MATLAB Tutorial for Linear Systems

For Undergraduate Students in Linear Systems (332:345) and Introduction in Automatic Control (332:415)

Rutgers University, Fall 1998

MATLAB Tutorial—Linear Systems and Control

I. Basic MATLAB Functions[1]

Here we present basic, general, MATLAB functions related to the basic mathematical operations and procedures of inputting data, starting and quitting MATLAB.

Entering and Quitting MATLAB

To enter MATLAB type `matlab`. To quit MATLAB type `quit` or `exit`.

The HELP Facility

MATLAB has an extensive built-in help system. If help is required on any function, simply type `help` followed by the function in question. MATLAB displays a brief text indicating the function use and examples of its usage. A HELP facility is available, providing on-line information on most MATLAB topics. To get a list of HELP topics type `help`. To get HELP on a specific topic, type `help` topic. For example, `help eig` provides information on the use of the eigenvalue function.

MATLAB was written so that it can be used much in the same way as you would if you were writing on paper. For example, to find the solution to 21/3, you would type

```
>>21/3
ans =
     7
```

Note that the double right caret, >>, indicates that MATLAB is ready for your input. Also note that any comment used in MATLAB starts with a percent sign. It is important to point out that no variable is either defined or used in the above operation. To assign the above value to the variable `x`, just use the assignment operator "=". Example,

```
>>x=21/3
x=
     7
```

Now the variable `x` has a value assigned to it.

There are two things to remember here. One is that `x` will hold that value until it is changed, until it is cleared, or until the MATLAB session is terminated. Typing the command `who` causes MATLAB to show all of the variables that have been declared thus far. The command `whos` lists all variables that have been declared plus additional information such as the type of variable, size of the variable, and the amount of RAM being consumed due to all of the variables that have been assigned. Issuing the `clear` command purges all of the declared variables from memory. The second thing to remember is that MATLAB defaults to being case sensitive, that is, `x` does not equal to `X`.

To see the present value of a variable, simply type the variable name and hit return.

```
>>x
```

[1] This part is mostly written by T. McCrimmon, a former Rutgers University undergraduate student.

```
x=
     7
```

The basic mathematics operators are

+ addition
– subtraction
* multiplication
/ division
^ power

Brackets are reserved for the identification of vectors and matrices. One of the special operators is the semicolon. The prior examples have not included the semicolon and the result was immediately displayed. Issuing the semicolon at the end of the line stops the result from being displayed. The advantages of this will become obvious when writing and debugging scripts.

```
>>x^2;
>>% does not display the result
>>x
x=
     49
```

The ellipses (three or more periods) is the concatenation function which is for continuing equations on the next line that will not fit on one line.

```
>>x=1+2+3+4  ...
+5+6  returns
x=
     21
% note a blank space after 4
```

Numbers and Arithmetic Expressions

Conventional decimal notation, with optional decimal point and leading minus sign, is used for numbers. A power-of-ten scale factor can be included as a suffix. For example

```
3         -99        0.0001
9.64595   1.606E-20  6.066e23
```

MATLAB includes several predefined variables such as `i,j,pi,inf,NaN`. The `i` and `j` variables are the square root of –1, `pi` is π, `inf` is ∞, and `NaN` is not a number, e.g. divide by zero. MATLAB will return an `NaN` error message if a divide by zero occurs.

Vectors and Matrices

Entering a vector or matrix is a painless operation. Simply use the brackets to delimit the elements of the vector, separated by at least one space. Do not use a comma to separate the elements. For example

```
>>x=[1  2  3  4]
```

results in the output

```
x  =
       1   2   3   4
```

Entering matrices is basically the same as entering vectors except that each row is delimited by a semicolon.

```
>>y=[1  2;3  4]
y  =
       1   2
       3   4
```

A matrix may also be entered this way (carriage returns replace the semicolons)

```
>>y=[1  2
     3  4]
y  =
       1   2
       3   4


>>y(2,2)
```

results in

```
ans=
       4
```

Using the parenthesis in this fashion allows the user to access any element in the matrix, in this case, the element in the second row and second column.

