

Introduction to GNU Octave

Alexander B. Pacheco

User Services Consultant
LSU HPC & LONI
sys-help@loni.org

HPC Training Series
Louisiana State University
Baton Rouge
Apr. 10, 2013



- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs
- 5 Loops and Conditions
- 6 File I/O
- 7 Functions
- 8 Scripting

- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs
- 5 Loops and Conditions
- 6 File I/O
- 7 Functions
- 8 Scripting



- Octave is a high-level language, primarily intended for numerical computations.
- It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments.
- It may also be used as a batch-oriented language.
- Octave is often viewed as a system for numerical computations with a language that is mostly compatible with MATLAB, but that is available as free software under the GNU GPL, and that can replace it in many circumstances.

Tutorial Goals

- The goal of this tutorial is to provide a brief introduction to a few of the capabilities of GNU Octave.
- Most of the functionality of MATLAB already exists in GNU Octave and octave can run most MATLAB scripts.
- MATLAB users should review differences between MATLAB and GNU Octave before porting MATLAB scripts to octave. http://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB



- GNU Octave is mostly compatible with MATLAB. However, Octave's parser allows some (often very useful) syntax that MATLAB's does not, so programs written for Octave might not run in MATLAB.
- Octave supports the use of both single and double quotes. MATLAB only supports single quotes, which means parsing errors will occur if you try to use double quotes (e.g. in an Octave script when run on MATLAB).
- Octave supports C-style autoincrement and assignment operators, MATLAB does not
`i++; ++i; i+=1; etc.`
- Octave supports temporary expressions.

```
columns = size(mtx)(2); % works in Octave, fails in MATLAB
tmp = size(mtx);
columns = tmp(2); % works in both
```

- MATLAB (7.0) and Octave (3.0.2) responds differently when computing the product of boolean values:

```
X = ones(2,2) ; prod(size(X)==1)
```

```
MATLAB: ??? Function 'prod' is not defined for values of
class 'logical'.
```

```
Octave: ans = 0
```



- MATLAB will execute a file named 'startup.m' in the directory it was called from on the command line. Octave does not. It will, however, execute a file named '.octaverc' which can be edited to execute existing files
- MATLAB lets you load empty files, OCTAVE does not.
- MATLAB doesn't support 'printf' as a command for printing to the screen.
- MATLAB doesn't allow whitespace before the transpose operator.
- MATLAB always requires '.' for line continuation.
- MATLAB uses the percent sign '%' to begin a comment. Octave uses both the hash symbol '#' and the percent sign '%' interchangeably.
- For exponentiation, Octave can use '^' or '**'; MATLAB requires '^'.
- For string delimiters, Octave can use ' or " ; MATLAB requires '.
- For ends, Octave can use 'end' or specify the block with 'end{if,for, . . . }'; MATLAB requires 'end'.
- Octave supports C-style hexadecimal notation (e.g. "0xF0"); MATLAB requires the hex2dec function (e.g. "hex2dec('F0')").
- The main difference is the lack of GUI for Octave.

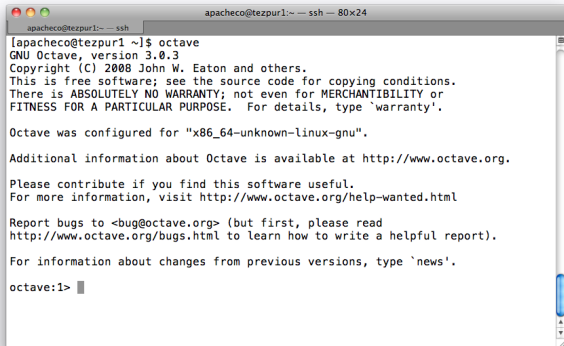
- Users can install GNU Octave on their laptops and desktops.
- Mac OS X: http://www.octave.org/wiki/index.php?title=Installing_MacOS_X_Bundle
- Windows: http://www.octave.org/wiki/index.php?title=Octave_for_Windows
- Linux: Check repositories for your distribution
 - 1 openSuSE: `zypper in octave`
 - 2 Ubuntu: `apt-get install octave`
 - 3 Fedora: `yum install octave`
- LSU HPC & LONI: Add soft key `+octave-3.0.3-intel-11.1` to your `.soft` file and resoft.
- SuperMike II: Add soft key `+octave-3.6.4-gcc-4.4.6` to your `.soft` file and resoft.



- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs
- 5 Loops and Conditions
- 6 File I/O
- 7 Functions
- 8 Scripting



- Type `octave` in a terminal window (Linux, LSU HPC & LONI only)



```
apacheco@tezpurl1:~ -- ssh -- 80x24
apacheco@tezpurl1:~ -- ssh
[apacheco@tezpurl1 ~]$ octave
GNU Octave, version 3.0.3
Copyright (C) 2008 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type `warranty'.

Octave was configured for "x86_64-unknown-linux-gnu".

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/help-wanted.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).

For information about changes from previous versions, type `news'.

octave:1> █
```

- The last line is the octave prompt
- You can also start octave with the `-q` option, octave information will not be printed

- Commands can be typed at the prompt or read from a script.
- Scripts are plain text files with file suffix `.m`.
- To run a script, type the script name without the suffix.
- `","` separates commands in a line and displays the output on the screen.
- To suppress output, use `;"`.
- Comments are preceded by `%`.
- Octave is case-sensitive.
- Getting Help:
 - `help` lists all built-in functions and internal variables.
 - `help name` explains the variable or function "name"

- Octave as a calculator:
 - Just type mathematical commands at the prompt.
 - Octave also has mathematical built-in functions.

Simple Arithmetic functions

Addition: +

Subtraction: -

Multiplication: *

Division: /

Exponentiation: ** or ^

Mathematical Functions

Trigonometric Functions: sin, cos, tan

Inverse Trigonometric Functions: asin, acos, atan

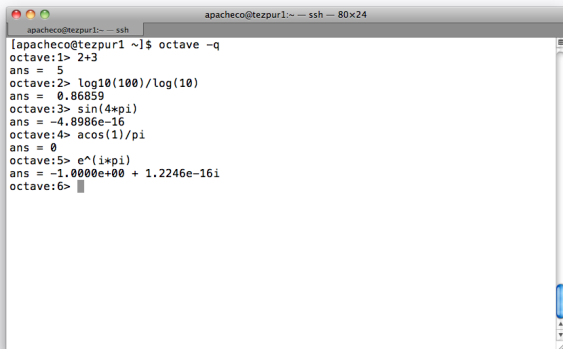
Logarithmic Functions: log, log10

Exponentiation: exp

Absolute value: abs

- trigonometric functions work in **radians**
- **pi**, **e**, **i** and **j** are predefined variables.
- **ans** variable is used to hold the result of the last operation.
- No need to declare variable or its type, variables are either **floating point** numbers or **strings**.
- To see the value of a variable, just type its name and hit return.





```
apache@tezpur1:~ -- ssh -- 80x24
[apache@tezpur1 ~]$ octave -q
octave:1> 2+3
ans = 5
octave:2> log10(100)/log(10)
ans = 0.86859
octave:3> sin(4*pi)
ans = -4.8986e-16
octave:4> acos(1)/pi
ans = 0
octave:5> e^(i*pi)
ans = -1.0000e+00 + 1.2246e-16i
octave:6> █
```

- Note that by default, octave prints variables with only 5 digits.
- At octave prompt, type `format long` to obtain variables with greater precision.

- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices**
- 4 Plotting Graphs
- 5 Loops and Conditions
- 6 File I/O
- 7 Functions
- 8 Scripting



- Defining a Row Vector: $vr = (1\ 2\ 3)$

```
octave:1> vr = [1 2 3]
vr =
```

```
1    2    3
```

- Defining a Column Vector: $vc = (vr)^T$

```
octave:2> vc = [1; 2; 3]
vc =
```

```
1
2
3
```

- Autogeneration of Vector with constant increment: `Start : [increment] : End`

```
octave:3> a = 4:2:10
a =
```

```
4    6    8   10
```

- A matrix $B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ is constructed as follows

```
octave:4> B = [1 2; 3 4]
```

```
B =
```

```
1 2
3 4
```

- Matrices can be assembled from submatrices

```
octave:5> b = 5:6
```

```
b =
```

```
5 6
```

```
octave:6> A = [ B b' ]
```

```
A =
```

```
1 2 5
3 4 6
```



