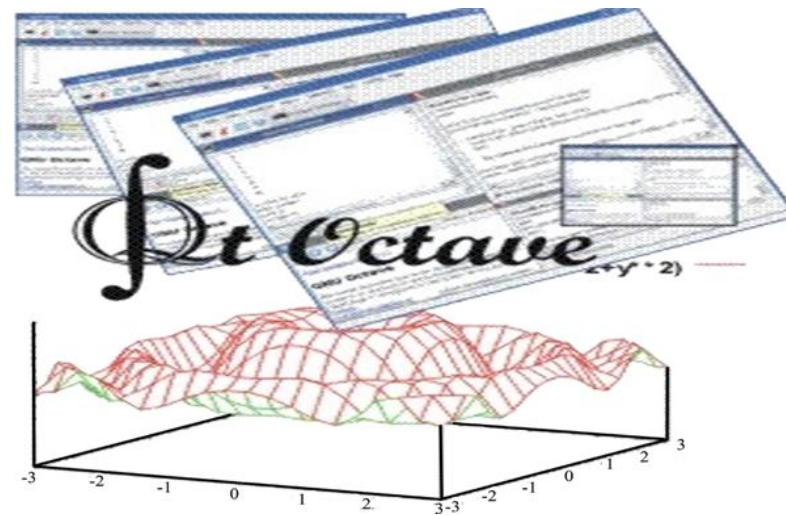




Octave

An alternative to MATLAB



Mr. Raveendra Pai G

Lecturer, Model Engineering College,

Thrikkakara

email-raviegg@gmail.com



Some FAQs

- Creator- John W. Eaton
- Octave became GNU Octave in 1997 (beginning with version 2.0.6)
- Gnu octave - Free software (GPL)
- Documentation- <http://www.octave.org/docs.html>
- Qt octave is a GUI for Gnu octave
- Convenient command line interface for solving linear and non linear problems numerically
- Prototyping, numerical experiments etc

Analysis Tool

- Solving numerical linear algebra problems
- Finding the roots of nonlinear equations
- Integrating ordinary functions
- Manipulating polynomials
- Integrating ordinary differential and differential-algebraic equations

System Simulation Tool

- Control Theory
- Signal Processing
- Image Processing
- Audio Processing

Computational Engine

- Programmable
- Easily Extendible
- Graphical Comparison Tool

Data types

- Matrix objects
- Character strings
- Scalars
- Ranges
- Structures

Matrices

- Octave supports matrix operations like MATLAB
- Matrix objects can be of any size, and can be dynamically reshaped and resized
- Some commonly used matrix manipulations are

`var=[a11 a12 a13 ; a21 a22 a23; a31 a32 a33];`

rows, `-var(i,:)` –selects *i*th row of the matrix

columns, `-var(:,j)` –selects *j*th column of the matrix

submatrices, `-var([i1:i2],[j1:j2]);`

`var(i,j)` – selects the element of the matrix

Range objects

- A range expression is defined by the value of the first element in the range, an optional value for the increment between elements, and a maximum value which the elements of the range will not exceed.
- The base, increment, and limit are separated by colons and may contain any arithmetic expressions and function calls.
- `a=[initial value : increment: final value];`

String objects

- A character string in Octave consists of a sequence of characters enclosed in either double-quote or single-quote marks.
- Internally, Octave stores strings as matrices of characters.
- All the indexing operations that work for matrix objects also work for strings.

Structure

- Octave's data structure type can help you to organize related objects of different types.
- The current implementation uses an associative array with indices limited to strings

```
x.a=1
```

```
x.b=[1,2;3,4]
```

```
x.c="string"
```

creates a structure with three elements.

Object sizes

- A group of functions allow you to display the size of a variable or expression.
- These functions are defined for all objects.
- They return -1 when the operation doesn't make sense.
- For example, the data structure type doesn't have rows or columns, so the rows and columns functions return -1 for structure arguments.

Object size functions

- `columns(A)`
 - Return the number of columns of A.
- `rows(A)`
 - Return the number of rows of A.
- `length(A)`
 - Return the number of rows of A or the number of columns of A, whichever is larger.

