Introduction to MATLAB for CSE390

Professor Vijay Kumar

Praveen Srinivasan

University of Pennsylvania

Introduction

MATLAB is an interactive program designed for scientific computations, visualization and programming. It stands for MATrix LABoratory. You can think of it as a scientific calculator with an environment for writing and debugging programs for scientific and engineering applications. MATLAB is installed on all PCs in the CETS labs and should be accessible to all engineering students. You can also buy a student edition for your own personal use at the bookstore. MATLAB has extensive on-line help facilities. This document provides some basic information designed to get you started.

Getting Started

When you run MATLAB, you will see a command window with the MATLAB prompt ">>". Typing "demo" at the MATLAB prompt allows the user to go through the demonstrations which show the capabilities of the program.

The preferred way to do computations in MATLAB is to generate a *script file* that contains all the commands you want MATLAB to execute. But before we do that, here are a few sample computations, which should indicate some of the powerful tools available to you.

At a very basic level, MATLAB can be used as a scientific calculator. To define a symbol *x* with the value 0.7 type x=0.7 at the MATLAB prompt. MATLAB returns:

```
>> x=0.7
x =
    0.7000
```

The following MATLAB commands (and output) illustrates how you might define symbols *x* and *y* with values and calculate the value of a third symbol $z = \exp(x) \cdot \sin y$. Note that pi (π) is a pre-defined constant in MATLAB.

```
>> x=0.7

x =
   0.7000

>> y=0.1*pi

y =
    0.3142

>> z=exp(x)*sin(y)
```

```
z =
    0.6223
```

Suppose you enter a symbol without giving it a value (try typing the symbol *a* at the prompt), you will get an error message:

```
??? Undefined function or variable 'a'.
```

Now we will define a square matrix *A* and a vector *x* and perform the calculation $y = A\,x$ and $y = x^{T}A$. All text after the % symbol is treated as a comment.

```
>> A=[1, 3; -1, 4]  % define a 2x2 matrix
A =
     1     3
    -1     4
```

Note the use of "," and ";" to separate entries of the matrix. “;” signifies the end of a row and the beginning of a new row. “,” allows you to enter a new column entry without terminating a row.

```
>> x=[2; -1]  % define a column vector
x =
     2
    -1

>> x'  % transpose of x
ans =
     2    -1

>> A*x

ans =
    -1
    -6

>> x'*A
ans =
     3     2
```

Try the following and you will get an error message because of the incompatibility between the matrix *A* and the vector *z*.

```
>> z=[2; 1; 1]
z =
     2
     1
     1
>> A*z
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Getting Help

MATLAB has a library of built-in functions. This library is displayed by simply typing "help" at the MATLAB prompt. MATLAB responds with:

```
matlab/general    - General purpose commands.
matlab/ops        - Operators and special characters.
matlab/lang       -  Programming language constructs.
matlab/elmat      -  Elementary matrices.
matlab/elfun      -  Elementary math functions.
matlab/specfun    -  Specialized math functions.
matlab/matfun     -  Matrix functions - linear algebra.
matlab/datafun    -  Data analysis, Fourier transforms.
matlab/polyfun    -  Interpolation and polynomials.
matlab/funfun     -  Function functions and ODE
solvers.
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 and output.
matlab/audiovideo -  Audio and Video support.
matlab/timefun    -  Time and dates.
matlab/datatypes  -  Data types and structures.
matlab/verctrl    -  Version control.
matlab/helptools  -  Help commands.
matlab/demos      -  Examples and demonstrations.
matlab/timeseries -  Time series data visualization
```

For example, the elementary math functions can be seen by typing:

```
>> help elfun
```

upon which MATLAB responds with:

```
Elementary math functions.

Trigonometric.
    sin         - Sine.
    sind        - Sine of argument in degrees.
    sinh        - Hyperbolic sine.
    asin        - Inverse sine.
    asind       - Inverse sine, result in degrees.
    asinh       - Inverse hyperbolic sine.
    cos         - Cosine.
    cosd        - Cosine of argument in degrees.
    cosh        - Hyperbolic cosine.
    acos        - Inverse cosine.
    acosd       - Inverse cosine, result in degrees.
```

```
    acosh       - Inverse hyperbolic cosine.
    tan         - Tangent.
    tand        - Tangent of argument in degrees.
    tanh        - Hyperbolic tangent.
    atan        - Inverse tangent.
    atand       - Inverse tangent, result in degrees.
    atan2       - Four quadrant inverse tangent.
    atanh       - Inverse hyperbolic tangent.
…

Exponential.
    exp         - Exponential.
    expm1       - Compute exp(x)-1 accurately.
    log         - Natural logarithm.
    log1p       - Compute log(1+x) accurately.
    log10       - Common (base 10) logarithm.
…

  Complex.
    abs         - Absolute value.
    angle       - Phase angle.
    complex     - Construct complex data
…
  Rounding and remainder.
    fix         - Round towards zero.
    floor       - Round towards minus infinity.
    ceil        - Round towards plus infinity.
    round       - Round towards nearest integer.
```

To get help on any of these functions, you can use the online help command. For example, at the MATLAB prompt, >>, type `help sqrt`:

```
>> help sqrt

 SQRT   Square root.
    SQRT(X) is the square root of the elements of X.
Complex
    results are produced if X is not positive.

