Using Matlab in the Chemical Engineering Curriculum at the University of Tennessee

A Workshop presented by
David Keffer
August 17, 2001

Abstract

This talk is intended to provide someone unfamiliar with Matlab the basic information to use Matlab as a platform for structured programming. The philosophy behind the presentation is that providing demonstrations of the syntax is the best way to learn the syntax. This philosophy applies to both the syntax used in writing code in Matlab, as well as the syntax used in locating information on how to write code in Matlab via the help files.

Finally, we provide a reminder of "A Web Resource for the Development of Modern Engineering Problem-Solving Skills", which contains graphical user interfaces to perform six basic but computationally nontrivial tasks (such as the solution of a system of nonlinear algebraic equations, or the solution of a system of nonlinear ordinary differential equations, or multivariate nonlinear optimization) which should prove very useful in many undergraduate courses.

Table of Contents

I. Basics

I.A. The command window

When you start up Matlab you see a white screen with a prompt that looks like “>>”. This is the command line prompt. Simple commands can be executed from the command line prompt. For example:

```
 » 2+2

ans =
     4
```

The command window returns the answer. Another example:

```
» sin(pi)

ans =
  1.2246e-016
```

The output can be stored in a variable. For example:

```
» a = sin(pi)

a =
  1.2246e-016
```

If you are not interested in immediately viewing the result of the calculation, end the statement with a semicolon. For example,

```
» a = sin(pi);
```

If you wish to view the contents of the variable a later, you can do so by typing the variable name at the command line prompt. For example

```
» a

a =
  1.2246e-016
```

I.B. Navigation

Generally, Matlab opens in a default directory. Presumably you wish to generate and save data in a directory which you own and control.

To change directories, use the cd (change directory) command. For example:

```
» cd z:\faculty\keffer
```

To determine which directory you are currently in, use the pwd (print working directory) command. For example:

```
» pwd
```

To view a listing of the contents of the directory, use the ls (list) command. For example

```
» ls
```

The default Matlab functions, such as plot and sign, can be used from any directory. If you create your own Matlab program, the best way to avoid problems is to be in the directory where you have stored the program before you attempt to run it.

I.C. Matlab intrinsic functions and help files

To view a list of the intrinsic functions available in MATLAB, type help at the command line prompt.

This shows a list of categories, such as

```
matlab\matfun   -  Matrix functions - numerical linear algebra.
```

If you want to view a list of intrinsic matrix functions, type

```
» help matfun
```

This returns a list of all matrix functions available, for example:

```
   eig         - Eigenvalues and eigenvectors.
```

If you want to view the help file on a particular intrinsic function, type help followed by the function name. For example

```
» help eig
```

EIG Eigenvalues and eigenvectors.
E = EIG(X) is a vector containing the eigenvalues of a square matrix X.

[V,D] = EIG(X) produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that X*V = V*D.

Example: Let's calculate the eigenvalues of the matrix $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 3 & 4 \end{bmatrix}$. At the command line prompt we first define the matrix, called a, then we find the eigenvalues.

```
» a = [1 1 1
1 2 1
2 3 4];
» eig(a)

ans =
    0.3542
    5.6458
    1.0000
```

(Here we have also shown how to input a matrix at the command line prompt.)

If we wanted both the eigenvalues and eigenvectors, then following the syntax given in the help file, we type:

```
» [eigvec,eigval] = eig(a)

eigvec =

    0.8736   -0.2650    0.0000
   -0.4792   -0.3220   -0.7071
   -0.0849   -0.9089    0.7071


eigval =

    0.3542         0         0
         0    5.6458         0
         0         0    1.0000
```

We did not have to type enter the matrix in again, because it was already stored in the variable a.

If we want to see more significant digits, we can use the format command. For help on using the format command, type help format at the command line prompt. The following command shows all sixteen digits.

```
» format long
» [eigvec,eigval] = eig(a)

eigvec =

   0.87357546912586  -0.26495137869971   0.00000000000000
  -0.47922932454258  -0.32198227875427  -0.70710678118655
  -0.08488317995931  -0.90891593620826   0.70710678118655


eigval =

   0.35424868893541                  0                  0
                  0   5.64575131106459                  0
                  0                  0   1.00000000000000
```

In general, the information needed to run any intrinsic function is obtained by typing help *command_name* at the command line prompt.

Finding the name of the function is accomplished by typing help to obtain a list of categories, then typing help *category_name*.