Matrix Operations

Transpose

The special character ' (apostrophe) denotes the transpose of a matrix. The statements

```
>>A=[1  2  3;4  5  6;7  8  0]
>>B=A'
```

result in

```
A  =
       1    2    3
       4    5    6
       7    8    0
B  =
       1    4    7
       2    5    8
       3    6    0
```

Addition and Subtraction

Addition and subtraction of matrices are denoted by + and –. For example, with the above matrices, the statement

```
>>C=A+B
```

results in

```
C  =
        2    6   10
        6   10   14
       10   14    0
```

Matrix Multiplication

Multiplication is denoted by *. For example, the statement

```
>>C=A*B
```

multiplies matrices $\mathbf{A}$ and $\mathbf{B}$ and stores the obtained result in matrix $\mathbf{C}$.

Matrix Powers

The expression `A^p` raises $\mathbf{A}$ to the p-th power and is defined if $\mathbf{A}$ is a square matrix and p is a scalar.

Eigenvalues and Eigenvectors

If $\mathbf{A}$ is an $n \times n$ matrix, the n scalars λ that satisfy $\mathbf{Ax} = \lambda\mathbf{x}$ are the eigenvalues of $\mathbf{A}$. They are found by using the function `eig`. For example

```
>>A=[2  1;0  3]
>>eig(A)
```

produces

```
ans =
     2
     3
```

Eigenvectors are obtained with the statement

```
>>[X,D]=eig(A)
```

in which case the diagonal elements of $\mathbf{D}$ are the eigenvalues and the columns of $\mathbf{X}$ are the corresponding eigenvectors.

Characteristic Polynomial

The coefficients of the characteristic polynomial of the matrix $\mathbf{A}$ are obtained by using the function `poly`. The characteristic equation, a polynomial equation, is solved by using the function `roots`. For example, the statement

```
>>poly(A)
```

for the matrix $\mathbf{A}$ given by

```
>>A=[1  2  3;4  5  6;7  8  0]
```

produces

```
p  =
    1  -6  -72  -27
```

The characteristic polynomial is given by $s^3 - 6s^2 - 72s - 27$. The roots of the characteristic

equations are obtained using the function `roots` as

```
>>r=roots(p)
```

producing

```
r  =
     12.1229
     -5.7345
     -0.3884
```

Polynomials

Entering polynomials is as simple as entering a vector. The following polynomial $h(s) = 5s^4 + 10s^2 + 18s + 23$ would be entered starting with the highest order first, as follows:

```
>>h=[5  0  10  18  23];
```

We have already seen that the function `roots(h)` finds all solutions of the polynomial equation $h(s) = 0$. The other useful MATLAB functions dealing with polynomials are:

1. `polyval(h,10)`
   evaluates the polynomial $h(s)$ at $s = 10$;
2. `[r,p,k]=residue(a,b)`
   performs partial fraction expansion where **a** = numerator, **b** = denominator, **r** = residues, **p** = poles, k = direct (constant) term;
3. `c=conv(a,b)`
   multiplication of polynomials $a(s)$ and $b(s)$. Note that the vector **c** contains coefficients of the polynomial $c(s)$ in descending order;
4. `[q,r]=deconv(c,a)`
   divides polynomial $c(s)$ by $a(s)$ with the quotient given by $q(s)$ and remainder by $r(s)$.

Examples:

```
>>p=[1  5  6];
>>polyval(p,1)
```

produces

```
ans=
      12
```

evaluates the polynomial $s^2 + 5s + 6$ at $s = 1$.

```
>>r=roots(p)
```

```
r=
      -3
      -2
```

finds roots of $p(s) = 0$.

```
>>a=[1  2  3];
>>b=[4  5  6];
>>c=conv(a,b)
```

```
c=
     4  13  28  27  18
```

Plots

The `plot` command creates linear *x–y* plots. If `y` is a vector, `plot(y)` produces a linear plot of the elements of `y`. Notice that the data are autoscaled and that *x–y* axes are drawn.

Loops

MATLAB provides several methods of looping. The `for`, `while`, and `elseif` are the most useful. Each one of these requires an `end` statement at the end of the loop. They are used much in the same way as one would in any program language with the exception of incrementing. This is done using the colon. The following is a quick set of examples using these methods of looping.

