CMPE-293 Spring 2004
Feedback Control Design

An Introduction to MATLAB and the Control Systems toolbox

Aravind Parchuri, Darren Hon and Albert Honein

MATLAB is essentially a programming interface that can be used for a variety of scientific calculations, programming and graphical visualization. Its basic data element is an array, and its computations are optimized for this data type, which makes it ideal for problems with matrix and vector formulations. MATLAB is also extendible by means of add-on script packages called toolboxes, which provide application-specific functions for use with MATLAB. For this course, we will mostly be using MATLAB's basic matrix/vector operations and graphing capabilities in conjunction with the control system toolbox.

Getting started

MATLAB can be started by clicking on the program icon on a Windows/Mac machine, or issuing the command **matlab** at a shell prompt in a *nix/X-windows environment. The command window can be used to interactively issue commands and evaluate expressions.

The extensive help files are essential for obtaining information about the various commands and functions available. Typing **help** at the command prompt generates a list of the various function categories and toolboxes with a brief description of each. Typing **help** *topic* generates help on the specified topic, which is generally a MATLAB command or toolbox name. This can be used to get the syntax for different commands. A more user-friendly graphical help system is also available via the help menu.

MATLAB uses conventional notation for real numbers. Scientific is also accepted in the form of a real number followed by the letter **e** and an integer exponent. Imaginary numbers are obtained by using either **i** or **j** as a suffix. Examples of valid numbers: 5, 4.55, 1.945e-20, i, 10+15i.

The basic mathematical operators (**+, -, *, /,** ^) can be used directly at the command prompt to perform calculations, as can various elementary mathematical functions. **help elfun** gives a list of the elementary mathematical functions. Remember that angles are specified in radians to functions like sin() and cos(). Examples of usage:

```
>> 8 + 3i + 5          >> sqrt(-4)          >> sinh(1)
ans =                  ans =                ans =
   13.0000 + 3.0000i          0 + 1.4142i          1.1752
```

Note: ">>" denotes the command prompt, and the lines below show the output corresponding to the command executed at the command prompt. A semi-colon at the end of a command suppresses the output.

Matrices and Vectors

Construction: The simplest way to construct a matrix in MATLAB is to enumerate its elements row by row within square brackets, the rows separated by semi-colons, the elements of each row separated by spaces. These elements can be real/complex numbers or other vectors and matrices, as long as the dimensions match up. For example,

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

Row vectors are simply matrices with one row of elements:

```
>> x = [4 5 6 7]
x =
        4     5     6     7
```

Vectors with equally spaced elements can be constructed using the colon notation:

x = starting value : *increment* : *maximum value*

A default increment of 1 is used if the increment is omitted. If the vector has to have a specific last element, we use the **linspace** command:

x = **linspace**(*first_element*,*last_element*,*number_of_elements*)

Column vectors are constructed as matrices with several rows of one element each:

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

```
y =
     1
     2
     3
```

Equally spaced column vectors are obtained by first generating a row vector using the colon notation or the linspace command, and then applying the transpose operator .'

There are other built-in functions to generate specific types of matrices:

- **eye(***m*,*n***)** generates an *m* x *n* matrix with ones on the main diagonal and zeros elsewhere. If *m* = *n*, **eye(***n***)** can be used instead.
- **zeros(***m*,*n***)** is an *m* x *n* matrix whose elements are all zeros
- **ones(***m*,*n***)** is an *m* x *n* matrix whose elements are all ones
- **diag(***v***)** is a square diagonal matrix with vector *v* on the main diagonal
- **diag(***A***)** is a column vector formed from the main diagonal of *A*

Addressing elements: The element in the *i*th row and *j*th column of a matrix *A* is *A*(*i*,*j*). The subscripts *i* and *j* cannot be negative or zero. In the case of a vector *x*, the *i*th element of the vector is addressed as *x*(*i*).

```
>> A = [1 2 3;7 8 9];
>> A(2,1)
ans =
     7
```

```
>> x = [4 5 6];
x(2)
ans =
     5
```

Matrix operations: Matrix addition, subtraction and multiplication are implemented using the traditional operators +, - and *. The inverse of a square matrix *A* is given by **inv(***A***)**. There are two matrix division operators, \ and /. If *A* is a non-singular matrix, then *A**B* and *B*/A* correspond to the left and right multiplication of *B* by inv(*A*).

The transpose of a matrix *A* is given by ***A*.'** . The expression ***A*'** produces the conjugate transpose of *A*, ie, it transposes *A* and replaces its elements by their complex conjugates. If the elements of *A* are real, then *A*' and *A*.' are equivalent.

Array operations: It is also possible to work with the matrix as an array, ie, to perform uniform element-wise operations. The array operators (+, -, .*, ./, .^) are used for this purpose. For example, if *A* and *B* are of the same dimensions, the expression *A*.**B* would multiply each element of *A* with the corresponding element of *B* to produce a matrix of the same dimension as *A* and *B*.

