)
cos(v)
atan2(a, b)
atan2(v, w)
sqrt(a)
sqrt(v)

%%%%%% PROGRAMMING CONSTRUCTS

% Functions
% Functions have the following layout:
% function [retval1, retval2, ...] <function_name>(arg1, arg2, ...)

```

```

%     <function body>
% end
% Returning values is performed by assigning values to the return values
% defined in the header of the function.
function y = add_two_numbers(a, b)
    y = a + b;
end

% For loops
for i=[1:10]
    if mod(i,2) == 0
        disp(['even: ', num2str(i)])
    else
        disp(['odd: ', num2str(i)])
    endif
endfor

% Always try to vectorize operations when possible!
v1 = [1:10]
v2 = [3:12]
dotProduct = 0

for i=1:length(v1)
    dotProduct = dotProduct + v1(i)*v2(i)
endfor
% Better:
dotProduct = sum(v1.*v2)

%%%%%% BASIC PLOTTING

% Create a vector of values in the range [1, 10] with an increment of 0.1
% and suppress the output (semicolon at the end).
x = -2*pi:0.1:2*pi;
% Compute sin() for all elements of the vector
y = sin(x);

% Close all existing plot windows
close all
% Plot the the values of x against those in y
plot(x, y)
% Draw following plots into the same figure. If this is not set subsequent
% plots erase the previously generated plots.
hold on
% Plot the cosine of the data points in green (g) with markers (+)
% instead of lines.
plot(x, cos(x), '+g');

```

```
% Plot a blue point
plot(2, 0.5, 'ob');

title("sine and cosine")
xlabel('x (rad)')
ylabel('y = f(x)')

% Add a grid
grid on
%Useful options: 'markersize', 'linewidth'
%See also the commands: xlabel, ylabel, title

% Save the complete plot to a file.
print('/tmp/plot.png', '-dpng')
```