A Tutorial Introduction to Octave

Center for Interdisciplinary Research and Consulting
Department of Mathematics and Statistics
University of Maryland, Baltimore County
www.umbc.edu/circ

Spring 2015
Mission and Goals: The Center for Interdisciplinary Research and Consulting (CIRC) is a consulting service on mathematics and statistics provided by the Department of Mathematics and Statistics at UMBC. Established in 2003, CIRC is dedicated to support interdisciplinary research for the UMBC campus community and the public at large. We provide a full range of consulting services from free initial consulting to long term support for research programs.

CIRC offers mathematical and statistical expertise in broad areas of applications, including biological sciences, engineering, and the social sciences. On the mathematics side, particular strengths include techniques of parallel computing and assistance with software packages such as MATLAB and COMSOL Multiphysics (formerly known as FEMLAB). On the statistics side, areas of particular strength include Toxicology, Industrial Hygiene, Bioequivalence, Biomechanical Engineering, Environmental Science, Finance, Information Theory, and packages such as SAS, SPSS, and S-Plus.

Copyright © 2003–2015 by the Center for Interdisciplinary Research and Consulting, Department of Mathematics and Statistics, University of Maryland, Baltimore County. All Rights Reserved.

This tutorial is provided as a service of CIRC to the community for personal uses only. Any use beyond this is only acceptable with prior permission from CIRC.

This document is under constant development and will periodically be updated. Standard disclaimers apply.

Acknowledgements: We gratefully acknowledge the efforts of the CIRC research assistants and students in Math/Stat 750 Introduction to Interdisciplinary Consulting in developing this tutorial series.

The source code for Octave is freely redistributable under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation, www.octave.org

1 Introduction

1.1 What is Octave?

GNU Octave is a free software package which utilizes a high-level language for numerical computations. Octave uses a command line interface for solving both linear and nonlinear problems. Octave's language is generally compatible with Matlab, and thus most Matlab code can be run in Octave with few or no changes. Octave was originally designed for an undergraduate-level textbook on chemical reactor design but was built to be a very flexible software package.

1.2 What is MATLAB?

MATLAB is a professional software package used for numerical calculations which is built on a foundation of efficient linear algebra operations. MATLAB is a contraction for "Matrix Laboratory", originally designed as a tool for the manipulation of matrices, and is now capable of performing a wide range of numerical computations. MATLAB also possesses extensive graphics capabilities.

1.3 Objectives of This Tutorial

- Entering commands in Octave
- Solving Linear Systems
- Computing Eigenvalues/Eigenvectors
- 2-D Plotting
- Further Help

The intention of this tutorial is in not to teach programming in Octave nor to serve as a replacement the product documentations that are available. Rather, it is designed to assist first-time users and to offer a glimpse of Octave's capabilities.

2 Starting and Exiting Octave

2.1 Starting Octave

At UMBC, Octave for Microsoft Windows is available in the Engineering 122 computer lab on campus. To start Octave, click on the Windows start orb to open the Start Menu, type `octave` to search for the program, then click on the version you want to run, `Octave (Experimental GUI)` for the experimental version of Octave's graphic user interface, or `Octave (Command Line)` for the command line prompt version of Octave.
Figure 1: The Octave Graphic User Interface window which appears when Octave is started.
2.2 Exiting Octave

Since Octave is a free software, it can easily be downloaded on Microsoft Windows or Linux machines. Visit www.octave.org and click on Download to learn the see the various ways in which to install Octave on your computer. Once installed, to start Octave in Microsoft Windows, click on the Windows Start Orb to open the Start Menu, type octave to search for the program, then click on the version you want to run, for example, Octave 3.8.1. For Linux users, type octave at the operating system prompt. The graphic user interface (GUI) that will appear when Octave is opened in Windows is shown in Figure 1.

Several windows in the GUI are currently visible. We will focus on the Command Window only. The Workspace window contains information about variables used in the current Octave session, the File Browser list all files in the directory in which Octave was opened and the Command History lists all recent commands issued in Octave.

The window that we will focus on is the command window; the last string object >> in the command window is the Octave prompt. This is where commands will be entered by the user. If at any point you enter an incorrect command and would like to enter the correct command, press the ↑ key, edit the incorrect command and then press return. Please note that in this tutorial, fonts of this type will denote both Octave commands and its resulting output.

