Octave Lab

Laboratory 1 Octave Tutorial

1.1 Introduction

The purpose of this lab[1] is to become familiar with the GNU Octave[2] [3] software environment.

1.2 Octave Review

All laboratory tasks will be performed using GNU Octave. Octave is a high level language, intended for numerical computations. It has a command line interface (CLI) and a graphics user interface (GUI) for solving linear and nonlinear problems numerically. Octave can be used for numerical analysis, matrix computation, signal processing, and graphics. Octave for Windows© installer can be downloaded from <www.gnu.org/software/octave/download/>. In this section, we will review some of its basic functions.

1.2.1 Starting Octave and Help

Start Octave on your workstation by double clicking the "Octave-x.x.x (GUI)' shortcut in the desktop.

After starting up, you will get an Octave prompt "`>>`" in the Command Window of the graphics user interface (GUI). To get help on any specific command, such as "plot", type the following.

```
>> help plot <enter>
```

in the Command Window. You can interrupt Octave by typing `Control-C` (usually written as `C-c`). Type `quit` or `exit` at the Octave prompt to end Octave.

1.2.2 Statements and Variables

Statements have the form

```
variable=expression
```

in Octave. A typical statement entering a 2 x 2 matrix in variable "a"

```
>> A=[1 2; 4 6] <enter>
```

[1] R.C. Dorf and R.H. Bishop, *Modern Control Systems*, 8th ed. Reading, MA: Addison-Wesley, 1998, pp. 818-834

[2] <http://www.gnu.org/software/octave/>

[3] J.W. Eaton, *GNU Octave Manual*, 2013 <https://www.gnu.org/software/octave/doc/interpreter/>

```
A =
  1  2
  4  6
```

the statement is executed after the enter key is pressed. The matrix "A" is automatically displayed after the statement is executed. The output display can be suppressed by following the statement with a semicolon (`;`).

```
>> A=[1 2; 4 6];
>>
```

Octave can be used in "calculator" mode. Omitting the variable and "`=`" from an expression assigns the result to `ans`.

```
>> 12.4/6.9
ans = 1.7971
```

Variable names consist of a letter followed by any number of letters, digits or underscores. Octave has no limit on the length of variable names.

```
>> num_students=25
num_students=25
```

Octave is case sensitive, the variables `A` and `a` are not the same.

```
>> a=[3 2 7]
a =

  3  2  7
```

Octave has predefined variables including `pi`, `Inf`, `Nan`, `i`, and `j`. `Nan` stands for *Not-a-Number* and results from undefined operations. `Inf` represents $+\infty$, and `pi` represents π. The variable `i` and `j` $= \sqrt{-1}$ and is used to represent complex numbers.

```
>> z=2.3+4*i
z = 2.3000 + 4.0000i

>> Inf
ans = Inf

>> 0/0
warning: division by zero
ans = NaN
```

Variables are stored in a workspace. The `whos` functions lists the variables in the workspace and gives additional information regarding variable dimension, type and memory allocation.

```
>> whos
Variables in the current scope:

  Attr Name             Size                Bytes  Class
  ==== ====             ====                =====  =====
       A                2x2                    32  double
       a                1x3                    24  double
       ans              1x1                     8  double
       num_students     1x1                     8  double
  c    z                1x1                    16  double

Total is 10 elements using 88 bytes
```

The list is also displayed in the Workspace (`C-3`) window.

Variables can be removed from the workspace with the `clear` function. Using the function `clear` by itself removes all variables and functions from the workspace `clear -variables` removes all variables from the workspace; `clear name1, name2 ...` removes the variables `name1`, `name2`, ...

```
>> clear -variables
>> who
>>
```

The `who` function lists the currently defined variables.

1.2.3 Matrices and Operations

Every element in Octave is a matrix. Vectors and scalars are special cases of matrices. A matrix expression is enclosed in brackets `[.]`. The column elements are separated by blanks or commas and the rows by semicolons or carriage returns. If matrix `A` is defined as

$$A = \begin{bmatrix} 1 & -2j & \sqrt{3} \\ \sin(\pi/2) & \cos(\pi/3) & \arcsin(0.5) \\ \arccos(0.3) & \log(-1) & \exp(0.7) \end{bmatrix}$$

one way to input `A` in Octave is

```
>> A=[1 -2j sqrt(3);
sin(pi/2) cos(pi/3) asin(0.5);
acos(0.3) log(-1) exp(0.7)]
A =

   1.00000 + 0.00000i  -0.00000 - 2.00000i   1.73205 + 0.00000i
   1.00000 + 0.00000i   0.50000 + 0.00000i   0.52360 + 0.00000i
   1.26610 + 0.00000i   0.00000 + 3.14159i   2.01375 + 0.00000i
```

The Octave command

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

  1  2  3
```

creates a matrix named "a" with dimensions of 1 x 3.
Alternatively, the operation

```
>> b=a'
b =

  1
  2
  3
```

stores the transpose of "a" into the vector "b". In this case, "b" is a 3 x 1 vector.
Since each element in Octave is a matrix, the operation

```
>> c=a*b
c = 14
```

computes the matrix product of "a" and "b" to generate a scalar value for "c" of $14 = 1 * 1 + 2 * 2 + 3 * 3$.
Often, you may want to apply an operation to each element of a vector. For example, you many want to square each value of "a". In this case, you may use the following command.