More object size functions

- `d=size(A)`
 - Return the number rows and columns of A, the result is returned in the 2 element row vector d.
- `[nr,nc]=size(A)`
 - The number of rows is assigned to nr and the number of columns is assigned to nc.
- `d=size(A,n)`
 - A second argument of either `n=1` or `n=2`, size will return only the row or column dimension.

Detecting object properties

- `ismatrix(A)`
 - Return 1 if A is a matrix. Otherwise, return 0.
- `isvector(A)`
 - Return 1 if A is a vector. Otherwise, return 0.
- `isscalar(A)`
 - Return 1 if A is a scalar. Otherwise, return 0

Detecting matrix properties

- `issquare(A)`
 - If A is a square matrix, then return the dimension of A . Otherwise, return 0.
- `issymmetric(A,tol)`
 - If A is symmetric within the tolerance specified, then return the dimension of A . Otherwise, return 0. If tol is omitted, $tol=eps$

Range definition

- The range 1:5
 - defines the set of values [1,2,3,4,5].
- The range 1:2:5
 - defines the set of values [1,3,5].
- The range 1:3:5
 - defines the set of values [1,4].
- The range 5:-3:1
 - defines the set of values [5,2].

More about ranges

- Note that the upper (or lower, if the increment is negative) bound on the range is not always included in the set of values.
- Ranges defined by floating point values can produce surprising results because floating point arithmetic is used.
- If it is important to include the endpoints of a range and the number of elements is known, use the `linspace()` function

Special matrix object `eye()`

- `eye(x)`
 - If invoked with a single scalar argument, `eye` returns a square identity matrix with the dimension specified.
- `eye(n,m)` or `eye(size(A))`
 - If you supply two scalar arguments, `eye` takes them to be the number of rows and columns.
- `eye`
 - Calling `eye` with no arguments is equivalent to calling it with an argument of 1.

Special matrix object ones()

- `ones(x)`
 - If invoked with a single scalar argument, `ones` returns a square matrix of 1's with the dimension specified.
- `ones(n,m)` or `ones(size(A))`
 - If you supply two scalar arguments, `ones` takes them to be the number of rows and columns.
- `ones`
 - Calling `ones` with no arguments is equivalent to calling it with an argument of 1.

Special matrix object zeros()

- `zeros(x)`
 - If invoked with a single scalar argument, `zeros` returns a square matrix of 0's with the dimension specified.
- `zeros(n,m)` or `zeros(size(A))`
 - If you supply two scalar arguments, `zeros` takes them to be the number of rows and columns.
- `zeros`
 - Calling `zeros` with no arguments is equivalent to calling it with an argument of 1.

Special matrix object rand()

- rand(x)

- If invoked with a single scalar argument, rand returns a square matrix of random numbers between 0 and 1 with the dimension specified.

- rand(n,m) or rand(size(A))

- If you supply two scalar arguments, rand takes them to be the number of rows and columns.

- rand

- Calling rand with no arguments is equivalent to calling it with an argument of 1.

Special matrix object randn()

- randn(x)
 - With a single scalar argument, randn returns a square matrix of Gaussian random numbers between 0 and 1 with the dimension specified.
- randn(n,m) or randn(size(A))
 - For two scalar arguments, randn takes them to be the number of rows and columns.
- randn
 - Calling rand with no arguments is equivalent to calling it with an argument of 1.

STRINGS

- A string constant consists of a sequence of characters enclosed in either double-quote or single-quote marks:
- Strings in Octave can be of any length.
- Since the single-quote mark is also used for the transpose operator it is best to use double-quote marks to denote strings.

Literals

- Some characters cannot be included literally in a string constant. You represent them instead with escape sequences, which are character sequences beginning with a backslash (\).
- Another use of backslash is to represent unprintable characters such as newline `\n` or tab `\t` and others

String functions

- `blanks(n)` Return a string of `n` blanks.
- `setstr(A)` Convert a matrix to a string. Each numeric element is converted to an ascii character.
- `strcat(s1,...,sn)` Return a string containing all the arguments concatenated.
- `str2mat(s1,..., sn)` Return a valid string matrix containing the strings `s1, ..., sn` as its rows.
- `deblank(s)` Removes the trailing blanks from the strings

String comparison

- `index(s1,s2)` Return the position of the first occurrence of the string `s2` in `s1`, or 0 if not found.
- Note: `index` does not work for arrays of strings.
- `rindex(s1,s2)` Return the position of the last occurrence of the string `s2` in `s1`, or 0 if not found.