2.2 Exiting Octave

To exit Octave simply type quit or exit at the Octave prompt >> in the command window or click on the × at the top right of the command window.

3 Solving Linear Systems

Originally, Octave was designed as a tool for chemical reactor design problems. Octave was then adapted for many other tasks, including to solve linear systems of equations. Consider the following system of equations:

\[-x_2 + x_3 = 3\]
\[x_1 - x_2 - x_3 = 0\]
\[-x_1 - x_3 = -3\]

This has the following solution: \(x = (x_1, x_2, x_3) = (1, -1, 2)\). This is equivalent to solving the following matrix equation

\[Ax = b\]

where \(A \in \mathbb{R}^{3 \times 3}\), \(x \in \mathbb{R}^{3 \times 1}\) and \(b \in \mathbb{R}^{3 \times 1}\). Solving this problem using Octave involves two steps:

- Enter the system matrix \(A\) and the right-hand side vector \(b\).
- Solve the system of equations.
3.1 Entering Vectors and Matrices into Octave

To enter the right-hand side vector \( b \) and then have Octave display it on the screen, simply type the following at the prompt and press return:

\[
\texttt{>> b = [3;0;-3]}
\]

This results in the output:

\[
\texttt{b = }
\begin{bmatrix}
3 \\
0 \\
-3
\end{bmatrix}
\]

Notice that the vector is in column form, because the brackets in the input line indicate that \( b \) is an array and the ‘;’ after each entry tells Octave to begin a new row. To obtain the same vector in row form, type the following at the prompt and press return:

\[
\texttt{>> c = [3 0 -3]}
\]

to obtain

\[
\texttt{c = }
\begin{bmatrix}
3 & 0 & -3
\end{bmatrix}
\]

You can confirm the difference between the two vectors by typing

\[
\texttt{>> size(b)}
\]

\[
\texttt{ans = }
\begin{bmatrix}
3 & 1
\end{bmatrix}
\]

and

\[
\texttt{>> size(c)}
\]

\[
\texttt{ans = }
\begin{bmatrix}
1 & 3
\end{bmatrix}
\]

The \texttt{size} command returns the dimensions of the array in row-column order. When the user does not specify a variable name for any output, Octave assigns the output to the temporary variable \texttt{ans}.

If you do not want Octave to output anything to the screen, simply include a ’;’ at the end of the command line before pressing return, as in
3.1 Entering Vectors and Matrices into Octave

\[
>> b = [3;0;-3];
\]

To verify that a variable has indeed been defined correctly, one may simply type the variable name (without a ;) at the prompt and press return:

\[
>> b
\]

\[
b =
\begin{bmatrix}
3 \\
0 \\
-3
\end{bmatrix}
\]

Before entering the system matrix \( A \) from our example, observe that a matrix is just an array of vectors. The first row of \( A \) is the vector of coefficients from the first equation, the second row of \( A \) from the second equation, etc. To enter the matrix \( A \), type the following at the prompt and press return:

\[
>> A = [0 -1 1; 1 -1 -1; -1 0 -1]
\]

to obtain

\[
A =
\begin{bmatrix}
0 & -1 & 1 \\
1 & -1 & -1 \\
-1 & 0 & -1
\end{bmatrix}
\]

The ; separates again the rows of the matrix.

To solve for \( x \) in \( Ax = b \), "left divide" by \( A \) as in \( A \backslash b \) and assign the result to \( x \) as in

\[
>> x = A \backslash b
\]

The result is:

\[
x =
\begin{bmatrix}
1 \\
-1 \\
2
\end{bmatrix}
\]

The \( \backslash \) operator performs an \( LU \)-decomposition of \( A \) to solve the system. You can check that the result is correct by computing

\[
>> b - A * x
\]
The result is:

\[
\text{ans} = \\
\begin{array}{c}
0 \\
0 \\
0 \\
\end{array}
\]

which shows that this small linear system was solved without error.

4 Computing Eigenvalues and Eigenvectors

Another very important function that Octave performs is the computation of eigenvalues and eigenvectors of a square matrix \( A \). This allows a matrix to be decomposed into the following:

\[
A = PDP^{-1}
\]

Here \( P \) is the matrix of eigenvectors and \( D \) is a diagonal matrix containing the eigenvalues of \( A \).