- Creating special matrices of size $m \times n$
 - 1 eye(m,n): Create a matrix with ones on the diagonal and zeros elsewhere. Identity matrix if $m = n$
 - 2 zeros(m,n): Create a null matrix
 - 3 ones(m,n): Create a matrix where all elements are 1
 - 4 rand(m,n): generates a random matrix whose entries are uniformly distributed in the interval (0,1).

```
octave:7> eye(3,2)
ans =
```

```
1 0
0 1
0 0
```

```
octave:8> eye(3,3)
ans =
```

```
1 0 0
0 1 0
0 0 1
```

```
octave:9> zeros(3,2)
ans =
```

```
0 0
0 0
0 0
```

```
octave:10> ones(3,4)
ans =
```

```
1 1 1 1
1 1 1 1
1 1 1 1
```

```
octave:11> rand(5,2)
ans =
```

```
1.7700e-01 1.4495e-01
7.5533e-01 7.9854e-01
3.4831e-04 7.6881e-01
5.6224e-01 1.0213e-01
5.3236e-01 9.2427e-01
```


- Basis Arithmetic:

- $+$, $-$ and $*$ denote matrix addition, subtraction and multiplication respectively.
- A' : transpose and conjugates A
- $A.'$: transposes A

```
octave:1> A = [1+i 2+2i;3-i 4-2i]
```

```
A =
```

```
 1 + 1i   2 + 2i
 3 - 1i   4 - 2i
```

```
octave:3> A+B,A*B
```

```
ans =
```

```
 3 + 1i   4 + 2i
 5 - 1i   6 - 2i
```

```
ans =
```

```
 6 + 6i   6 + 6i
14 - 6i  14 - 6i
```

```
octave:2> B = 2*ones(2,2)
```

```
B =
```

```
 2   2
 2   2
```

```
octave:4> A',A.'
```

```
ans =
```

```
 1 - 1i   3 + 1i
 2 - 2i   4 + 2i
```

```
ans =
```

```
 1 + 1i   3 - 1i
 2 + 2i   4 - 2i
```



- Element wise operations: Use `.{operator}` for element wise operation

```
octave:1> A = [ 1 2; 3 4]
A =
```

```
 1  2
 3  4
```

```
octave:2> A.**2
ans =
```

```
 1  4
 9 16
```

```
octave:3> B = [ 2 2; 6 4]
B =
```

```
 2  2
 6  4
```

```
octave:4> A./B
ans =
```

```
 0.50000  1.00000
 0.50000  1.00000
```



- Indexing and Slicing
 - $v(k)$: k^{th} element of vector v
 - $A(k, l)$: matrix element A_{kl}
 - $v(m : n)$: slice of vector v from element m through n
 - $A(k, :)$: k^{th} row of matrix A
 - $A(:, l)$: l^{th} column of matrix A
- `length(v)`: returns the number of elements of vector v
- `size(A)`: returns the number of rows and columns of matrix A

- GNU Octave is capable of solving Linear Algebra problems

⑦ Solving $Ax = b$

Solve the equations $x + y = 3$ and $2x - 3y = 5$

```
octave:1> A = [ 1 1; 2 -3] , B = [3 5]'
```

```
A =
```

```
  1  1
  2 -3
```

```
B =
```

```
  3
  5
```

```
octave:2> A \ B
```

```
ans =
```

```
  2.80000
  0.20000
```

```
octave:3> inv(A) * B
```

```
ans =
```

```
  2.80000
  0.20000
```

```
octave:4> A * (A \ B)
```

```
ans =
```

```
  3
  5
```

- Calculate determinant of a matrix: $\det(A)$
- Calculate inverse of a matrix: $\text{inv}(A)$

- Calculate eigenvectors and eigenvalues of a matrix

```
octave:1> A = [1 2 3; 4 5 6; 7 8 9]
A =
```

```
  1  2  3
  4  5  6
  7  8  9
```

```
octave:2> eig(A)
ans =
```

```
 1.6117e+01
-1.1168e+00
-1.3037e-15
```

```
octave:3> [V,D] = eig(A)
V =
```

```
-0.231971 -0.785830  0.408248
-0.525322 -0.086751 -0.816497
-0.818673  0.612328  0.408248
```

```
D =
```

```
Diagonal Matrix
```

```
 1.6117e+01      0      0
      0 -1.1168e+00      0
      0      0 -1.3037e-15
```