Note: `rindex` does not work for arrays of strings.

- `strcmp(s1,s2)` Compares two strings, return 1 if they are the same, otherwise 0.
- `isstr(s)` Return 1 if `s` is a string, otherwise, 0.

Substring functions

- `findstr(s1,s2)` Return the vector of all positions in the longer string where an occurrence of the shorter substring starts.
- `strrep(s1,s2,s3)` In string `s1`, replace all occurrences of the substring `s2` with substring `s3`.
- `substr(s,n1,n2)` Return the substring of `s` starting at character `n1` and is `n2` characters long.

String conversions

- `bin2dec(s)` Return a decimal number corresponding to the binary number represented as a string of 0s and 1s.
- `dec2bin(n)` Return a binary number as a string of 0s and 1s corresponding to the non-negative decimal number `n`.
- `hex2dec(s)` Return a decimal number corresponding to the hexadecimal number stored in the string `s`.
- `dec2hex(n)` Return the hex number corresponding to the non-negative decimal number `n`, as a string.
- `str2num(s)` Convert the string `s` to a number.
- `num2str(n)` Convert the number `n` to a string.

More string conversions

- `toascii(s)` Return ascii representation of `s` in a matrix.
- `tolower(s)` Return a copy of the string `s`, with each upper-case character replaced by the corresponding lower-case one; non-alphabetic characters are left unchanged.
- `toupper(s)` Return a copy of the string `s`, with each lower-case character replaced by the corresponding upper-case one; non-alphabetic characters are left unchanged.

Testing characters

isalnum(s) isalpha(s) isascii(s) iscntrl(s)

isdigit(s) isgraph(s) islower(s) isprint(s)

ispunct(s) isspace(s) isupper(s) isxdigit(s)

- The above functions return 1 (true) or 0 (false) if the tested character is in the set represented by the function.

Variables

- Variables let you give names to values and refer to them later.
- The name of an Octave variable must be a sequence of letters, digits and underscores, but it may not begin with a digit.
- There is no limit on the number of characters in a variable name.
- Case is significant in variable names. The symbols a and A are distinct variables.

Built-in variables

- A number of variables have special built-in meanings. For example, PWD holds the current working directory, and pi names the ratio of the circumference of a circle to its diameter.
- Octave has a long list of all the predefined variables. Some of these built-in symbols are constants and may not be changed.

Status of variables

- clear options pattern -Delete the names matching the given patterns from the symbol table.
- who options pattern, whos options pattern-List currently defined symbols matching the given patterns.

Options- They may be shortened to one character but may not be combined.

- | | |
|--------------|--|
| -a(ll) | List all currently defined symbols. |
| -b(uiltins) | List built-in variables and functions. |
| -f(unctions) | List user-defined functions. |
| -l(ong) | Print a long listing of symbols |
| -v(ariables) | List user-defined variables. |

Expressions

Expressions are the basic building block of statements in Octave.

- An expression evaluates to a value, which you can print, test, store in a variable, pass to a function, or assign a new value to a variable with an assignment operator.
- An expression alone can serve as a statement. Most statements contain one or more expressions which specify data to be operated on.
- Expressions include variables, array references, constants, and function calls, as well as combinations of these with various operators.

Index expressions

- An index expression allows you to reference or extract selected elements of a matrix or vector.
- Indices may be scalars, vectors, ranges, or the special operator (:), which may be used to select entire rows or columns.
 - $A(i,:)$
 - $A(:,j)$
 - $A(i1:i2,j1:j2)$

Addition operators

- $x+y$, Addition- If both operands are matrices, the number of rows and columns must both agree.
- $x.+y$, Element by element addition- This is equivalent to the $+$ operator.
- $x-y$, Subtraction- If both operands are matrices, the number of rows and columns of both must agree.
- $x.-y$, Element by element subtraction- This is equivalent to the $-$ operator.
- If both are matrices in any case the $r=c$

Multiplication operators

- $x*y$, Matrix multiplication- The number of columns of x must agree with the number of rows of y .
- $x.*y$, Element by element multiplication. If both operands are matrices, the number of rows and columns must both agree.