Let us compute the eigenvalues and eigenvectors of the following matrix:

\[
A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}.
\]

After entering the matrix into Octave, use Octave's `eig` function to compute the eigenvalues of the \( 2 \times 2 \) matrix \( A \):

\[
\begin{align*}
\text{>> } & A = \begin{bmatrix} 1 & -1; \ 1 & 1 \end{bmatrix}; \\
\text{>> } & \text{values} = \text{eig}(A)
\end{align*}
\]

This produces the vector `values` which contains the two (complex) eigenvalues of \( A \): `values` =

\[
\begin{array}{c}
1 + 1i \\
1 - 1i \\
\end{array}
\]

The fact that the components of `values` are complex numbers is indicated by the appearance of the imaginary unit \( i \). In conventional mathematical notation with the notation \( i = \sqrt{-1} \) for the imaginary unit, the first component of `values` reads \( 1 + i \) and the second one \( 1 - i \). This example demonstrates the important fact from linear algebra that even a real-valued matrix such as \( A \) can have complex eigenvalues. In turn, this example is designed to bring out that Octave uses complex arithmetic by default.

The `eig` function is an example of an Octave function that gives different return values, depending on how many output arguments are given. Above, `values` was the
only output argument. Specifying two output parameters to the \texttt{eig} routine produces a matrix containing the eigenvectors in addition to the eigenvalues. In keeping with the earlier notation this is accomplished by the following:

\begin{verbatim}
>> [P,D] = eig(A)
\end{verbatim}

This produces a matrix $P$ that contains the eigenvectors of $A$:

\[
P =
\begin{pmatrix}
0.70711 + 0.00000i & 0.70711 - 0.00000i \\
0.00000 - 0.70711i & 0.00000 + 0.70711i
\end{pmatrix}
\]

It also produces a diagonal matrix $D$ which contains the eigenvalues of $A$ along its diagonal.

\[
D =
\begin{pmatrix}
1 + 1i & 0 \\
0 & 1 - 1i
\end{pmatrix}
\]

To check our solution, we should be able to multiply the matrices generated from the output of the \texttt{eig} routine to obtain the original matrix $A$. Entering the following

\begin{verbatim}
>> A = P*D*inv(P)
\end{verbatim}

results in

\[
A =
\begin{pmatrix}
1 & -1 \\
1 & 1
\end{pmatrix}
\]

which is indeed the correct matrix. Observe that matrix multiplication (assuming all inner dimensions agree) is accomplished by the * command and the inverse of a matrix is obtained by the \texttt{inv} function. Then, multiplying $PP^{-1}$ (or $P^{-1}P$) should result in the identity matrix $I$:

\begin{verbatim}
>> P*inv(P)
\end{verbatim}

\[
\text{ans} =
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
or

```matlab
>> inv(P)*P
```

```matlab
ans =

1 0
0 1
```

Another approach to checking our solution is to examine each eigenvalue/eigenvector pair individually. To extract the eigenvectors, \( x_1 \) and \( x_2 \), (i.e. the columns of \( P \)) perform the following:

```matlab
>> x1 = P(:,1)
>> x2 = P(:,2)
```

This results in:

```matlab
x1 =

0.70711 + 0.00000i
0.00000 - 0.70711i
```

and

```matlab
x2 =

0.70711 - 0.00000i
0.00000 + 0.70711i
```

The colon `:` in the first argument of `A` indicates that we are extracting all rows, while the number in the second argument of `A` specifies which column to extract. To extract the eigenvalues, \( \lambda_1 \) and \( \lambda_2 \), (i.e. the diagonal entries of \( D \)) perform

```matlab
>> lambda1 = D(1,1);
>> lambda2 = D(2,2);
```

Here, we simply specify the component indices of each eigenvalue in the matrix \( D \). Recall that each eigenvalue/eigenvector pair must satisfy

\[
Ax_i = \lambda_i x_i \Leftrightarrow Ax_i - \lambda_i x_i = 0
\]

Let us verify the result by computing the following:

```matlab
>> A*x1 - lambda1*x1
>> A*x2 - lambda2*x2
```

This returns in
ans =

0
0

for both cases.