Although, Matlab offers other methods of help (i.e. their website), they are generally not helpful.

A list of the function categories is reproduced below:

```
HELP topics:

matlab\general      -  General purpose commands.
matlab\ops          -  Operators and special characters.
matlab\lang         -  Programming language constructs.
matlab\elmat        -  Elementary matrices and matrix manipulation.
matlab\elfun        -  Elementary math functions.
matlab\specfun      -  Specialized math functions.
matlab\matfun       -  Matrix functions - numerical linear algebra.
matlab\datafun      -  Data analysis and Fourier transforms.
matlab\polyfun      -  Interpolation and polynomials.
matlab\funfun       -  Function functions and ODE solvers.
matlab\sparfun      -  Sparse matrices.
matlab\graph2d      -  Two dimensional graphs.
matlab\graph3d      -  Three dimensional graphs.
matlab\specgraph    -  Specialized graphs.
matlab\graphics     -  Handle Graphics.
matlab\uitools      -  Graphical user interface tools.
matlab\strfun       -  Character strings.
matlab\iofun        -  File input/output.
matlab\timefun      -  Time and dates.
matlab\datatypes    -  Data types and structures.
matlab\dde          -  Dynamic data exchange (DDE).
matlab\demos        -  Examples and demonstrations.
toolbox\pde         -  Partial Differential Equation Toolbox.
toolbox\signal      -  Signal Processing Toolbox.
toolbox\optim       -  Optimization Toolbox.
toolbox\control     -  Control System Toolbox.
control\obsolete    -  (No table of contents file)
stateflow\stateflow -  Stateflow
stateflow\sfdemos   -  (No table of contents file)
simulink\simulink   -  Simulink
simulink\blocks     -  Simulink block library.
simulink\simdemos   -  Simulink demonstrations and samples.
simulink\dee        -  Differential Equation Editor
toolbox\local       -  Preferences.
```

II. Structured Programming In Matlab

II.A. M-files

We don’t want to reinvent the wheel every time, so we don’t want to use the command window for complicated tasks. Doing so would require us to retype every line each time we restarted Matlab. Therefore, we save our programs in source files, called M-files in Matlab. An M-file contains the source code of a program, just as a FORTRAN *.f or C *.c file contains the same.

There are two types of M-files: scripts and functions.

If you don’t like the fact that there can be both scripts and functions, then consider that scripts are a simple subset of functions and only use functions.

II.A.1. Scripts

Scripts are just a listing of commands, which could have been entered one after another at the command line prompt. Let’s create a script file.

We open a new M-file by going to File → New → M-file in the Matlab command window menu. We now have a new blank window. In this window, we type, for example:

```
a = [1 1 1
1 2 1
2 3 4];
[eigvec,eigval] = eig(a);
fid = 1;
fprintf(fid,'eigenvalue 1 = %e \n', eigval(1,1));
fprintf(fid,'eigenvalue 2 = %e \n', eigval(2,2));
fprintf(fid,'eigenvalue 3 = %e \n', eigval(3,3));
```

We save this file in a directory of our choice and call it demo_001.m. The *.m nomenclature designates the file as an M-file. Before we run this M-file, we make sure we are in the directory to which we saved the file. We run the code by typing the file name, less the .m extension at the command line prompt. For example:

```
» demo_001
eigenvalue 1 = 3.542487e-001
eigenvalue 2 = 5.645751e+000
eigenvalue 3 = 1.000000e+000
```

Comments: because we put semicolons after the definition of the matrix a and the calculation of the eigenvalues, they are not printed out. The three print statements give a formatted output. The variable fid is the file id number. We set it to “1”, which mean print to the screen. We then put the string we want it to print, contained within single quotes. The “%e” and “\n” are C string formatting properties, that tell the string to accept a real number in exponential form and then line return. Type help fprintf for more information on formatted printing. Finally, we tell it to print the eigenvalues, which are the diagonal elements of the matrix called eigval.

II.A.2. Functions

Functions are generally the same as Scripts but can have input and output arguments. The first line of a function must be of the form:

function [y_out1, y_out2, y_out3] = function_name(x_in1, x_in2, x_in3)

where y_outi is a scalar or vector of outputs, x_ini is a scalar or vector of inputs, and function_name is the name of the M-file, such as demo_001.

