Introduction to MATLAB

Michael K. Johnston and Robert G. King

September 15, 2004

Abstract

MATLAB (**Mat**rix **Lab**oratory) version 6.5 is available in the Institute for Economic Development (IED) computer labs, rooms 514, 523, 525 and 540 of the Economics Department as well as room 327 of the College of Arts and Sciences. We will use Matlab to illustrate some of the mathematical concepts presented in class. This document provides an introduction to some of the basic MATLAB commands and features.

Contents

1 Useful facts **3**

2 Interface Basics **3**

3 Matrices and Vectors **4**

4 Eigenvalues and Eigenvectors **5**

5 Functions **6**

5.1 Function Help 6
5.2 Defining Functions 7
5.3 Executing Functions 8

6 Logical Operators **10**

7 Conditionals **10**

8 Plots **11**

9 References **12**

1 Useful facts

Remark 1.1 (Program Names) *All MATLAB programs end with the suffix* `.m`.

Remark 1.2 (Comments) *Lines beginning with a percent sign* `%` *are* ***comments*** *inserted into the program as explanatory remarks but will not be executed as part of the program.*

Remark 1.3 (Temporary Result) *The built-in variable* `ans` *is always equal to the result of the last calculation not assigned to a variable.*

2 Interface Basics

The **Command Window** is used to enter commands and return the resulting output. Enter commands at the **prompt** >>. For example, define a 3x3 matrix A as follows:

```
>> A = [11 12 13;21 22 23;31 32 33]
```

After you press the return key MATLAB will display the matrix A in the output window

```
A =
 11 12 13
 21 22 23
 31 32 33
```

When entering matrices, vectors and arrays, rows are separated by semi-colons (`;`) and columns are separated by spaces and commas (`,`). In what follows, we will present multiple ways to perform some operations, not all of which will be valid in the command window.

Remark 2.1 (Semi-colons) *Semi-colons have two uses: when found in a line they serve the same purpose as a line ending; when found at the end of a line they serve to suppress output resulting from commands on that line.*

The **Command History** window begins with a commented line showing the initial date and time and is followed by a list of the commands entered thus far. For example, at this point it should look something like:

```
%-- 7/10/04 11:03 AM --%
A = [11 12 13;21 22 23;31 32 33]
```

You may right-click on commands in the list to delete, execute, copy, etc.

The **Workspace** window contains information on variables, strings, arrays, etc. which you have defined. For example, right now the Workspace area should have an entry with the matrix `A` which we defined above. Right-click in the Workspace for additional options (i.e., to view, edit, delete, export, import, etc.). Equivalently, you may see the information displayed in the Workspace by entering **whos** into the Command Window and pressing the Return key.

At the bottom of the Workspace area, click the **Current Directory** tab. At the top of this area the path to the active directory in which you are currently working is displayed. You should adjust this path to point to the directory containing the MATLAB files for the project on which you are currently working. You may right-click for additional options (i.e., to import data, execute programs, open files, etc.).

Click on the ellipses (...) just to the right of the current directory and use the browser which appears to select the folder containing `intro.m`. Double-click on `intro.m` to open it. Alternatively, if you have the appropriate current directory selected, you may just type `intro` at the command prompt >>. From the Debug menu, you may select Run to begin the execution of `intro.m`. Return to the main MATLAB program window and look at the Output pane to see the results of the execution. Throughout the program are inserted `pause` commands which prevent the execution from continuing until the return key is pressed. Read through the explanation of this document and press the return key when appropriate to continue through the introductory program.

3 Matrices and Vectors

It is easy to create vectors in MATLAB via a variety of operations. A **row vector** can be defined on multiple lines as

```
B = [1 2 3]
```

The equivalent **column vector** can be defined on multiple lines as

```
B = [1
     2
     3]
```

or via transposition on a single line as

```
B = [1 2 3]'
```

since the symbol (´) denotes the **transpose** of a vector or matrix with real elements.

A **matrix** can be defined on multiple lines as

```
M = [1 2 3
     4 5 6
     7 8 9]
```

or on a single line as

```
M = [1 2 3; 4 5 6; 7 8 9]
```

where the semi-colon (;) is used to indicate the end of a line (which, in this case, translates to the creation of a new row).