5 2-D Plotting with Octave

5.1 Plot Data Given in a File

This section examines Octave’s ability to plot 2-D curves using data points that are given in a data file with two columns listing the x- and y-values, respectively. Octave’s plot function can plot such curves once the x- and y-values are assembled in vector form.

We consider the case when one has been given data in the file ‘octavedata.dat’ and must generate a plot from the data. To download this file, visit www.umbc.edu/circ, click on Software Workshops and scroll down to the section on Octave. There, under “Datafile for Basic Octave Workshop,” you will see the file ‘octavedata.dat’ which you can download to your computer. Examining the first few lines of the data file shows that two columns of data are given:

-6.2831853e+00  -6.1470950e+00
-6.1850105e+00  -3.2604297e+00
-6.0868358e+00  3.6814515e+00
-5.9886510e+00  5.7807787e+00
-5.8904862e+00  8.2377784e-01
-5.7923115e+00  -4.8947573e+00

... ... ... ... ...

This data actually comes from the function \( f(x) = x \sin(x^2) \) on the interval \([-2\pi, 2\pi]\). The first column corresponds to the x-values and the second column to the y-values.

We must first transfer this data from the given file into Octave. This is accomplished using Octave’s load command. The load commands takes a file name as its argument and stores the data in a matrix. To accomplish this type

\[
>> A = \text{load ('octavedata.dat')};
\]

Now the data is stored in the two column matrix \( A \).

Octave’s plot command requires two equal-length vectors to produce a plot. Using the same extraction technique as before for the first and second column

\[
>> x = A(:,1);
\]
\[
>> y = A(:,2);
\]
we obtain the two vectors \( x \) and \( y \).

Before using Octave's `plot` command to plot the data, it is useful to verify that \( x \) and \( y \) are the same size (which, of course, is necessary to be able to produce a plot). Octave's `length` command,

\[
\begin{align*}
\gg & \text{ length}(x) \\
\gg & \text{ length}(y)
\end{align*}
\]

returns the following as length of each of the vectors:

\[
\text{ans} =
\]

\[
129
\]

Next, we invoke Octave's `plot` command.

\[
\gg \text{ plot}(x,y)
\]

which opens a figure window that contains the plot and that is shown in Figure 2.

![Figure 2: Plot of \( y = x \sin(x^2) \).](image)

We observe that the axes are not labeled, there are no grid lines, no title exists, as well as other desirable annotations are missing. These options will be discussed in the next section.

### 5.2 Plotting Data Computed in Octave

If we carefully inspect the plot in Figure 2, we notice that the peaks of the curve are rather coarse; we want to improve this by redrawing the plot with higher resolution. Notice from above that the vectors \( x \) and \( y \) had length 129, that is, a resolution of
5.2 Plotting Data Computed in Octave

129 data points is used. Therefore, this section shows how one can compute the data in Octave and control the resolution yourself instead of loading it from a file. Also, in this section we demonstrate the use of command-line programming when creating and annotating the plot.

To improve the resolution, we start by defining a vector \( x \) that will have 1025 equally spaced data points in the interval from \(-2\pi\) to \(2\pi\). Create this vector by

\[
\text{>> } x = [-2*\pi : 4*\pi/1024 : 2*\pi];
\]

Here, the use of the two colons \( : \) sets up a vector of numbers that range from \(-2\pi\) (the number before the first colon) to \(2\pi\) (the number after the second colon) with a step size of \(4\pi/1024\) (the number between the two colons). Since the stepsize is \(1/1024\)th of the interval width \(4\pi\), \( x \) will be 1025 elements long, which you can confirm by

\[
\text{>> } \text{length}(x)
\]

To call the \texttt{plot} function for the function \( f(x) = x \sin(x^2) \) at all values of \( x \) given in the components \( x(i) \) of the vector \( x \), we need to compute a vector \( y \) with the components \( y(i) \) such that \( y(i) = x(i) \cdot \sin(x(i)^2) \) for all indices \( i \). This is accomplished in one line by the command

\[
\text{>> } y = x \cdot \sin(x.^2);
\]

Some more explanations are useful here: Octave has all basic mathematical functions such as \( \sin \) predefined, and they work componentwise by default. The expression \( x.^2 \) inside the \( \sin \) argument computes the exponentiation of each component of \( x \). The point here is that we use the dot-operator \( .^\text{2} \), which computes the power of each component of \( x \) as \( x(i)^2 \). (If we tried to use \( ^\text{2} \) without the dot, we would get an error, because Octave interprets all operators as operators in the sense of matrix- or vector-algebra and the power of a vector is not defined mathematically.) The same logic applies to the dot-operator \( .\cdot \) that computes the product of each component \( x(i) \) of \( x \) with the corresponding component \( \sin(x(i)^2) \) of \( \sin(x.^2) \). You can try this out by manually evaluating, for instance, \( x.^\text{2} \) at the command-line.