- To calculate eigenvectors, you need to provide two variables for the answer.
- Check if we can obtain A by evaluating $A = VDV^{-1}$

- For non square matrix, Octave can also carry out Singular Value Decomposition (SVD)
- SVD takes a $m \times n$ matrix A and factors it into $A = USV^T$ where
 - U is a $m \times m$ orthogonal matrix whose columns are eigenvectors of AA^T
 - V is a $n \times n$ orthogonal matrix whose columns are eigenvectors of $A^T A$
 - S is a $m \times n$ diagonal matrix whose elements are the square roots of the eigenvalues of AA^T and $A^T A$

```

octave:1> A = [ 1 3 -2 3; 3 5 1 5; -2 1 4 2]
A =
    1    3   -2    3
    3    5    1    5
   -2    1    4    2

octave:2> svd(A)
ans =
    8.9310
    5.0412
    1.6801

octave:3> [U,S,V] = svd(A,0)
U =
   -4.6734e-01   3.8640e-01   7.9516e-01
   -8.6205e-01   3.3920e-04  -5.0682e-01
   -1.9611e-01  -9.2233e-01   3.3294e-01

S =
    8.93102   0.00000   0.00000
    0.00000   5.04125   0.00000
    0.00000   0.00000   1.68010

V =
   -0.297983   0.442764  -0.828029
   -0.661559   0.047326   0.109729
   -0.079700  -0.885056  -0.455551
   -0.683516  -0.135631   0.307898

octave:4> U*S*V'
ans =
    1.0000    3.0000   -2.0000    3.0000
    3.0000    5.0000    1.0000    5.0000
   -2.0000    1.0000    4.0000    2.0000
  
```

- A/B computes X such that $XB = A$.
- $A \setminus B$ computes X such that $AX = B$.
- $\text{norm}(A, p)$ computes p -norm of the matrix (or vector) A , default is $p = 2$.
- $\text{rank}(A)$ computes the numerical rank of matrix A .
- $\text{trace}(A)$ computes the trace of a matrix A .
- $\text{logm}(A)$ computes the matrix logarithm of a square matrix.
- $\text{expm}(A)$ computes the matrix exponential of a square matrix.
- $\text{sqrtn}(A)$ computes the matrix square root of a square matrix.
- $R = \text{chol}(A)$ computes the Cholesky factorization of the symmetric definite matrix A such that $R^T R = A$
- $[L, U] = \text{lu}(A)$ computes the LU decomposition of A , $A = LU$
- $[Q, R] = \text{qr}(A)$ computes the QR decomposition of A , $A = QR$



- Numerical Integration
- Differential Equations
- Polynomial Manipulations
- Statistical Analysis
- Interpolation
- Signal and Image Processing
- Object Oriented Programming

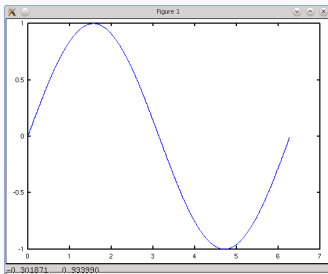
- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs**
- 5 Loops and Conditions
- 6 File I/O
- 7 Functions
- 8 Scripting



- Octave can plot graphs via the open-source program GNU PLOT
- If given just one pair of numbers, it plots a point.
- If given a pair of vectors, it plots all the points given by the two vectors.
- Usage:
 - `plot(x, y[, fmt])`: Plots a line through the points (x_i, y_i) . Select line style and color with the `fmt` string.
 - `semilogx(x, y[, fmt])`: Plot with a logarithmic scale for the x-axis
 - `semilogy(x, y[, fmt])`: Plot with a logarithmic scale for the y-axis
 - `semilog(x, y[, fmt])`: Plot with a logarithmic scale on both axes



- Procedure for plotting 2D graphics: $y = f(x)$
 - 1 Generate a vector with the x-coordinates to be plotted.
octave:1> `x = 0:0.01:2*pi;`
 - 2 Generate a vector containing the corresponding y-values
octave:2> `y=sin(x);`
 - 3 Use the plot command to plot $\sin(x)$
octave:3> `plot(x,y);`