An example function follows. This function calculates the factorial of x. (There is no intrinsic Matlab factorial function.)

```
function [f] = factorial(n)
f = 1;
for i = 1:1:n
   f = f*i;
end
```

This function takes one input, n, and returns the factorial of n, n!, in the variable f. To execute this code from a command line prompt, we type:

```
» y = factorial(3)

y =
      6
```

The function has five lines. The first line defines the inputs, outputs, and name of the function. The second line initializes the values of the output. The next three lines compute the factorial. We will discuss the syntax of the for loop in a following section.

This function is simple but flawed, because the factorial is only defined for non-negative integers with the special definition that 0! = 1. We can write a more sophisticated function, which takes these limitations into account:

```
function [f] = factorial_v2(n)
if (n < 0)
   fprintf(1,'You supplied a value of n = %f \n', n);
   fprintf(1,'The factorial is undefined for negative numbers. \n');
   error('The code is aborting.');
elseif (abs(n-floor(n)) > 1.0e-14 )
   fprintf(1,'You supplied a value of n = %f \n', n);
   fprintf(1,'The factorial is undefined for non-integers. \n');
   error('The code is aborting.');
else
   f = 1;
   for i = 1:1:n
      f = f*i;
   end
end
```

This version of the code, stored as factorial_v2.m informs the user that the factorial cannot be computed for negative numbers or for non-integers. Here are four examples of executing it from the command line prompt.

```
» y = factorial_v2(6)

y =
   720

» y = factorial_v2(0)

y =
     1

» y = factorial_v2(-6)
You supplied a value of n = -6.000000
The factorial is undefined for negative numbers.
??? Error using ==> factorial_v2
The code is aborting.

» y = factorial_v2(6.4)
You supplied a value of n = 6.400000
The factorial is undefined for non-integers.
??? Error using ==> factorial_v2
The code is aborting.
```

Functions can be called by other functions. Consider the binomial probability distribution, b(x;n,p), which gives the probability of having x successes in n trials, given that the probability of a successful trial is p. The binomial probability distribution is defined as:

$$b(x;n,p) = \left(\frac{n!}{x!(n-x)!}\right)p^{x}(1-p)^{n-x}$$

We can create a function, called binomial.m, that calls factorial_v2.m to compute the necessary factorial. The basic code follows:

```
function f = binomial(x,n,p)
fact_n = factorial(n);
fact_x = factorial(x);
fact_nx = factorial(n-x);
f = fact_n/(fact_x*fact_nx)*p^x*(1-p)^(n-x);
```

So, for example, if we want to know the probability of finding 5 success in 8 trials, given that the probability of an individual success is 0.5, we would type at the command line prompt:

```
» prob = binomial(5,8,0.5)

prob =
   0.21875000000000
```

II.B. Syntax

From the examples above, you can see that Matlab can be used as a structured programming language, just as FORTRAN and C can.

The advantages of Matlab are

(1) every machine in Dougherty has Matlab.
(2) Matlab has easier syntax to master.

The primary disadvantage of Matlab is

(1) Matlab is dog-slow compared to Fortran. (My estimate is that it is approximately 300 times slower than the Fortran on a workstation.) If the computation takes 300 milliseconds rather than 1 millisecond, who cares. However, if the computation takes 300 minutes rather than 1 minute, that really stinks. In short, use Matlab only for simple problems that you want quick solutions to.

The above functions have examples of if-blocks and for-loops.

If you type help ops, you will see that the operators to be used in if-blocks are chosen from

```
 Relational operators.
   eq        - Equal                     ==
   ne        - Not equal                 ~=
   lt        - Less than                 <
   gt        - Greater than              >
   le        - Less than or equal        <=
   ge        - Greater than or equal     >=

 Logical operators.
   and       - Logical AND               &
   or        - Logical OR                |
   not       - Logical NOT               ~
```

If you type help lang, you will see the various constructs available in Matlab

```
   if          - Conditionally execute statements.
   else        - IF statement condition.
   elseif      - IF statement condition.
   end         - Terminate scope of FOR, WHILE, SWITCH and IF statements.
   for         - Repeat statements a specific number of times.
   while       - Repeat statements an indefinite number of times.
   break       - Terminate execution of WHILE or FOR loop.
   switch      - Switch among several cases based on expression.
   case        - SWITCH statement case.
   otherwise   - Default SWITCH statement case.
   return      - Return to invoking function.
```

With the information provided in this section, if you know Fortran-77, you can basically use the syntax of Fortran 77 in M-files, without knowing anything else about Matlab.

The main differences are

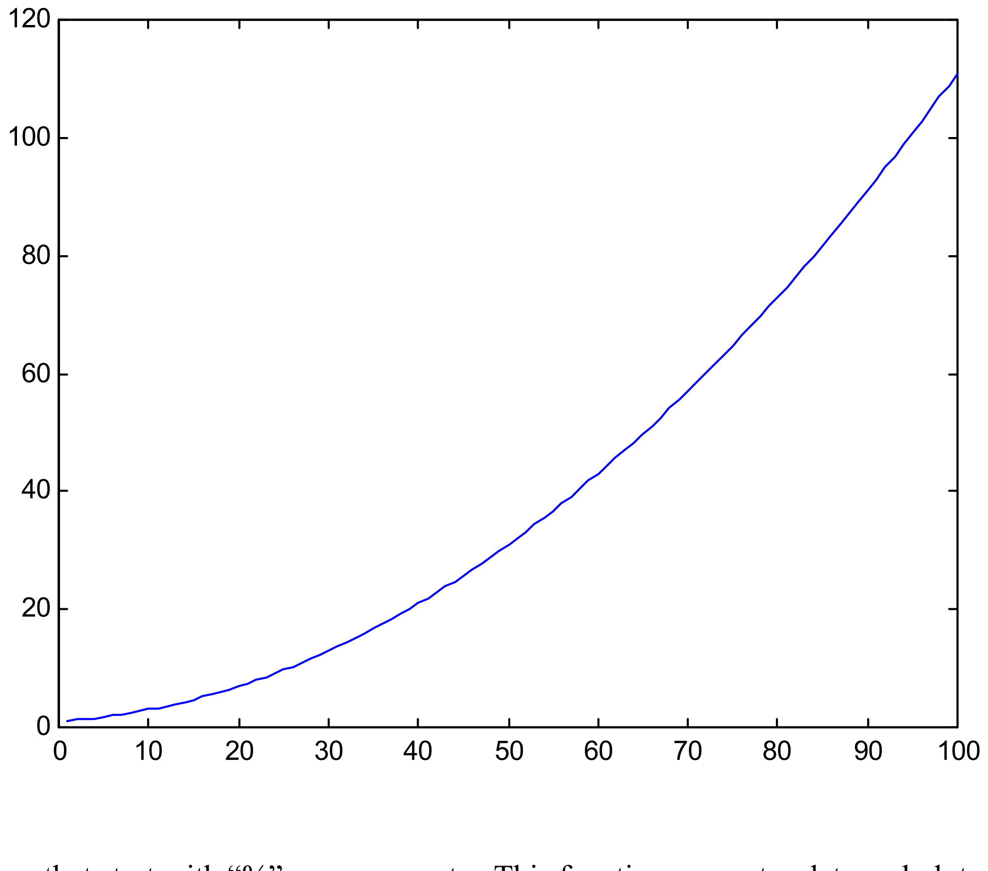
1) you don't have to define or dimension variables in Matlab
2) do-loops become for-loops
3) .eq. becomes ==, .lt. becomes <, .and. becomes &, etc
4) end each line in a semicolon, so they don't all print out

Matlab does provide an environment for structured programming.

III. Plotting

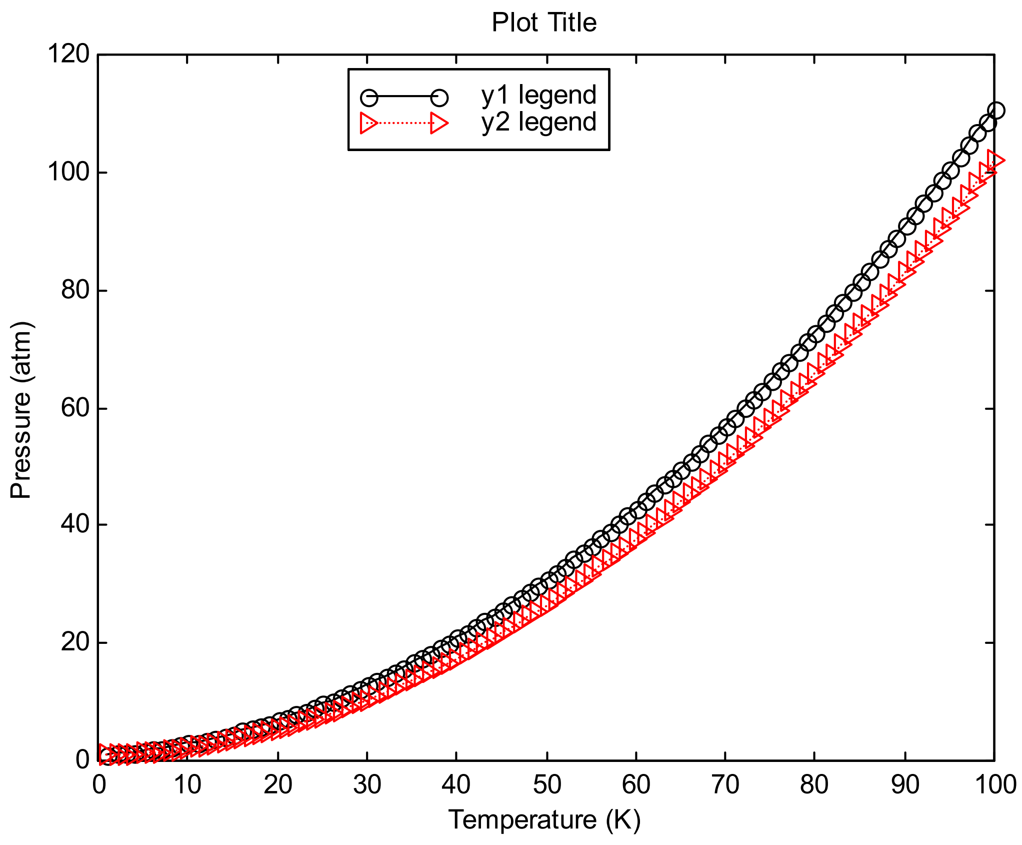
Let's consider the following M-file, plot_demo.m

```
function plot_demo
%
% generate some data to plot
%
n = 100;
a = 1;
b = 0.1;
c = 0.01;
for i = 1:1:n
   x(i) = i;
   y(i) = a + b*x(i) + c*x(i)^2;
end
%
% plot
%
figure(1)
plot(x,y);
```



The lines that start with "%" are comments. This function generates data and plots it. If you want more help on plotting options, type help plot.
Here is a similar function with a more complicated plotting option.

```
function plot_demo2
%
% generate some data to plot
%
n = 100;
a = 1;
b = 0.1;
c = 0.01;
for i = 1:1:n
   x(i) = i;
   y1(i) = a + b*x(i) + c*x(i)^2;
   y2(i) = a + c*x(i) + c*x(i)^2;
end
%
% plot
%
figure(1)
plot(x,y1,'k-o');
hold on;
plot(x,y2,'r:>');
xlabel('Temperature (K)');
ylabel('Pressure (atm)');
title('Plot Title');
legend('y1 legend','y2 legend');
hold off;
```



IV. Graphical User Interfaces

You can make graphical user interfaces (GUIs) in Matlab to make the use of complicated tools much simpler.

As I have mentioned before, I have identified certain tasks which I feel are most important for an undergraduate chemical engineer to solve:

1. Solving systems of linear algebraic equations
2. Solving single nonlinear algebraic equations
3. Solving systems of nonlinear algebraic equations
4. Solving single/systems of linear/nonlinear ordinary differential equations
5. Numerical Integration
6. Multivariate Nonlinear Optimization

I have created GUIs for tasks 2-6. Although, Matlab has intrinsic functions for some of these functions, I tried to illustrate in an earlier talk, "A Web Resource for the Development of Modern Engineering Problem-Solving Skills" presented to the Department on February 20, 2001, why these GUIs are necessary. If you want copy of those slides, I can give you an electronic copy.

The GUIs are located at:
A Web Resource for the Development of Modern Engineering Problem-Solving Skills
http://clausius.engr.utk.edu/webresource/index.html

Since February, the only new GUI is for Multivariate Nonlinear Optimization. I recognized that students do not have a tool to optimize multivariate nonlinear algebraic equations. For example, consider the Antoine Equation, relating vapor pressure to temperature.

$$P^{vap} = \exp\left(A - \frac{B}{T + C}\right)$$

If we give students vapor pressure as a function of temperature data, they do not know how to perform a nonlinear least squares regression to obtain the best fit coefficients A, B, and C. This is a basic but computationally non-trivial task that they ought to know how to do.

A demonstration on the use of the multivariate nonlinear optimizer GUI, nonlinopt_gui.m, follows.