The built-in functions can be used with matrices and vectors just as they are used with numbers. The function is applied simultaneously to all the elements of the matrix.

```
>>A = [1 2 3;4 5 6];
>>sqrt(A)
ans =
```

```
>> A = [1 2 3];
>> B = [4 5 6];
>> A.*B
```

```
    1.0000    1.4142    1.7321          ans =
    2.2361    2.4495    2.6458               4        10        18
```

Scripts

Instead of executing individual commands at the prompt, it is possible to make a text file with a sequence of commands, ie, a MATLAB script, and execute all the commands in sequence. The script must be a plain ASCII file with a '.m' extension. When this file is in the working directory, typing the name of the file without the extension is sufficient to execute the script.

Plotting

The general form of the 2-d plot command is **plot(***x*,*y*,*S***)** where *x* and *y* are vectors of the same type and dimension and *S* is a string of characters within quotes which specifies plot attributes like color, line style, etc. Use **help plot** to find out options and related commands.

Control system toolbox

The Control System Toolbox is a collection of algorithms, written mostly as M-files, that implements common control system design, analysis, and modeling techniques. Convenient graphical user interfaces (GUI's) simplify typical control engineering tasks. Control systems can be modeled as transfer functions, in zero-pole-gain, or state-space form. **help control** gives a list of the different functions. The most commonly used functions are presented below, with some common modes of usage documented. You are encouraged to look through the help files for other functions and options that you might find useful in future.

Creating models:

- **Transfer function models** are created using the function **tf(***numerator*,*denominator***)** where *numerator* and *denominator* are two vectors containing the coefficients of the polynomials in the numerator and denominator of the transfer function.

```
>> num = [1 2 0];
>> den = [2 2 1];
>> sys = tf(num,den)
Transfer function:
  s^2 + 2 s
---------------
2 s^2 + 2 s + 1
```

- **State space models** are created using the function **ss(***A*,*B*,*C*,*D***)** where *A*,*B*,*C* and *D* are the matrices forming the state space system

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

- **Zero-Pole-Gain** forms of transfer functions, ie,
$$G(s) = \frac{K(s - z_1)(s - z_2)...}{(s - p_1)(s - p_2)...}$$
can be specified directly as **zpk(***z,p,K***)** where *z* and *p* are the vectors of zero and pole values in the complex plane (z = [z_1 z_2 …] and p = [p_1 p_2 …]), and *K* is the gain.

The models thus generated can be used in conjunction with the operators +,- and * to generate new models. *sys1 * sys2* produces a series interconnection from input through *sys2* and *sys1* (in that order) to the output. *sys1* ± *sys2* represent parallel interconnections.

Converting models:

Control system models can be converted from one form to the other. The three functions presented above (ss(), tf(), zpk()) are overloaded to perform arbitrary system conversions. For example, if *sys1* is a state-space model, we can generate an equivalent transfer function model *sys2* by issuing the command

```
>> sys2 = tf(sys1);
```

In addition to these, there are other functions that are used for converting elements of one form(say, the matrices of a state-space model) to the elements of another form(say, the numerator and denominator of a transfer function model: **tf2ss, ss2tf, tf2zp, zp2tf, zp2ss, ss2zp.**

System analysis:

Time domain analysis in the form of time response of a system to a specified input can be obtained using the following commands:

- **step(***sys***)** plots the step response of a system *sys*.
- **impulse(***sys***)** plots the impulse response of *sys*
- **lsim(***sys,input_vector,time_vector,initial_state_vector***)** plots the response of *sys* to an arbitrary input. The elements of *input_vector* define the values of the input at times corresponding to the elements of *time_vector*. The initial state vector can be specified for state-space models only.

Classical and frequency domain analysis tools:

- **rlocus(***sys***)** plots the root locus of the system *sys* with variations in gain.
- **bode(***sys***)** plots the magnitude and phase angle Bode plots.
- **[***Gm,Pm,Wg,Wp***] = margin(***sys***)** calculates the gain margin *Gm*, the phase margin *Pm*, and the frequencies corresponding to their occurrence. Issuing the command **margin(***sys***)** alone plots the Bode diagram and marks the margins on it.

- **nyquist(***sys***)** plots the Nyquist plot of the system. Note that the loop at infinity is not represented on the plot.
- **sisotool(***plant*,*compensator***)** opens an interactive mode of the root locus and bode plots, which can be used to modify the compensator and gain to achieve the desired system characteristics. The function can be called without passing an initial compensator, in which case the poles and zeros of the compensator can be placed using the graphical interface. Final designs can be exported back to the MATLAB workspace for further analysis.

State – space tools

- **ctrb(***sys***)** returns the controllability matrix of a state-space system *sys*
- **obsv(***sys***)** returns the observability matrix of a state-space system *sys*
- **[***K*,*S*,*E***] = lqr(***A*,*B*,*Q*,*R***)** : Linear quadratic regulator design – calculates optimal gain *K* for a state space system, based on the optimizing matrices *Q* and *R* provided by the designer.