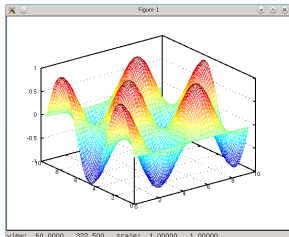


- Procedure for plotting 3D graphics
 - 1 Generate a grid data for a 3D plot: Requires two matrices xx whose rows are copies of x and yy whose columns are copies of y

```
octave:1> x = 0:0.1:3*pi;  
octave:2> y = 0:0.1:3*pi;  
octave:3> [xx,yy]=meshgrid(x,y);
```

- 2 Plot a surface in 3D

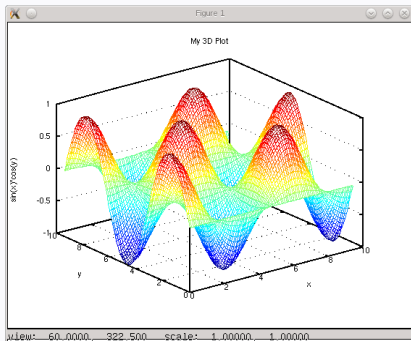
```
octave:4> z = sin(x)' * cos(y);  
octave:5> mesh(x,y,z);
```



- `title(string)` writes `string` as title for the graphics
- `xlabel(string)` labels the `x`-axis with `string`
- `ylabel(string)` labels the `y`-axis with `string`
- `zlabel(string)` labels the `z`-axis with `string`
- `axis(v)` set axes limits for the plot. `v` is a vector of the form `v = (xmin, xmax, ymin, ymax[, zmin, zmax])`.
- `hold [on|off]` controls whether the next graphics output should or shouldn't clear the previous graphics.
- `clf` clears the graphics window.



```
octave:6> title('My 3D Plot');  
octave:7> xlabel('x');  
octave:8> ylabel('y');  
octave:9> zlabel('sin(x)*cos(y)');
```



- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs
- 5 Loops and Conditions**
- 6 File I/O
- 7 Functions
- 8 Scripting



- Control Statements such as `if`, `do`, `while` and so on control the flow of execution in Octave Programs.
- Each control statement has an `end` statement that marks the end of the control statement.

The IF Statement

The general form of the IF statement is

```
if (condition)
  then-body
elseif (condition)
  elseif-body
else
  else-body
endif

if (rem (x,2) == 0)
  printf ("x is even\n");
elseif (rem (x,3) == 0)
  printf ("x is odd and divisible by 3\n");
else
  printf ("x is odd\n");
endif
```

The `elseif` and `else` blocks are optional



The SWITCH Statement

The SWITCH statement is similar to the SELECT CASE statement in Fortran 90 and allows one to carry out different operation based on one variable.

```
switch expression          a = rem (y,2);
  case label               switch a
    command_list          case 0
  case label               printf ("x is even\n");
    command_list          otherwise
  otherwise                b = rem (y,3);
    command_list          switch b
endswitch                  case 0
                           printf ("x is odd and divisible by 3\n");
                           otherwise
                             printf ("x is odd\n")
                           endswitch
                           endswitch
```



The WHILE Statement

The `WHILE` statement is the simplest form of a looping construct. It repeatedly executes a statement as long as a condition is true

```
while (condition)
  body
endwhile

fib = ones(1, 10);
i = 3;
while ( i <= 10 )
  fib(i) = fib(i-1) + fib(i-2);
  i++;
endwhile
```



The DO-UNTIL Statement

The `DO-UNTIL` statement is similar to the `WHILE` statement except that it repeatedly executes a statement until a condition is true. The test of the condition is at the end of the loop so that the loop executes at least once.

```
do                                fib = ones(1, 10);
  body                            i = 2;
until (condition)                do
                                  i++;
                                  fib(i) = fib(i-1) + fib(i-2)
                                  until ( i == 10)
```



The FOR Statement

The FOR statement makes it convenient to count iterations of a loop

```
for var = expression          for I=1:N
    body                      sumft=0;
endfor                       for J=1:M
                              sumft=sumft+ftmwvels(I,J)^2;
                              endfor
                              fprintf(fid," %15.8e  %21.14e\n", (I-1)*dw, sumft);
                              endfor
```



