GNU Octave/Matlab Primer

Sachin Shanbhag*

August 27, 2007

1 Introduction

The objective of this tutorial is to get you up and running with MATLAB or GNU Octave.[1] Once you've figured out how to fire MATLAB up, you can try simple things at the command prompt such as:

```
a = 2;
b = 3;
a + b
```

Try to fool around with the semicolons, and with other operations such as `a^b` etc, and convince yourself that MATLAB is *at least* as good as your calculator. MATLAB likes arrays (vectors) and array operations - something that we will exploit time and again. Try the following and see if it makes any sense to you. If it does not, try again, or try harder!

```
v=[0 1 2]
u=[3; 4; 5]
w=[1 2 3; 4 5 6; 7 8 9]
transpose(u)
inverse(w)
v + u'
u(2)
```

*This is loosely based an excellent tutorial by Dr. Robert Lionberger.

[1]GNU Octave is a MATLAB-like program that is (a) platform independent, and (b) free, which is why I like it.

```
v(1)
w(:,2)
w(2,:)
x=[5 6 7]
x * v  % invalid, why?
x .* v % valid. what does this operation do? This is IMPORTANT!
```

You can always get help on a command (say `inverse`) by typing `help inverse`. And of course, there's also Google. The percent sign denotes the start of a comment, and MATLAB ignores it.

So far we've been talking about using MATLAB in an interactive mode. But it is far more powerful, and liberating to use it in a "batch" mode, by writing function and script files.

2 Function and Script Files

MATLAB files end in `.m` and there are two types of files we will use, function files and script files. Function files provide definitions of new functions while script files save a list of commands that can be executed again.

In order for MATLAB to use your functions and run your scripts it must find them and you must tell MATLAB where to find them. You add directories to the search path MATLAB uses, via a pull-down menu in the MATLAB window, or the command `editpath`. Every time you use a new personal computer you will have to use the `editpath` command. Under Unix/Linux your preferences are shared across different machines so you only have to do this once.

2.1 Function Files

We will write a function that uses the ideal gas law $PV = RT$ to calculate the pressure given T, V, R. Function files have the format

```
function P = Pressure(T,V,R)
% Pressure returns ideal gas pressure
P = R.*T./V;  % Why am I using .* and ./?
end
```

and are named after the function they define, for example this is the file `Pressure.m`. The operators ".*" and "./" tell MATLAB to do element by element multiplication if `T` and `V` are arrays of values.[2] You can use this function from the command window as

```
Pressure(298,0.022,8.314)
ans =
1.1262e+05
```

When you call the function `Pressure` MATLAB looks for the file `Pressure.m` in the current directory and then in the MATLAB search path. Over the course of the class you will build up a collection of functions. I have posted some MATLAB functions/scripts that you will use and modify at:

`https://people.scs.fsu.edu/sachins/thermo_resources.html`

If you want to change a function or a script you must tell MATLAB to use the new version by entering the command `clear Pressure` from the command line to remove the old version from memory. The command `clear all` removes all definitions so can start with a blank slate of MATLAB.

2.2 Script Files

A script file is just a list of commands so you dont have to retype them. Here is an example file `IdealGasScript.m`

```
% script M-file IdealGasScript.m
%
R = 8.314; % Gas Constant
T = 298;
V = 0.0022;
Pressure(T,V,R)
```

You can run this file from the `Run Script` menu item or by typing the filename without the .m on the MATLAB command line. (The script file must be found in your MATLAB Path). You should save the code that generates your homework solutions as this type of file. If script files are not doing what you expect the command `clear all` will remove all old definitions from memory. This is often necessary when you are changing the script file.

[2]Remember what I said about exploiting vectors/arrays?

3 Plotting

3.1 Plotting using Arrays

The best way to plot is to calculate the data points to be plotted and store them in an array and then plot the array. An example is in the following script file:

```
R = 8.314;
T1 = 298;
T2 = 398;
T3 = 498;
pts = 30;

%Create an array of Volume values
% from 0.01 to 0.1
V = linspace( 0.01, 0.1, pts);

% Fill the data arrays
P1 = Pressure(T1,V,R);
P2 = Pressure(T2,V,R);
P3 = Pressure(T3,V,R);

% Plot the data points
plot(V,P1,V,P2,V,P3)
title('Ideal Gas Plot')
xlabel('Molar Volume')
ylabel('Pressure')
legend('T=298','T=398','T=498',0)
```

3.2 Plot Style

To add different color lines or symbols to your plots add secret MATLAB codes to the previous example

```
% Plot the data points
plot(V,P1,'gp-',V,P2,'rx:',V,P3,'bs')
title('Ideal Gas Plot')
xlabel('Molar Volume')
```

```
ylabel('Pressure')
legend('T=298','T=398','T=498',0)
```

The first letter in the code sets the color of the line.