```
>> c=a.*a
c =
  1  4  9
```

The dot before the $*$ tells Octave that the multiplication should be applied to each corresponding element of "a". Therefore the .* operation is not a matrix operation. The dot convention works with many other Octave commands such as divide ./, and power .^.

1.2.4 Expressions

Expressions use familiar arithmetic **operators** and precedence rules.
Common operators are:

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`^`	Power

Numbers in Octave uses conventional decimal notation, with an optional decimal point and leading plus or minus for numbers. Imaginary numbers use either `i` or `j` as a suffix.

Octave performs computations in double precision. However, the screen output can be displayed in several formats. The default output format contains five digits for fixed point. The output format can be changed by using the `format` function.

```
>> pi
pi = 3.1416
>> format long; pi
pi = 3.14159265358979
>> format short e; pi
pi = 3.1416e+00
>> format long e; pi
pi = 3.141592653589793e+00
>> format short
```

The colon notation generates a row vector containing numbers from a given starting value, `xi` , to a final value `xf` , with a specified increment, `dx`.

```
x=[xi:dx:xf]
```

The colon notation is useful for developing **x-y plots**. To generate a plot of $y = x\sin(x)$ versus x for $x = 0, 0.1, 0.2, ..., 1.0$, generate first a table of x-y data.

```
>> x=[0:0.1:1]';y=x.*sin(x);
>> [x y]
ans =

  0.00000  0.00000
  0.10000  0.00998
  0.20000  0.03973
  0.30000  0.08866
  0.40000  0.15577
  0.50000  0.23971
  0.60000  0.33879
  0.70000  0.45095
  0.80000  0.57388
  0.90000  0.70499
  1.00000  0.84147
```

The plot of $y = x\sin(x)$ versus x is a simple step once the table of x-y data is generated.

1.2.5 Mathematical Functions

Octave has built-in trigonometric functions `sin` (sine), `cos` (cosine) and `tan` (tangent) that take an argument in radians; the inverse obtained by appending an `a` as in `asin`, `acos` or

`atan` and returns the value in radians, and exponential functions `exp`, `log` and `log10`. These functions operate on an input and the syntax is `sin(x)` rather than `sin x`.

Other functions:

`round(x)`	Rounds a number to the nearest integer
`ceil(x)`	Rounds a number up to the nearest integer
`floor(x)`	Rounds a number down to the nearest integer
`fix(x)`	Rounds a number to the nearest integer towards zero
`rem(x,y)`	The remainder left after division
`mod(x,y)`	The signed remainder left after division
`abs(x)`	The absolute value of x
`sign(x)`	The sign of x
`factor(x)`	The prime factors of x

1.2.6 Graphics

Octave uses a **graph display** to present plots. The graph display is automatically activated when a plot is generated using any function that generates a plot.

`plot(x,y)`	Plots the vector x versus the vector y.
`semilogx(x,y)`	Plots the vector x versus the vector y. The x-axis is logarithmic; the y-axis is linear.
`semilogy(x,y)`	Plots the vector x versus the vector y. The x-axis is linear; the y-axis is logarithmic.
`loglog(x,y)`	Plots the vector x versus the vector y. Creates a plot with logarithmic scales on both axes.

The plots can be customized by adding titles, axis labels, and text to the plots and to change the scales and display multiple plots.

`title('text')`	Puts 'text' at the top of the plot.
`xlabel('text')`	Labels the x-axis with 'text'.
`ylabel('text')`	Labels the y-axis with 'text'.
`subplot`	Subdivides the graphics windows.
`grid`	Draws grid lines on the current plot.

Plot of $y = x\sin(x)$ versus x.

```
>> x=[0:0.1:1]';y=x.*sin(x);
>> plot(x,y)
>> title('Plot of x sin(x) vs. x')
>> xlabel('x')
>> ylabel('y')
```

```
>> grid
>> refresh
```

Plots can be saved using the `print` command.

```
>> print('plotXY.jpg');
```

1.2.7 Flow Control

Octave has seven flow control statements:

- The `if` statement

  The `if` statement is a decision-making statement. In its simplest form:

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

  condition is an expression that controls what the rest of the statements will do. The *then-body* is executed only if the condition is true.

  The general form of the `if` statement:

  ```
  if (condition)
    command_list
  elseif (condition)
    command_list
  else
    command_list
  endif
  ```

- `switch` statements

  The general form of the `switch` statement:

  ```
  switch expression
    case label
      command_list
    case label
      command_list
      ...
    otherwise
      command_list
  endswitch
  ```

- `while` loops

  The `while` statement repeatedly executes a statement as long as a condition is true.

  ```
  while (condition)
    body
  endwhile
  ```

- `do-until` loops

  The `do-until` statement is similar to the `while` statement except that the test of the condition is at the end of the loop. Hence, the body of the loop is always executed at least once.