Conducting matrix operations in MATLAB is easy. For example, to get $y = Mx$, we simply write

```
y = M*x
```

Remark 3.1 (Element-by-element Operators) *For many operators (i.e., *, /, \, ^, etc.) adding a period (.) before the operator causes its operation to become element-by-element. Addition and subtraction operators already operate element-by-element.*

Example 3.1 (Element-by-element Operators) *Let x = [2 4]. Then x´*x = [4 8; 8 16], x*x' = [20] and x.*x = [4 16].*

4 Eigenvalues and Eigenvectors

To find the **eigenvalues** and **eigenvectors** of any matrix M, simply write

```
[P,MU] = eig(M)
```

To take the **determinant** of any matrix M, write

```
d = det(M)
```

To get the **inverse** of any matrix M, write

```
IM = inv(M)
```

Note that in this example, the determinant of the matrix M is zero and there is a zero eigenvalue. This is consistent with the general property of matrices that *the determinant is the product of the eigenvalues.*

Remark 4.1 (Inverses) *You can find the left and right inverses of a matrix using backward slashes and forward slashes, respectively.*

Example 4.1 (Inverses) *Given a linear system of the form $y = Ax$ (assuming A square, at least initially), all three of the following are equivalent ways to find a solution*

```
x = A\b;
x = b/A;
C = inv(A);   x = Cb;
```

5 Functions

5.1 Function Help

MATLAB also allows for **functions** that take scalars, vectors and matrices as inputs and produce scalars, vectors and matrices as outputs. In fact, the commands used above – `det, inv` and `eig` – are just functions built-in to MATLAB. To gain further information about these standard functions, you can type

```
help functionname
```

at the prompt.

Remark 5.1 *Type >> `help /` to list all operators and special characters, along with their descriptions.*

Example 5.1 (Using Built-in Help) *At the prompt, type*

>> `help inv`

and press the return key to bring up information defining the function and containing information on warning messages, function documentation and related functions built-in to MATLAB

```
% 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.
%
% Reference page in Help browser
% doc inv
```

Remark 5.2 (Error Messages) *If you attempt to invert the matrix M here, an error message will be displayed because the matrix M is singular and is thus not invertible.*

5.2 Defining Functions

Up to this point, you have seen how to give commands in the Command Window and to use a MATLAB M-file (a file with suffix `.m`) to execute a sequence of such commands. An M-File with a sequence of commands is called a MATLAB **script**. However, M-files can also be MATLAB **functions** – user-defined functions – which take a set of input arguments, use local variables to execute tasks and return output arguments. The general form in which one should delcare MATLAB functions is

```
function [output1,...,outputK] = functionname(input1,...,inputL)
```

This will be followed by a sequence of commands manipulating the input arguments `(input1,..., inputL)` to produce the necessary output arguments.

Remark 5.3 (Local Variables) *All variables used inside the function are local cannot be called from outside the function unless you define the function such that the variables you want to use globally are returned as output arguments.*

Remark 5.4 (Restrictions on Function Names) *One cannot create a USER function with the same name as an existing MATLAB function (i.e., you cannot create your own function with the name `inv`).*

An example of a user function is one that creates a sum of squared values of a vector x. Try creating a function to do this by starting a new MATLAB M-file and entering the commands as they are shown in the example below. Name the M-file `ssq.m` to be consistent with the function name.

Remark 5.5 (Naming Files Consistently) *The name of the M-file in which the function is defined should be the same as the name of the function. If MATLAB cannot find a function it will search for files on the disk of that name with the suffix `.m` in attempt to locate it.*

Example 5.2 (A Simple Function)
```
function y = ssq(x);
% finds the sum of squares (y) from an input vector (x)
    y = 0
    for i=1:length(x)
        y = y + x(i)^2;
    end
```

The commented line in the example function above will be returned if you command >> `help ssq`.

Remark 5.6 (Creating Vectors that are Sequences) *`1:n` is equivalent to `[1 2 ⋯ n-1 n]`. We will discuss how to change the step size from its default (1) later.*

This function uses the MATLAB function `length(x)`, to determine the number of elements of the vector x. It also uses `x(i)` to get the ith element of x and squares the resulting scalar (`x^a` raises x to the ath power, x^a). This function also makes use of a **for-loop**, one of the two kinds of loops offered in MATLAB (the other being a **while-loop**). A for-loop repeats the statements it contains (the statements between `for` and `end`) for each of the values in the loop index (`i=1:length(x)`). Thus the input of this function is the vector x and the output of the function is the scalar y. Using this function produces the sum of squares (since x = [1 2 3] it follows that the squared elements of x are [1 4 9] and their sum is 14).

5.3 Executing Functions

To **execute** this function from a MATLAB program, create a new document by selecting New M-File from the File menu. Save the document as `ssq` (MATLAB will automatically append the `.m` suffix to make the actual file name `ssq.m`) and place it in the same directory as the program with which you wish to use it and simply write

```
y = ssq(x);
```

If ssq.m is in your current directory you may call it from the Output window. In practice, there are simpler ways of creating the sum of squares and one need not create a special function for this purpose. First, since x is a column vector, it follows that

```
y1 = x´*x
```

yields the sum of squares. Second, one can perform operations on each element in a vector with various MATLAB commands. For example,

```
y2 = x.^2
```

yields a new vector with the squared elements of x. At this point, you should already be aware of the difference between the operators `.*` and `*`. There is a corresponding difference between `.^` and `^`.

Using the built-in MATLAB function, we can **sum** the elements of this new vector by writing

```
y2 = sum(y2);
```

to get a third measure of the sum of squares.