The BREAK and CONTINUE Statements

The BREAK statement allows you to jump from the inside of a loop past the end of a loop

The CONTINUE statement allows you to jump from the inside of a loop to the beginning of the loop

The BREAK and CONTINUE can be used inside a for, while or do...until loop.

```
total = 0;
while true
    x = input('Value to add (enter 0 to stop): ');
    if x == 0
        break;
    endif
    total = total+x;
    disp(['Total: ', num2str(total)]);
endwhile

N = 5;
A = zeros(N); % Create an N x N matrix filled with 0s

for row = 1:N
    for column = 1:N
        if column > row
            continue;
        endif
        A(row, column) = 1;
    endfor
endfor
```



- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs
- 5 Loops and Conditions
- 6 File I/O**
- 7 Functions
- 8 Scripting



- save and load commands allow data to be written to and read from disk files in various formats.

```
apacheco-3:~ apacheco> octave -q
octave-3.2.3:1> A = [1 2 3; 4 5 6; 7 8 9];
octave-3.2.3:2> save myfile.mat A
octave-3.2.3:3>
apacheco-3:~ apacheco> cat myfile.mat
# Created by Octave 3.2.3, Fri Apr 13 12:53:36 2012 CDT <apacheco@apacheco-3.lsu.edu>
# name: A
# type: matrix
# rows: 3
# columns: 3
 1 2 3
 4 5 6
 7 8 9
apacheco-3:~ apacheco> octave -q
octave-3.2.3:1> load myfile.mat
octave-3.2.3:2> A
A =

    1    2    3
    4    5    6
    7    8    9
```

- Octave can save and read data in various formats such as ASCII, binary, MATLAB binary format, hdf5.



- Octave C style input and output functions provides most of the functionality of C programming language's standard I/O library.
- Opening and Closing Files: When reading or writing data to a file, it must be opened first

```
fid = fopen("water-hexamer-cage-bomd.ftmwvels", "w");  
dw = 1/(N*deltat);  
for I=[1:N]  
    sumft=0;  
    for J=[1:M]  
        sumft=sumft+ftmwvels(I, J)^2;  
    endfor  
    fprintf(fid, " %15.8e %21.14e\n", (I-1)*dw, sumft);  
endfor  
fclose(fid);
```

- Octave can open files in various modes, the above example is for writing.

- r : Open a file for reading.
- w : Open a file for writing.
- a : Open or create a file for writing at the end of the file.
- r+ : Open an existing file for reading and writing.
- w+ : Open a file for reading and writing and discard previous contents.
- a+ : Open or create a file for reading and writing at the end of the file.



- Octave provides function for printed formatted output which are modelled on the C language functions.
`printf (template, ...)` : Print optional arguments under the control of string `template`.
`fprintf (fid, template, ...)` : Same as `printf` except that output is written to stream `fid` instead of `stdout`.
`sprintf (template, ...)` : Same as `printf` except that the output is returned as a string.
- Output Conversion Syntax:

General Form: `% flags width [.precision] type conversion`

- `%7d` : Prints an integer with a width of 7.
- `%8.3f` : Prints a floating point number with a width of 8 and precision of 3.
- `%11s` : Prints a string of width 11.
- `%21.14e` : Prints a floating point number in exponential notation with a width of 21 and precision 14.



- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs
- 5 Loops and Conditions
- 6 File I/O
- 7 Functions**
- 8 Scripting

- Complicated Octave programs can be simplified by defining functions.
- In the simplest form, a function, name, is defined as follows:

```
function name (argument-list)
    body
endfunction

[apacheco@tezpurl octave]$ cat hello.m
function hello (message)
    printf ("%s\n",message)
endfunction
[apacheco@tezpurl octave]$ octave -q
octave:1> hello ("Hello World")
Hello World
```

- In some instances, your program may need some information back from the function that you have defined.
- Syntax for writing a function that returns

1 a single value

```
function ret-var = name (argument-list)
    body
endfunction
```

2 multiple values

```
function [ret-list] = name (argument-list)
    body
endfunction
```