  ```
  do
    body
  until (condition)
  ```

- `for` loops

  The `for` statement is a convenient way to count iterations of a loop.

  ```
  for var = expression
    body
  endfor
  ```

- `break` statements

  The `break` statement jumps of the `for` or `while` loop.

- `continue` statements

  The `continue` statement skips over the rest of the loop body.

1.2.8 Octave Scripts and Functions

Octave has two methods for saving sequences of commands as standard files. These two methods are called scripts and functions. Scripts execute a sequence of Octave commands just as if you typed them directly into the Octave command window. Functions differ from scripts because they take inputs and return outputs. A script-file is a text file with the filename extension .m . The file should contain a sequence of Octave commands. The script-file can be run by typing its name at the Octave prompt without the .m extension. This is equivalent to typing in the commands at the prompt. Within the script-file, you can access variables you defined earlier in Octave. All variables in the script-file are global, i.e. after the execution of the script-file, you can access its variables at the Octave prompt.

To plot the equation $y(t) = \sin \alpha t$, where α is a variable that we want to vary, invoke the editor,

```
>> edit
```

Using the Octave editor, write the script and save as `plotsin.m`

```
% A script to plot y=sin(alpha*t)
%
% alpha must exists in the workspace before
% invoking the script
%
t=[0:0.1:1];
y=sin(alpha*t);
plot(t,y)
title('Plot of sin(alpha*t)')
xlabel('t')
ylabel('sin(alpha*t)')
grid
```

Input the value α at the command prompt and execute the script `plotsin`. After executing the script, another value for α can be entered and the script can be executed again.

```
>> help plotsin

>> alpha=10;plotsin
>> alpha=20;plotsin
```

Functions are M-files that accepts input arguments and returns output arguments. To create a function called "func", create a file called "func.m". The file "func.m" should contain:

```
function output = func(input)
    body
endfunction
```

where "input" designates the set of input variables, and "output" are your output variables. The rest of the function file then contains the desired operations. All variables in the function are local; that means the function cannot access Octave workspace variables that you don't pass as inputs. After the execution of the function, you cannot access internal variables of the function.

The following function computes for the area of a circle:

```
function retval=areaCircle(radius)
%
% A function to compute area of a circle
```

```
%   given the radius
%
retval=pi*(radius)^2;
endfunction
```

In the command line, test the function for a circle of `radius = 4`.

```
>> area=areaCircle(4)
area = 50.265
```

1.2.9 EXERCISE

1. Consider the two matrices

$$A = \begin{bmatrix} 4 & 2\pi \\ 3j & 1+\sqrt{3}j \end{bmatrix},$$
$$B = \begin{bmatrix} 3 & -2\pi \\ \pi & 6 \end{bmatrix},$$

   Using Octave, compute the following:

   a) $A + B$
   b) AB
   c) A^2
   d) A^T
   e) B^{-1}
   f) $B^T A^T$
   g) $A^2 + B^2 - AB$

2. Create and run the script:

```
x = -3;
if x > 0
   a = 9;
   elseif x < 0
      a = 10;
   elseif x == 0
      a = 11;
   else
      a = 13;
endif
```

What is the value of `a` after execution of the above code?

3. Create and run the script:

```
x = 0;
for i = 1:10
   x = x+1;
endfor
```

What is the value of `a` after execution of the above code?

4. Consider the following set of linear algebraic equations:

$$\begin{aligned} 5x + 9y &= -3, \\ x - 5y - 7z &= -36, \\ -2y + 4z &= 32. \end{aligned}$$

Determine the values of x, y and z so that the set of algebraic equation is satisfied. (*Hint:* Write the equation in matrix vector form.)

5. Generate a plot of

$$y(x) = e^{-0.5x} \sin \omega x,$$

where $\omega = 5$ rads/s, and $0 \leq x \leq 10$. Utilize the colon notation to generate the x vector in increments of 0.1.

6. Develop an Octave script to plot the function

$$y(x) = \frac{1}{\pi} + \frac{1}{2} \sin \omega x - \frac{2}{3\pi} \cos 2\omega x,$$

where ω is a variable input at the command prompt. Label the x-axis with `time(seconds)` and the y-axis with `y(x) = (1/pi) + (1/2) * sin(wx) - (2/3pi) * cos(2wx)`. Include a descriptive header in the script, verify that the help function will display the header. Plot the function for $\omega = 1, 3$ and 10 rads/s.

7. Consider the function

$$y(x) = 5 + 5e^{-x} \cos(\omega x + 0.5),$$

Develop a script to co-plot $y(x)$ for the three values of $\omega = 1, 3, 10$ rads/s with $0 \leq x \leq 5$ seconds. The final plot should have the following attributes:

Title	`y(x) = 5 + 5 exp(-x)*cos(wx + 0.5)`
x-axis label	`time(seconds)`
y-axis label	`y(x)`
Line type	$\omega = 1$: solid line
	$\omega = 3$: "+" line
	$\omega = 10$: "o" line
Grid	

Email your comments and suggestions to improve this material.
R. Stephen L. Ruiz rslruiz@gmail.com