Division operators

- x/y , Right division- Equivalent to $(\text{inv}(y') * x)'$
- $x./y$, Element by element right division- Each element of x is divided by each corresponding element of y .
- $x \setminus y$, Left division- Equivalent to the $\text{inv}(x) * y$
- $x. \setminus y$, Element by element left division- Each element of y is divided by each corresponding element of x .

Power operators

- x^y or $x^{**}y$, Power operator- x and y both scalar: returns x raised to the power y . x scalar, y is a square matrix : returns result using eigen value expansion. x is a square matrix and y scalar: returns result by repeated multiplication if y is an integer, else by eigenvalue expansion. x and y both matrices: returns an error.
- $x.^y$ or $x.^{*}y$, Element by element power operator- If both operands are matrices, the number of rows and columns must both agree.

Unary operators

- $+x$ or $+x.$, A unary plus operator has no effect on the operand.
- $-x$ or $-x.$, Negation or element by element negation.
- x' , Complex conjugate transpose. For real arguments, this is the same as the transpose operator. For complex arguments, equivalent to $\text{conj}(x.)'$

Comparison operators

Comparison operators compare numeric values for relationships.

- All of comparison operators return a value of 1 if the comparison is true, or 0 if it is false.
- For matrix values, the comparison is on an element-by-element basis.

```
[1,2;3,4]==[1,3;2,4];ans=[1,0;0,1]
```

- For mixed scalar and matrix operands, the scalar is compared to each element in turn.

```
[1,2;3,4]==2;ans=[0,1;0,0]
```

Relational operators

- $x < y$ True if x is less than y .
- $x \leq y$ True if x is less than or equal to y .
- $x == y$ True if x is equal to y .
- $x \geq y$ True if x is greater than or equal to y .
- $x > y$ True if x is greater than y .
- $x \neq y$ True if x is not equal to y .
- $x \sim = y$ True if x is not equal to y .
- $x \langle \rangle y$ True if x is not equal to y .

Boolean expressions

- A boolean expression is a combination of comparisons using the boolean operators "or" (\vee), "and" ($\&$), and "not" (\neg).
- Boolean expressions can be used wherever comparison expressions can be used.
- If a matrix value used as the condition it is only true if all of its elements are nonzero.
- Each element of an element-by-element boolean expression has a numeric value (1 true, 0 false).

Boolean operators

- $b1 \ \& \ b2$
 - Elements of the result are true if both corresponding elements of $b1$ and $b2$ are true.
- $b1 \ | \ b2$
 - Elements of the result are true if either of the corresponding elements of $b1$ or $b2$ is true.
- $!b$
- $\sim b$
 - Each element of the result is true if the corresponding element of b is false.

Assignment expressions

- An assignment is an expression that stores a new value into a variable.

```
z=1
```

- Assignments can store string values also.

```
thing="food"
```

```
kind="good"
```

```
message=["this ",thing," is ",kind]
```

- It is important to note that variables do not have permanent types. The type of a variable is whatever it happens to hold .

Assigning indexed expressions

- Assignment of a scalar to an indexed matrix sets all of the elements that are referenced by the indices to the scalar value.

$A(:,2)=5$

- Assigning an empty matrix `[]` allows you to delete rows or columns of matrices and vectors.

$A(3,:)=[]$

$A(:,1:2:5)=[]$

Assigning multiple variables

- An assignment is an expression, so it has a value. Thus, $z=1$ as an expression has the value 1. One consequence of this is that you can write multiple assignments together:

$x=y=z=0$

- This is also true of assignments to lists, so the following are valid expressions

$[a,b,c]=[u,s,v]=\text{svd}(A)$

$[a,b,c,d]=[u,s,v]=\text{svd}(A)$

$[a,b]=[u,s,v]=\text{svd}(A)$

Increment operators

- Increment operators increase or decrease the value of a variable by 1.
 - The operators to increment and decrement a variable are written as `++` and `--`.
 - It may be used to increment a variable either before (`++x`) or after (`x++`) taking its value.
 - For matrix and vector arguments, the increment and decrement operators work on each element of the operand.

Control Statements

- Control statements control the flow of execution in programs.
 - All the control statements start with special keywords
 - Each control statement has a corresponding end keyword
 - The list of statements contained between the start keyword the corresponding end keyword is called the body of a control statement.