Symbol	Color	Symbol	Color
g	green	b	blue
r	red	c	cyan
m	magenta	y	yellow
k	black	w	white

The second symbol in the code sets the marker for the line.

Symbol	Marker Style	Symbol	Marker Style
.	point	v	triangle (down)
x	cross	^	triangle (up)
+	plus	<	triangle (left)
s	square	>	triangle (right)
d	diamond	*	asterisk
p	pentagram	h	hexagram

The third symbol sets the line style,if it is missing a solid line will be used, if it is a space then just points will be plotted with no line connecting them.

Symbol	Line Style
-	solid line
--	dashed line
:	dotted line
-.	dash-dot line
no symbol	no line

On the homeworks plot the predictions of equations as lines connecting the symbols, if you are comparing the equations to experimental data then plot the data as symbols with no lines.

4 Root Finding

4.1 Polynomial equations

To find the roots of a polynomial equation of the type $a_n x^n + a_{n-1} x^{n-1} ... a_1 x + a_0$, we can use the `roots` command. In thermodynamics, we are often required to find the roots of a cubic polynominal equation. This is simple to do in MATLAB. To define the polynomial $x^4 - 12x^3 + 25x + 116$, and find its roots use,

```
pol=[1 -12 0 25 116]
roots(p)
```

For non-polynomial equations, we can also use `fzero`.

4.2 fzero with real functions

To illustrate, lets find the molar volume which gives P = 10^6MPa at T = 298K. This is easy to do analytically, $V = RT/P = 0.00247$. To solve this problem, you first define a regular function in a .m file whose *first argument* is the one that you want to adjust to make the value of the function equal zero. For this problem that function would be,

```
function Z = fV(V,T,R,P)
fV = R.*T./V - P;
```

The first argument to fzero is the function while the second is an initial guess, a bad guess can cause a bad answer so the closer you can guess the better.

```
% For MATLAB version 5.3 and higher
T = 298;
R = 8.314;
P = 10^6;
W = fzero('fV', 0.01,[],T,R,P)
```

The empty arrays tell MATLAB to use its default values for the other parameters. These parameters control the accuracy of the result. For every problem we do in this class the default values are fine. You pass the additional parameters to function by sticking them at the end of the call to `fzero`.

5 Loops and Conditions

MATLAB bestows you with a wide range of “control” tools with which you can tame any beast. Among these, `for` and `if` structures are somewhat important. If you can use these, you will have no trouble learning and using other structures such as `while`.

```
for i=1:5 % try for i=1:2:10, and see what changes
  x(i)=i
  v(i)=i^2
end

plot(x,v)
```

As some of you might have recognized, you can get the same effect by the following:

```
x=[1:1:5];
v=x.^2; % notice our friend ".^", what if you used only ^?
plot(x,v)
```

You can use the `if` to control flow. Consider the following script.

```
x=3

if(x==3) % note the == to test equality
  disp("three") % disp prints to screen, look up printf as well
elseif(x==4)
  disp("four")
else
  disp("neither three nor four")
end
```

6 Reading Data from a File

Say you have a file of columns of data, that are generated by another program. Consider a file `VaporT=500.txt` which contains one column of pressure and one column volume. The commands first read in a matrix of data and then breaks it into arrays

```
rawdata = load('VaporT=500.txt');
Pdata = rawdata(:,1)
Vdata = rawdata(:,2)
```

The file you are reading from can be in the same directory where you store your script files.

7 Log and Semilog Plotting in MATLAB

To get log or semilog plots you can replace the plot command with

```
plot(V,P1,'gp-',V,P2,'rx:',V,P3,'bs')
semilogx(V,P1,'gp-',V,P2,'rx:',V,P3,'bs')
semilogy(V,P1,'gp-',V,P2,'rx:',V,P3,'bs')
loglog(V,P1,'gp-',V,P2,'rx:',V,P3,'bs')
```

To generate values that are log spaced replace linspace with logspace

```
V = logspace( -2, 1, pts); % from 10^-2 to 10^1
V = linspace( 0.01, 10, pts);
```

Here is an example semilog plot of the ideal gas law found in the file `LogPlotScript.m`

```
R = 8.314;
T1 = 298;T2 = 398;T3 = 498;
pts = 30;

%Create an array of Volume values
% from 0.01 to 0.1
V = logspace( -2, 1, pts);

% Fill the data arrays
P1 = R.*T1./V;
P2 = R.*T2./V;
P3 = R.*T3./V;

% Plot the data points
semilogx(V,P1,'gp-',V,P2,'rx:',V,P3,'bs')
```

```
axis([10^-2 10 0 4.0e5]) % manually control the limits to be plotted
title('Ideal Gas Plot')
xlabel(Molar Volume')
ylabel('Pressure')
legend('T=298','T=398','T=498', 0)
```