Example 1:

```
>>for k=1:2:10
for m=1:5
if k==m
A(k,m)=2;
elseif abs(k-m)==1
A(k,m)=-1;
else
A(k,m)=0;
end
end
end
```

Example 2:

```
>>count=0
>>n=10
>>while (n-1)>=2
count=count+1;
n=n-1;
end
```

For more information about MATLAB looping functions use `help` followed by the function name.

Script Files

Script files, heretofore known as m-files, are simply text files containing all of the code necessary to perform some function. As stated before, a text editor is required for the creation and editing of m-files. The name m-files comes from the fact that the extension for all scripts must be m, e.g. script.m is an acceptable filename. The exclamation mark tells MATLAB that the following line is to be executed by the operating system. Not only is this useful in running the text editor from within MATLAB, but operating system commands, such as accessing an external port from an m-file, may also be run from within MATLAB. M-files can also become a function, where arguments are passed from MATLAB to the m-file and processed. Also m-files

maybe written to execute interactively, prompting the user for information or data. MATLAB was written in C and uses many of the I/O functions found in C, increasing its flexibility even more.

To access an m-file simply execute the following:

```
>>!editor script_name.m
```

`!` causes MATLAB to move the command to the operating system.

`editor` is the name of your editor.

`script_name.m` is the filename; make sure that `.m` is the extension.

Normally, when writing an m-file, several lines of comments are placed at the top of the file indicating the purpose and use of the file. The help function can access these lines and display them as it would any of the other functions in the system. So, later on, long after you have forgotten how to use the file, simply type `help filename` (no extension is required) and those comments at the top of the file are displayed.

Example: The created m-file finds the poles, zeros, and evaluates $H(s)$ at $s = -10$, where $H(s)$ is given by

$$H(s) = \frac{s^3 + 6.4s^2 + 11.29s + 6.76}{s^4 + 14s^3 + 46s^2 + 64s + 40}$$

The filename is sample1.m

```
num=[1  6.4  11.29  6.76];
den=[1  14  46  64  40];
pole=roots(den);
zero=roots(num);
value=(polyval(num,-10))/(polyval(den,-10));
```

To execute this m-file type the filename without the extension

```
>>sample1
```

and the file will run by itself.

II. Main Linear Algebra and System and Control MATLAB Functions

The main linear algebra MATLAB functions are presented in Table 1.

`B=A'`	Matrix transpose
`C=A+B`	Matrix addition (subtraction)
`C=A*B`	Matrix multiplication
`expm(A)`	Matrix exponent of $\mathbf{A}$, i.e. $e^{\mathbf{A}}$
`inv(A)`	Matrix inversion of $\mathbf{A}$
`eig(A)`	Eigenvalues of $\mathbf{A}$
`[X,D]=eig(A)`	Eigenvectors and eigenvalues of $\mathbf{A}$
`rank(A)`	Calculates the rank of $\mathbf{A}$
`p=poly(A)`	Characteristic polynomial of $\mathbf{A}$
`r=roots(p)`	Root of the polynomial equation
`det(A)`	Determinant of $\mathbf{A}$

Table 1: Linear algebra functions

The system/control MATLAB functions of particular interest for undergraduate students are listed below.

(1) `[NUM,den]=ss2tf(A,B,C,D,in)`
Finds the system transfer function from the in-th input, that is
$\mathbf{H}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} = \mathbf{NUM}(s)/den(s)$
where $\mathbf{NUM}(s)$ contains r (number of outputs) rows.

(2) `[A,B,C,D]=tf2ss(num,den)`
State space form of the transfer function.

(3) `[z,p,K]=ss2zp(A,B,C,D,in)`
This function gives the factored expression for the transfer function

$$H(s) = \frac{K(s - z_1)(s - z_2)\cdots(s - z_m)}{(s - p_1)(s - p_2)\cdots(s - p_n)}$$

(4) `[A,B,C,D]=zp2ss(z,p,K)`

(5) `[z,p,K]=tf2zp(num,den)`

(6) `[num,den]=zp2tf(z,p,K)`
Functions (4)–(6) are self-explanatory.

(7) `co=ctrb(A,B)`
Calculates the controllability matrix.

(8) `ob=obsv(A,C)`
Calculates the observability matrix.