Control structures

Octave if statement

– The else and elseif clauses are optional. Any number of elseif clauses may exist.

if (condition)

then-body;

elseif (condition)

elseif-body;

else

else-body;

endif

More control structures

- Octave switch statement— Any number of case labels are possible

switch expression

case label

command_list;

case label

command_list;

...

otherwise

command_list;

endswitch

More control structures

- Octave while statement

```
while (condition)
```

```
    body;
```

```
endwhile
```

- Octave for statement

```
for var = expression
```

```
    body;
```

```
endfor
```

More control statements

- The break statement
 - jumps out of the innermost for or while loop that encloses it. The break statement may only be used within the body of a loop.
- The continue statement
 - like break, is used only inside for or while loops. It skips over the rest of the loop body, causing the next cycle around the loop to begin immediately.

Functions

- A function is a name for a particular calculation. For example, the function `sqrt` computes the square root of a number.
- A fixed set of functions are built-in, which means they are available in every program. The `sqrt` function is a built-in function.
- In addition, you can define your own functions.

Calling functions

- A function call expression is a function name and list of arguments in parentheses.
 - The arguments are expressions which give the data for function to operate on.
 - When there is more than one argument, they are separated by commas.
 - If there are no arguments, you can omit the parentheses.

Arguments for functions

- Most functions expects a particular number of arguments.

`sqrt(x2+y2)` # One argument

`ones(n,m)` # Two arguments

`rand()` # No arguments

`rand(“seed”,1)` # Two arguments

- Some functions like `rand` take a variable number of arguments and behave differently depending on the number of arguments.

Return values for functions

- Most functions return one value

`y=sqrt(x)`

- Functions in Octave (in common with perl) may return multiple values.

`[u,s,v]=svd(A)`

- computes the singular value decomposition of the matrix A and assigns the three result matrices to u,s, and v.

Functions and script files

- Complicated programs can often be simplified by defining functions.
- Functions can be defined directly on the command line during interactive sessions.
- Alternatively, functions can be created as external files, and can be called just like built-in functions.

Defining functions

- In its simplest form, the definition of a function named name looks like this:

function name

body;

endfunction

- A valid function name any valid variable name.
- The function body consists of expressions and control statements.

Passing information to functions

- Normally, you will want to pass some information to the functions you define.

```
function name(arg-list)
```

```
  body;
```

```
endfunction
```

- where arg-list is a comma-separated list of arguments.
When the function is called, the argument names hold the values given in the call.

Returning information

- In most cases, you will also want to get some information back from the functions you define.

```
function ret-var=name(arg-list)
```

```
body;
```

```
endfunction
```

- The symbol `ret-var` is the name of the variable, defined within the function, that will hold the value to be returned.

Returning more information

- Functions may return more than one value.

```
function [ret-list]=name(arg-list)
```

```
body;
```

```
endfunction
```

- where ret-list is a comma-separated list of variable names that will hold the values returned from the function. Note that ret-list is a vector enclosed in square brackets.

Script files

- A script file is a file containing (almost) any sequence of commands.
- It is read and evaluated just as if you had typed each command at the prompt.
- It provides a way to store a sequence of commands that do not logically belong inside a function.
- Unlike a function file, a script file must not begin with the keyword function..
- Variables named in a script file are not local variables, but are in the same scope as the other variables entered at the prompt.

Some useful commands

- `-ls, dir, pwd, clear, clc()`
- `-help commandname`
- `-command; or command ?`
- `zeros(m,n), ones(m,n), eye(m)`
- `inv(a), a=[m:n:p], a(m,n)... etc`
- `.*(for scalar and vector) and *, .^`
- `linspace(x1,x2,N); logspace(x1,x2,N);`
- `clf()`

Graphical output

- Octave plotting functions use gnu plot to handle the actual graphics.
 - There are two low-level functions, gplot and gspot, that behave almost exactly like the corresponding gnuplot functions plot and splot.
 - A number of other higher level plotting functions, patterned after the graphics functions found in MATLAB version

Two dimensional plotting

- The MATLAB-style two-dimensional plotting commands are:
 - – `plot(x,y,fmt ...)`
 - – `axis(limits)`
 - – `hold on|off`
 - – `ishold`
 - – `replot`
 - – `clearplot`
 - – `closeplot`

Gnu plot

Gnuplot is a portable command-line driven interactive data and function plotting utility for UNIX, IBM OS/2, MS Windows, DOS, Macintosh, VMS, Atari and many other platforms. The software is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally intended as to allow scientists and students to visualize mathematical functions and data.