Remark 5.7 (Suppressing Item Display) *Notice that the last two measures are not displayed because they are followed by a semi-colon `(;)`.*

To be certain that all three measures are equal, we can form a vector

```
d = [y y1 y2];
```

and then use the **display** command `disp` to display text and the vector.

```
disp(´Three measures of the sum of squares´)
disp(d)
```

Many frequently used matrices can be generated from built-in functions. To create an m by n **matrix of zeros**, write

```
zeros(m,n)
```

To create an m by n **matrix of ones**, write

```
ones(m,n)
```

To create an n by n **identity matrix**, write

```
eye(n)
```

For any n-vector v, an n by n diagonal matrix with the elements of v on the diagonal, write

```
diag(v)
```

Remark 5.8 (Other Built-in Functions) *There are also built-in functions for most trigonometric relationships (i.e., sin(x), cos(x), tan(x), etc.) and many useful constants (i.e., pi).*

6 Logical Operators

The following binary operators return the values 0 and 1 for scalar arguments:

$<$	*less than*
$>$	*greater than*
$<=$	*less than or equal to*
$>=$	*greater than or equal to*
$==$	*logical equality*
$\tilde{}=$	*not equal*

Example 6.1 (Logical Operators) `>> 2<1`

```
ans =
 0

>> 2>1
ans =
 1
>> 0==1
ans =
 0
```

Remark 6.1 (Relations and Assignments) *Note that* `==` *is used in a relation (i.e., as a test of equality), while* `=` *is used to assign an equality.*

The binary operators described above can be modified by logical operators

&	*and*
|	*or*
$\tilde{}$	*not*

Remark 6.2 (Logical Operators with More than One Variable) *The use of non scalar-valued binary operators is slightly more tricky because the comparison takes place element-by-element. This can yield counterintuitive results. For example,* `~=` *will return false only if* ***all*** *elements are different, but a false test does not imply the two matrices are the same (it might be the case that only one element between the two is the same).*

7 Conditionals

The basic form for conditionals as they are used in MATLAB is as follows

```
if expression
      statements
elseif anotherexpression
      statements
⋮
⋮
else
      statements
end
```

Thus, when executed, the terms of each expression is evaluated and the appropriate statements (those following a conditional which tests true) are executed.

Example 7.1 (Flipping A Coin) *x = rand(1,1)*

```
if 0 <= x < .5
      disp('Tails')
elseif .5 <= x <= 1
      disp('Heads')
else
      disp('Error.  Number not a member of [0,1]')
end
```

8 Plots

MATLAB contains commands for loading data from ASCII files and for plotting, as we will illustrate. We begin by looking at quarterly real GDP for the United States for the post-war period 1947Q1-2004Q1. Use the `GDPDAT.txt` file provided or download it yourself from the web site of the Federal Reserve Bank of St. Louis (http://research.stlouisfed.org/fred/). The St. Louis branch of the Federal Reserve maintains an easy to use set of data files on a large number of macro and financial aggregates. Double-click on `GDPDAT.txt` to view the contents of the text file. Observe that after several lines of header information (ignored by MATLAB because of the percent signs `%` at the beginning of each line) the data are of the form

```
1947-01-01 1570.5
1947-04-01 1568.7
1947-07-01 1568.0
1947-10-01 1590.9
⋮    ⋮    ⋮
2003-07-01 10493.1
2003-10-01 10600.1
2004-01-01 10702.1
```

Obviously the first column contains the date and the second column contains the GDP information in which we are interested. Load the `GDPDAT.txt` file into an array with the name `GDPDAT` by writing

```
load GDPDAT.txt;
```

To use only the second column, write

```
gdp = GDPDAT(:,2);
```

This makes use of a **selection** command for use with matrices which instructs MATLAB to take all of the rows of the second column of the array `GDPDAT` and to create a vector called `gdp`. To make this more clear, observe that to get the first four rows of the second column only, we can write

```
gdp2 = GDP(1:4,2)
```

Remark 8.1 (Data Structures) *Additional information on selecting components from matrices is available in the Data Structures subsection of the Programming section of the MATLAB help manual.*

Since the data are quarterly from 1947Q1 through 2004Q1, we can create an accompanying date vector (row vector) by writing

```
date = 1947:.25:2004;
```

To be clear, the first argument here indicates the starting value, the last the ending value and the middle the increment (step) size. We have selected .25 because the data are quarterly. However, it should be clear that if the data were annual one should make use of the default unit increment and simply write

```
date2 = 1947:2004;
```

To create a plot with the date on the horizontal axis and log GDP on the vertical axis, use the built-in MATLAB function `plot`. Write

```
plot(date,log(gdp))
```

Label each of the axes and provide a title for the plot by writing

```
xlabel('Date')
ylabel('log(Billions of Chained 2000 Dollars)')
title('U.S. real GDP in the last half century')
```

9 References

http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html