- The symbol `ret-var` is the name of the variable that will hold the value to be returned by the function.
- The symbol `ret-list` is a comma separated list of variables that will hold the values returned by the function.
- `ret-var` and `ret-list` must be defined before the end of the function body.
- Variables defined in the function including `ret-var`, `ret-list` and `argument-list` are local to the function.

```
v = rand(10,1);

function average = avg (a)
    average = sum(a)/length(a) ;
endfunction
```

```
function [max,id] = vmax(a)
    id = 1;
    max = a(id);
    for i = 2:length(a)
        if ( a(i) > max )
            max = a(i) ;
            id = i ;
        endif
    endfor
endfunction
```

```
b = rand(20,1);
[max,id] = vmax(b);
printf ( "Average of vector v = %f\n", avg(v))
printf ( "Maximum value of vector b = %f with \
        id = %d\n",max,id )
```

```
[apacheco@tezpurl octave] ./func.sh
Average of vector v = 0.512198
Maximum value of vector b = 0.996040 with id = 7
```



- Instead of defining functions each time you need them, you can save the function to a Function File and use them whenever needed.
- Function Files end with a `.m` extension with a prefix that matches the function name.
- Function files should contain only one function, see `hello.m` two slides back.
- When a function is called, octave searches a list of directory for a file that contains the function declaration.
- If the function file is not in the current directory, you can add the directory to search for the function file using the `addpath` command

```
addpath ("~/Octave:~/Octave-Func"): Add ~/Octave and  
~/Octave-Func to the load path for searching function files.
```



- 1 Introduction
- 2 Getting Started
- 3 Vectors and Matrices
- 4 Plotting Graphs
- 5 Loops and Conditions
- 6 File I/O
- 7 Functions
- 8 Scripting



- A script file is a file containing any sequence of Octave commands.
- It is read and evaluated as if you have entered the commands interactively at the octave prompt.
- A script file must not begin with a function keyword. If the script file begins with a function keyword, Octave will assume that it is a function file and will evaluate only the function when it is called.
- Unlike a function file, variables in the script file are global and can be accessed by any line/command in the script.
- Octave normally executes commands from a script file with a `.m` extension.
- The `source` command allows Octave to execute commands from any file
`source(file.exe)` to execute commands in the `file.exe` file.



```
[apacheco@tezpurl octave]$ cat func.txt
v = rand(10,1);

function average = avg (a)
    average = sum(a)/length(a) ;
endfunction

function [max,id] = vmax(a)
    id = 1;
    max = a(id);
    for i = 2:length(a)
        if ( a(i) > max )
            max = a(i) ;
            id = i ;
        endif
    endfor
endfunction

b = rand(20,1);
[max,id] = vmax(b);

printf ( "Average of vector v = %f\n", avg(v) )
printf ( "Maximum value of vector b = %f with id = %d\n",max,id )

[apacheco@tezpurl octave]$ octave -q
octave:1> source('func.txt')
Average of vector v = 0.487381
Maximum value of vector b = 0.878363 with id = 14
```



- Octave also allows you to create an executable script file.
- The first line of the executable script file should refer the interpreter.
- Octave Executable scripts can take command line arguments.
- The built-in function `argv` returns a cell array containing the command line arguments passed to the executable script.
- The built-in function `nargin` returns the number of arguments passed.
- On Tezpur, first line should be

```
#!/usr/local/packages/octave/3.0.3/intel-11.1/bin/octave -qf
```



```
[apacheco@tezpurl octave]$ cat func.sh
#!/usr/local/packages/octave/3.0.3/intel-11.1/bin/octave -qf

v = rand(10,1);

function average = avg (a)
    average = sum(a)/length(a) ;
endfunction

function [max,id] = vmax(a)
    id = 1;
    max = a(id);
    for i = 2:length(a)
        if ( a(i) > max )
            max = a(i) ;
            id = i ;
        endif
    endfor
endfunction

b = rand(20,1);
[max,id] = vmax(b);

printf ( "Average of vector v = %f\n", avg(v) )
printf ( "Maximum value of vector b = %f with id = %d\n",max,id )

[apacheco@tezpurl octave]$ ls -l func.sh
-rwxr-xr-x  1 apacheco Admins 450 Apr 16 11:39 func.sh
```



```
[apacheco@tezpurl octave]$ ./func.sh  
Average of vector v = 0.599684  
Maximum value of vector b = 0.986472 with id = 15
```



- Octave script to calculate Fourier Transform of Auto-Correlation Function of Mass Weighted Velocities. /home/apacheco/octave-tutorial/getftmwvels.sh

```
[apacheco@tezpurl octave-tutorial]$ cat getftmwvels.sh
#!/usr/local/packages/octave/3.0.3/intel-11.1/bin/octave -qf

if(nargin!=4)
    printf ("%s\n", "This script needs 4 arguments, [Velocity File (input)], \  
[FT-VAC File (output)], [Number of Atoms], [Time Step in fs]" )
endif

arg_list = argv();
printf ("Argument list:");
for i = 1:nargin
    printf (" %s", arg_list{i});
endfor
printf ("\n");

Vels = arg_list{1};
VelsFT = arg_list{2};
NAtoms = eval(arg_list{3});
deltat = eval(arg_list{4});

M = NAtoms*3;
mwvels = load(Vels);
N = rows(mwvels);
ftmwvels = abs(fft(mwvels,N));
fid = fopen(VelsFT,"w");
```



```
dw = 1/(N*deltat);
for I=[1:N]
    sumft=0;
    for J=[1:M]
        sumft=sumft+ftmwvels(I,J)^2;
    endfor
    fprintf(fid," %15.8e %21.14e\n", (I-1)*dw, sumft);
endfor
fclose(fid);

# Below was added for octave tutorial
A=load(VelsFT);
int=A(:,1)*33356; # convert from fs^-1 to cm^-1
spectra_orig=A(:,2)/norm(A(:,2)); # normalized original spectra
# Do interpolation of Spectra for octave demo
intensity=linspace(0,4000,400000);
# Linear Interpolation
spectra_lin=interp1(int,spectra_orig,intensity,'linear');
# Spline Interpolation
spectra_spl=interp1(int,spectra_orig,intensity,'spline');
# Cubic Interpolation
spectra_cub=interp1(int,spectra_orig,intensity,'cubic');

plot(int,spectra_orig,'r',intensity,spectra_lin+0.2,'g', \
      intensity,spectra_spl+0.4,'b',intensity,spectra_cub+0.6,'m');

legend("original","linear","spline","cubic");
axis([0,4000,0,1]);
```

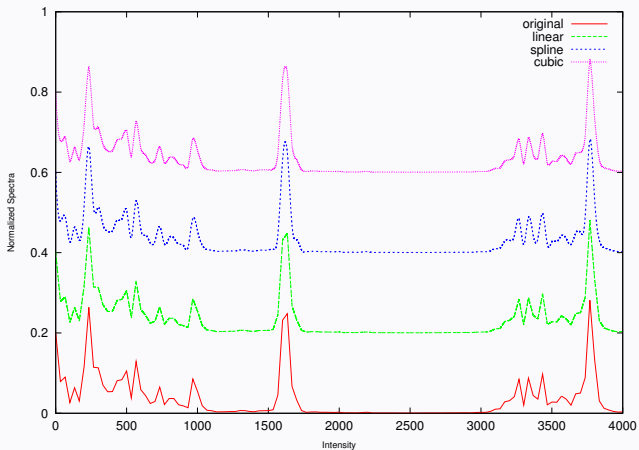


```
xlabel('Intensity');  
ylabel('Normalized Spectra');  
  
print -deps spectra.eps;
```

- Usage:

```
[apacheco@tezpurl water-hexamer]$ ./getftmwvels.sh water-hexamer-cage-admp.mwvels \  
> water-hexamer-cage-admp.ftmwvels 18 0.25  
Argument list: water-hexamer-cage-admp.mwvels water-hexamer-cage-admp.ftmwvels 18 0.25  
  
[apacheco@tezpurl water-hexamer]$
```





- 1 Octave Manual: <http://www.gnu.org/software/octave/doc/interpreter/index.html> and <http://www.gnu.org/software/octave/octave.pdf>
- 2 Octave-Forge: <http://octave.sourceforge.net/>
- 3 Octave Tutorials: http://en.wikibooks.org/wiki/Octave_Programming_Tutorial

The End

Any Questions?

Next Week

Introduction to GNUPlot

Survey:

<http://www.hpc.lsu.edu/survey>