With these new vectors \( x \) and \( y \) of length 1025, we can now obtain the higher-resolution plot by

\[
\text{>> } \text{plot}(x,y)
\]

To annotate the plot with a grid, a title, axis labels, and control the extent of the \( x \)-axis, enter at the command-line the commands

\[
\begin{align*}
\text{>> } & \text{axis on} \\
\text{>> } & \text{grid on} \\
\text{>> } & \text{title ('Graph of } f(x) = x \sin(x^2) \text{'}) \\
\text{>> } & \text{xlabel ('x')} \\
\text{>> } & \text{ylabel ('f(x)')} \\
\text{>> } & \text{xlim }([-2*\pi \ 2*\pi])
\end{align*}
\]
Note that we do not specify any limits for the y-axis because Octave will pick those limits automatically to fit the data reasonably in the plot. The final plot with annotation should look like Figure 3.

![Plot](image)

Figure 3: Plot of $f(x) = x \sin(x^2)$ with 1025 data points.

6 A Basic Example of Octave Programming

We will now introduce the use of a so-called script file. This type of file is the most basic type of Octave program and is nothing but a sequence of commands. When the script file is executed by calling its name from the command-line, one command is executed after the other, just as if you had cut-and-pasted them into the command-line. Script files are very useful so that a complete documentation exists how the result was obtained. For instance, let's collect all commands that created the plot in Figure 3 in a script file. Printing out this script file would then give a complete documentation of how the plot was created.

Script files in Octave must have the filename extension .m. They can have just about any name, but it is best to pick a meaningful name that gives an idea about its purpose and that is unique to avoid confusion with existing functions in Octave. In this spirit, let's call the script file to create Figure 3 *plotxsinxx.m*. To create the file by this name and start editing it with the Octave editor, click on File, New and then Script. The editor window of Octave will open. You must be in a directory folder, for which you have write permission to be able to save your file to disk later. You may need to change directory accordingly under the File -> Save As menu.

Type in the commands that will create the desired plot with all annotations

```octave
x = [-2*pi : 4*pi/1024 : 2*pi];
y = x .* sin(x.^2);
```
plot (x,y);
axis on
grid on
title ('Graph of f(x)=x sin(x^2)')
xlabel ('x')
ylabel ('f(x)')
xlim([-2*pi 2*pi])

After you save the file, by selecting File -> Save within the Editor's menu, you simply execute it at the command-line by calling its name

>> plotxsinxx

and a plot complete with all annotations will come up.

To see why it is useful to use script files to create plots, let's say that you wish to further increase the resolution of the plot from 1024 to 2048 points. Now that you have the script file, this is simply accomplished by changing the 4*pi/1024 to 4*pi/2048 in the first line, and the rerunning the script file by saying plotxsinxx at the command-line. Contrast this to the earlier approach, in which you would have had to re-enter or re-call each individual plot and annotation command.

Script files are one type of so-called m-files in Octave. The other type is the function file, and there exist if-statements, for- and while-loops, and other control structures. Introducing these is beyond the scope of this tutorial introduction.

7 Using Octave's Help Feature

Octave provides the user with extensive online help, and links to the various forms of support can be found by visiting www.octave.org and clicking on Support. This link includes a reference manual as well as access to Octave's Wiki for support.

Octave also has built in help and documentation features which can be accessed through the command window. To access basic documentation for a particular Octave function, for instance for the plot function, you would say

>> help plot

within the command window. To access the detailed documentation, you would say

>> doc plot

at the command-line, which will load the appropriate page in the Documentation. You can also click on the Documentation tab at the bottom of the GUI and search for the documentation for a specific function. For many basic functions, the information provided in both ways is identical. For more sophisticated functions however, the additional examples provided in the Help Browser are often helpful. This is particularly true for graphics functions such as plot above.