    See also sqrtm, realsqrt, hypot.

    Overloaded functions or methods (ones with the same
name in other directories)
       help sym/sqrt.m

    Reference page in Help browser
       doc sqrt
```

Clicking on `doc sqrt` takes you to a help documentation page which provides examples of how `sqrt` is used. Another example of the help facility follows:

```
>> help inv
 INV    Matrix inverse.
    INV(X) is the inverse of the square matrix X.
    A warning message is printed if X is badly scaled
or
    nearly singular.

    See also slash, pinv, cond, condest, lsqnonneg,
lscov.

    Overloaded functions or methods (ones with the same
name in other directories)
       help zpk/inv.m
       help tf/inv.m
       help ss/inv.m
       help lti/inv.m
       help frd/inv.m
       help idmodel/inv.m
       help sym/inv.m

    Reference page in Help browser
       doc inv
```

Again clicking on the `doc inv` link takes to the help documentation for the command `inv`. You can also go directly to help page by clicking on the "?" symbol on the top of the menu.

Solving Systems of Equations

Another function which is useful for solving simultaneous equations is "\". \, the backslash command is the matrix left division operator. *a**b* is roughly the same as inv(*a*)**b*, except it is computed in a different way. If *a* is an $n \times n$ matrix and *b* is a column vector with *n* components, or a matrix with several such columns, then $x = a \backslash b$ is the solution to the equation $a * x = b$ computed by Gaussian Elimination. A warning message is printed if *a* is badly scaled or nearly singular.

```
>> A=[1, 3; -1, 4]  % define a 2x2 matrix
A =
     1     3
    -1     4

>> y=[-1; -6]

y =
```

```
    -1
    -6
>> inv(A)*y

ans =

     2
    -1

>> A\y

ans =

     2
    -1
```

Note that all characters following a "%" are ignored. (This is very useful for commenting a script file.) MATLAB computes all numerical answers to a high degree of accuracy. However, you can choose to display your results. The command FORMAT allows you to change the format of your displayed result between 5 significant digits to 15 significant digits.

```
>> pi

ans =

    3.1416

>> format long
>> pi

ans =

   3.14159265358979

>> format short
>> pi

ans =

    3.1416
```

Plotting

The "plot" command plots vectors or matrices. "plot(*x*,*y*)" plots vector *x* versus vector *y*. If *x* or *y* is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever lines up.

```
>> % Generating and sorting a table
```

```
>> x=0:0.01:2;
```
(x is an array of 201 elements ranging from 0 to 2 in steps of 0.01)

```
>> a0=1; a1=3; a2=-5;
```
(define constants a0, a1, a2, and a3)

```
>> a3=1;
>> y=a0+a1*x+a2*x.*x+a3*x.*x.*x;
```
(y is an array with the the 201 expressions of the cubic $a_0 + a_1x + a_2x^2 + a_3x^3$. Note that ".*" is used for element by element multiplication, in contrast to matrix multiplication.)

```
>> z=max(y);
```
(z now contains the maximum value of y)

```
>> plot(x,y);
```
(The cubic y vs. x is plotted using the 201 values)

If you simply type z at the prompt without the semi-colon at the end, you will get