(9) `[y,x]=impulse(A,B,C,D,in,t)`
The impulse response during `t=ti:dt:tf` (`ti` = initial time, `td` = sampling time, `tf` = final time) from the in-th input. Also,
`y=impulse(num,den,t)`
or
`impulse(num,den)`

(10) `[y,x]=step(A,B,C,D,in,t)`
The step response during `t=ti:dt:tf` from the in-th input. Also,
`y=step(num,den,t)`
or
`step(num,den)`

(11) `[y,x]=lsim(A,B,C,D,f,t)`
The system response during `t=0:dt:(k-1)*dt` due to arbitrary inputs whose values at the discrete time instants k are defined in the matrix `f` of dimensional $k \times r$, where r stands for the number of system inputs. Also,
`[y,x]=lsim(A,B,C,D,f,t,x0)`
where `x0` stands for the system initial condition. Also
`lsim(num,den,f,t)`

(12) `[Ad,Bd]=c2d(A,B,T)`
From the continuous-time to the discrete-time linear system, where T stands for the sampling period.

(13) `r=rlocus(A,B,C,D,K)`
The root locus of the control system given in the state space form with $u = -\mathbf{K}y$, where `K` is a vector of gains. Also,
`r=rlocus(num,den,K)`
or
`rlocus(num,den)`

(14) `plot(r,'-')`
Plotting function.

(15) `[mag,phase]=bode(A,B,C,D,in,w)`
Bode diagram, where `mag,phase,` represent magnitude and phase. `w` is a vector of frequencies. Also,
`[mag,phase]=bode(num,den,w)`
or

`[mag,phase]=bode(num,den)`
or
`bode(num,den)`

(16) `[re,im]=nyquist(A,B,C,D,iu,w)`
Nyquist plot, where `w` is a vector of frequencies. Also,
`[re,im]=nyquist(num,den,w)`
or
`nyquist(num,den)`

(17) `[Gm,Ph,wcp,wcg]=margin(num,den)`
The phase and gain margins and the corresponding crossover frequencies.

(18) `gram(A,B)`
The same function is used to calculate the observability Grammian with $\mathbf{A}$ replaced by $\mathbf{A}^T$ and $\mathbf{B}$ replaced by $\mathbf{C}^T$.

Note that the functions `impulse`, `step`, and `lsim` applied to discrete-time domain systems have a prefix `d`. The controllability/observability functions have the same form in both continuous- and discrete-time domains.

III. Some MATLAB Functions for Advanced Courses in Controls

Many problems in control theory require solutions of the algebraic Lyapunov and Riccati equations. These equations can be solved by using MATLAB functions `are,lyap,dlyap,lqr,dlqr`.

The function `are` solves the *algebraic Riccati equation* with `S=B*inv(R2)*B'`. Executing

```
>>P=are(A,S,R1)
```

produces the positive semidefinite solution of the algebraic Riccati equation.

The algebraic Riccati equation can be also solved by using the function `lqr` (linear-quadratic regulator). In addition to solving the algebraic Riccati equation, this function also calculates the optimal feedback gain $\mathbf{F}$. The required solution is obtained by

```
>>[F,P]=lqr(A,B,R1,R2)
```

One can also use the function `lqr2`, which is identical to `lqr`, but produces better accuracy. Note that the discrete-time linear-quadratic regulator problem can be solved similarly by using operator `dlqr`.

The *algebraic Lyapunov equation* $\mathbf{AX}+\mathbf{XA}^T=-\mathbf{Q}$ can be solved by using the MATLAB function `lyap`, for example

```
>>X=lyap(A,Q)
```

Note that the matrix $\mathbf{A}$ ought to be stable in order to get the unique solution. Also note that the above equation is the variance (or filter) type Lyapunov equation, on the contrary to the regulator type Lyapunov equation given by $\mathbf{A}^T\mathbf{X}+\mathbf{XA}=-\mathbf{Q}$. Solution of the last equation is obtained by

```
>>X=lyap(A',Q)
```

that is, one must transpose the matrix $\mathbf{A}$ in this case.

The function `lyap` can be used to solve a more general "Lyapunov type" equation frequently known as Sylvester's equation $\mathbf{AX}+\mathbf{XB}=-\mathbf{C}$. This equation can be solved as

```
>>X=lyap(A,B,C)
```

Discrete-time Lyapunov algebraic equation can be solved by using the MATLAB function `dlyap`. Discrete-time Riccati algebraic equation can be solved by using the MATLAB function `dare`.