Gnuplot is a portable command-line driven interactive data and function plotting utility for UNIX, IBM OS/2, MS Windows, DOS, Macintosh, VMS, Atari and many other platforms. The software is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally intended as to allow scientists and students to visualize mathematical functions and data.

Basic Sine Function

```
octave:>angle=linspace(0,2pi,100);
```

```
octave:>y=sin(angle);plot(angle,y,'option');
```

```
octave:>xlabel("angle"),ylabel("value"),title("sinewave");
```

Options

color- w,r,g,b,k..etc

mark- +,* ,s,p,o..etc

```
octave:>clf(),plot(angle,y,angle,cos(angle)),legend("sine","cosine");
```

hold on and hold off- for plotting in the same window

figure -command for plotting in separate window

Let us Octave

Ex. 1) Simple linear algebra problem

$$x_1 - 12x_2 - 4x_3 = -5$$

$$-20x_1 + 3x_2 - 5x_3 = -119$$

$$-14x_1 - 3x_2 - 17x_3 = -53$$

This can be represented using the matrix equation $Ax=b$

Using Octave, this can be solved as follows:

```
octave:>A=[1 -12 -4;-20 3 -5; -14 -3 -17];
```

```
octave:>b=[-5;-119;-53];
```

```
octave:>x=inv(A)*b;
```

```
octave:>x
```

7

2

-3

Simple?

Let us Octave

Ex. 2) Basic Signal Processing

Define a basic sine wave; $x(k)=3\sin(2\pi k/10)$ as

```
octave:51>for k=1:256;s(k)=cos(2*pi*k/10);end
```

Generate some Gaussian noise (no of cycles ?)

```
octave:52>n=2*randn(256,1);
```

Add it to the signal

```
octave:54>x=s+n';
```

Take the magnitude of the Fourier transform of the time series

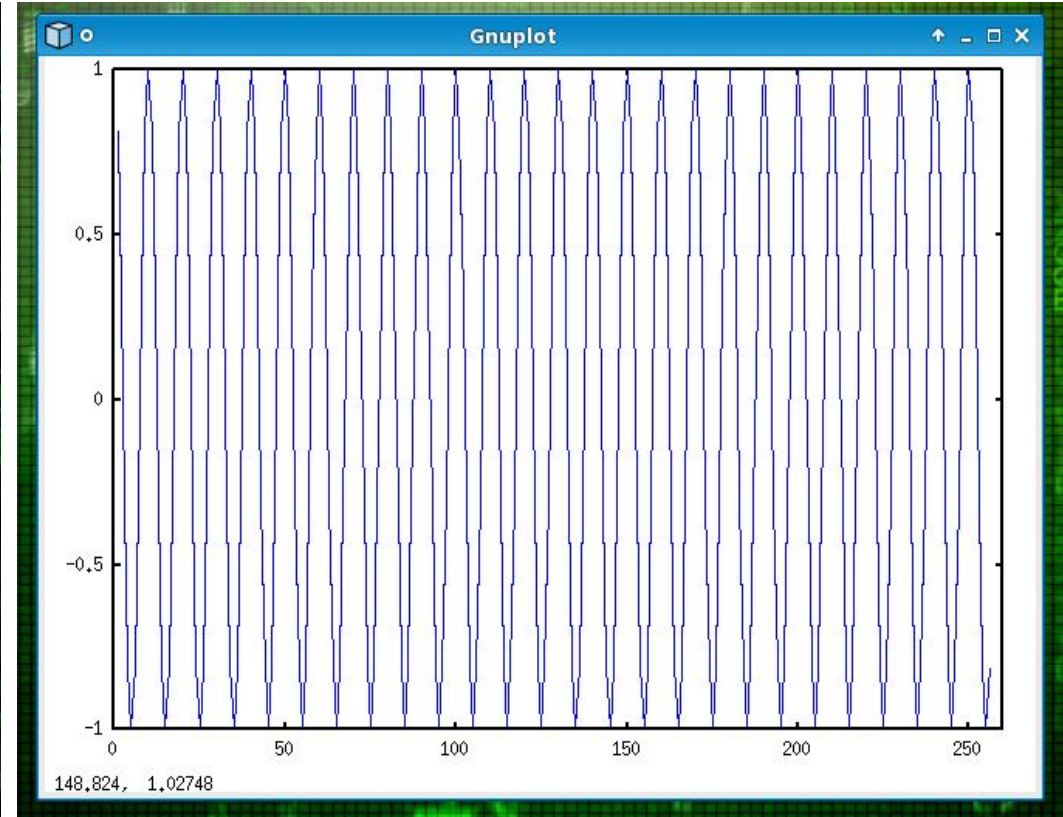
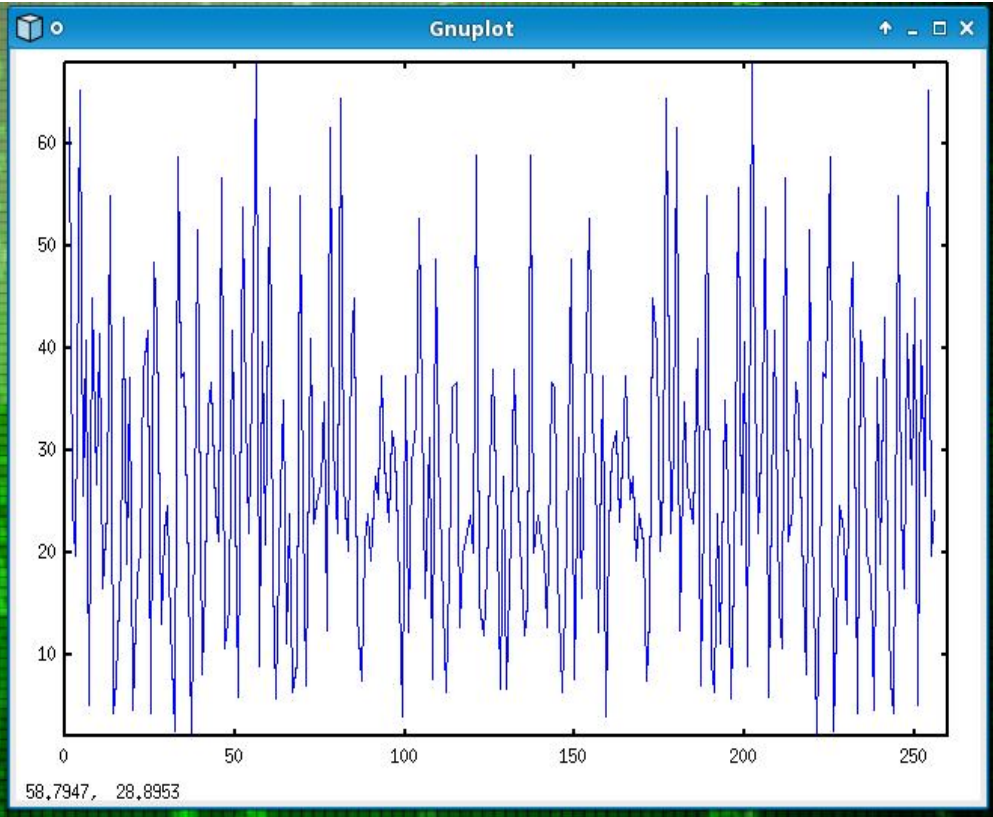
```
octave:55>p=abs(fft(x));
```

Plot the result

```
octave:56>plot(p),axis([xmin,xmax,ymin,ymax]),print("graph.png");
```

Signal(t) + noise (t)
plot(p)

Signal(t)
plot(s)



Let us Octave

Ex.3) Rectified sine wave

```
plot(angle,abs(sin(angle)));
```

Octave updates

GNU Octave is freely redistributable software. You may redistribute it and/or modify it under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation. Octave was written by John W. Eaton and many others. Because Octave is free software you are encouraged to help make Octave more useful by writing and contributing additional functions for it, and by reporting any problems you may have. Visit the Octave web site: <http://www.che.wisc.edu/octave/octave.html>