```
>> z

z =

    1.4814
```

and the plot command generates the figure below.

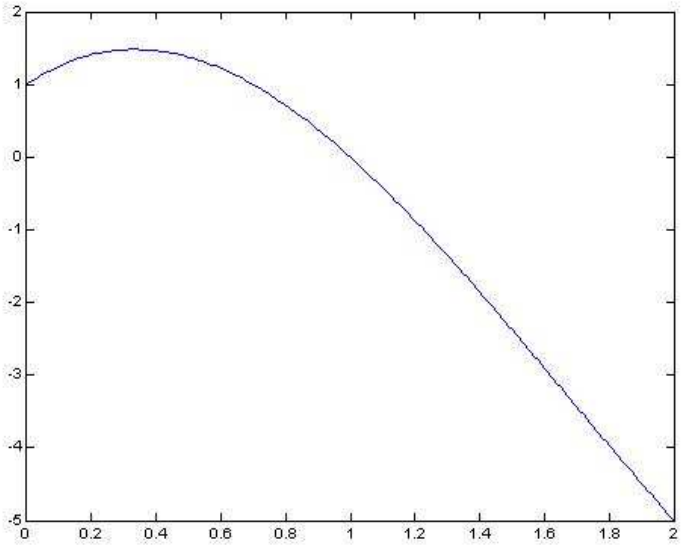


Figure: A plot of the polynomial function $y = a_0 + a_1x + a_2x^2 + a_3x^3$

"plot(*x1*,*y1*,*x2*,*y2*)" is another way of producing multiple lines on the plot, by plotting *y1* vs. *x1* and *y2* vs. *x2* on the same plot. "plot(*x1*,*y1*,*x2*,*y2*,'+')" uses a dotted line for the first curve

and the point symbol + for the second curve. You should use the on-line help documentation or simply type `help plot` at the prompt to find out more about the "plot" command and also other related functions such as semi, loglog, polar, grid, shg, clc, clg, title, xlabel ylabel, axis, hold, mesh, contour, subplot.

`>> w=[x; y]';` (`w'` is the transpose of w. x and y are 1×201 arrays. `[x; y]` is a new 2×201 array. The ";" separating x and y tells MATLAB not to simply attach y at the end of x, but to start a new row for y. Thus w, the transpose of `[x; y]`, is a 201×2 array.)

The following example shows you how to construct matrices, solve a linear system of equations and compute eigenvalues and eigenvectors.

`>> A = [1 2; 3 4]` (enter the 2x2 matrix A)
`>> B = [5 6]'` (enter the column vector B)
`>> A*B` (multiply A and B)
`>> X = A\B` (the solution to AX = B)
`>> [v,d] = eig(A)` (d is a diagonal matrix containing the eigenvalues of A along the diagonal and v is matrix whose columns are the corresponding eigenvectors)

It is easy to generate plots as shown in the following example:

`>> % Plotting example` (all characters following a "%" are ignored)
`>> x=0:0.01:2*pi;` (x is an array ranging from 0 to 2π in steps of 0.01)
`>> y=sin(x);` (y is an array with the sines of the numbers in x)
`>> z=sin(x).*exp(-x);` (z is an exponentially decaying sine wave; the "••" ensures element by element multiplication, while "•" results in matrix multiplication)

```
>> plot(x,y,'-',x,z,'.'); ...
   title('SINE WAVE AND EXPONENTIAL DECAY');
```

(The curves y vs. x and z vs. x are plotted with different symbols; the title is automatically positioned)

Script Files

When doing computations that require a long series of MATLAB commands, it becomes cumbersome to type in the commands interactively. It is possible then to use a text editor such as

the garden-variety editor that comes with MATLAB (which can be started from the menu on the main MATLAB window) or more sophisticated editors such as `emacs` or `Word` to create[1] a script file. The script file should be named "filename.m". Now if the command "filename" is typed at the MATLAB prompt, the sequence of commands in the script file will be executed.

We can create a script file "`PlotMyCurve.m`" that automatically goes through the computations and plots the curves in the above example. In order to do this, we use an editor to create a file "`PlotMyCurve.m`" with our preferred text editor:

```
% Plotting example
x=0:0.01:2*pi;          % generate an equally spaced vector
y=sin(x);               % compute the sine of each point in x
% %%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%
z=y.*exp(-x);           % multiply y with exp(-x)
plot(x,y,'-',x,z,'.'); title('SINE WAVE AND EXPONENTIAL DECAY');
```

Now simply type `PlotMyCurve` at the MATLAB prompt to automatically execute the commands. The semi-colon at the end of each command suppresses intermediate outputs from MATLAB.

Saving files and printing

The default directory for MATLAB is generally the folder or directory in which the executable is stored. The current directory is shown on the main MATLAB window and can be changed to your working directory (usually somewhere in your eniac account under a suitable directory/folder). If you want to save your MATLAB work for future continuation, you can use the command "save". It will, by default, save all your work to `matlab.mat`. When you resume work, you can reload the saved file with the "load" command. The default again is `matlab.mat`, but you can specify the file you want to load. If you want to merely get a hardcopy, the "diary" command saves all prior work in the session in a text file, which can then be printed out.

The menu in the figure window should allow you to manipulate the figure, print it, or save it in different formats.

The `print` command is useful for generating hardcopies of figures. If you type `help print` at the MATLAB prompt you will see that it can be used in different ways. `PRINT`

[1]When using a word processor as an editor, make sure you save the script file as a text file.

`<filename>` saves the current Figure window as PostScript. If a filename is specified, the output is written to the designated file, overwriting it if it already exists. If the specified filename does not include an extension, an appropriate one is appended. The postscript file can be included in any word document or printed to any laser printer. Another option is to copy the figure from the figure menu and paste the figure into a WORD document via the clipboard before printing the document.

Loops

Often, you will need to have Matlab perform the same task over and over. For example, if you wanted to print "Hello" ten times, you could write a script file that contained the command `disp('Hello')` ten times on separate lines. However, this is impractical in general. Instead, you can perform the same task with Matlab using a loop. The most basic loop, known as a `while` loop consists of two elements: a body, and a termination condition:

```
while (termination condition)
     body
end
```

The body is the piece of code that is executed multiple times. Here is a while loop that prints "Hello" ten times:

```
i = 1;
while (i <= 10)
     disp('Hello');
     i = i + 1;
end
```

To make sure the loop terminates after exactly ten times, we use a variable `i` that keeps track of how many times we have printed "Hello." The same result can also be achieved with a different type of loop known as a `for` loop:

```
for variable = array
     body
end
```

We can rewrite the above `while` loop as a `for` loop:

```
for i = 1:10
     disp('Hello');
end
```

This `for` loop consists of an index variable, in this case `i`, an array of values (`1:10` in this case), and a body (`disp('Hello')`). The body is executed once for each element in the array, and each time the body is executed with i assuming the value of that element in the array. For example:

```
for i = 1:10
     i
end
```

will print the numbers 1 through 10 successively on separate lines. Note that the array of elements that the loop iterates over need not be fixed; it can be determined at runtime:

```
% Generate a random integer in [1,10]
v = floor(rand(1) * 10) + 1;

for i = 1:v
     disp('Hello');
end
```

This will print "Hello" a random number of times (between 1 and 10), depending on the choice of `v`.

Exercises

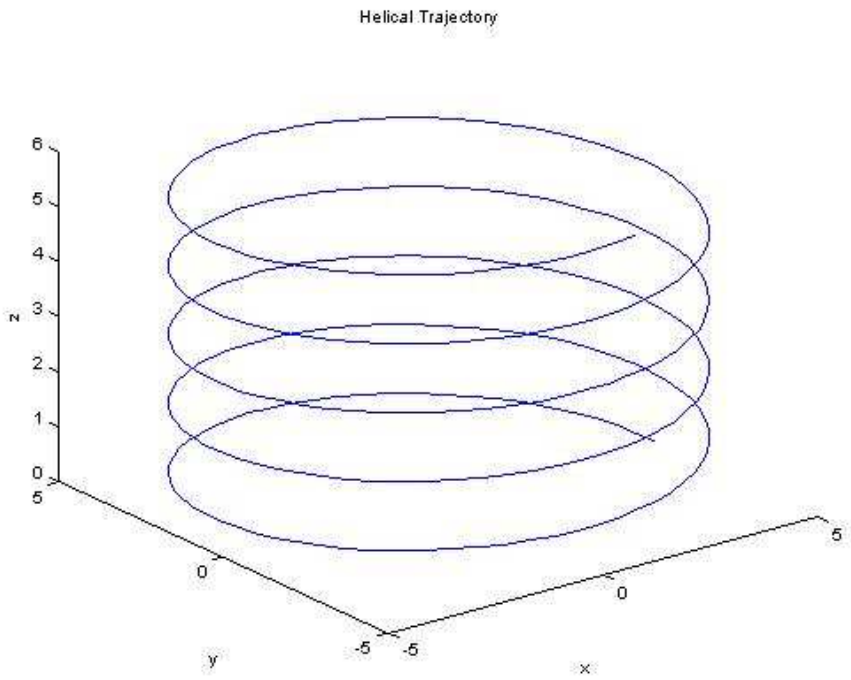
1. A particle moves in a helical motion at a constant pitch with a position vector:

$$\mathbf{r}(t) = R\cos at\,\mathbf{i} + R\sin at\,\mathbf{j} + bt\,\mathbf{k}$$

where R =5, a = 1 and b = 0.2. Plot the trajectory of the particle for 30 seconds, labeling the x, y, and z axes with the caption "Helical Trajectory".

Solution

```
>> t=0:0.1:30; % spacing of 0.1 seconds
>> R=5; a=1; b=0.2;
>> x=R*cos(a*t); y=R*sin(a*t); z=b*t; plot3(x, y, z);
>> xlabel('x'); ylabel('y'); zlabel('z');
title('Helical Trajectory');
```



References

1. Online tutorial for starting students,
http://www.mathworks.com/academia/student_center/tutorials/launchpad.html.
2. Matlab tutorial, http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf