A MATLAB- and Simulink-based Signals and Systems Laboratory

Shlomo Engelberg

April 7, 2008

\footnote{Copyright ©2008 by Shlomo Engelberg}
Contents

1 Preface 7

2 Getting Started 9
   2.1 Some Very Basic Instructions ................................. 9
   2.2 MATLAB as a Vector-enabled Calculator .................... 10
   2.3 Ways of Manipulating Vectors ................................. 11
   2.4 Plotting with MATLAB ....................................... 12
   2.5 Using the Editor .............................................. 13
   2.6 Comments ..................................................... 14
   2.7 Exercises .................................................... 15

3 Symbolic Calculations Using MATLAB 17
   3.1 Overview .................................................... 17
   3.2 Getting Started .............................................. 17
   3.3 Plotting Symbolic Functions ................................ 18
   3.4 Substitutions ............................................... 18
   3.5 Exercises .................................................... 19

4 Convolution Using the Symbolic Toolbox 23
   4.1 Overview .................................................... 23
   4.2 An Easy to Perform Convolution .............................. 23
   4.3 An Exercise .................................................. 26
   4.4 Exercises .................................................... 27

5 Solving Differential Equations 29
   5.1 Overview .................................................... 29
   5.2 Solving Ordinary Differential Equations Symbolically ..... 29
   5.3 An Introduction to Matrices ................................ 30
   5.4 Looping in MATLAB ......................................... 31
   5.5 Solving Differential Equations Numerically–A Very Brief Introduction ........................................ 32
Chapter 1

Preface

MATLAB is an environment for performing calculations and simulations of a variety of types. MATLAB, short for matrix laboratory, was developed by Cleve Moler, a professor of mathematics and computer science, in the late 70s and early 80s. In 1984, Cleve Moler and Jack Little founded The MathWorks which has been developing MATLAB ever since. In the earliest version of MATLAB, there were about 80 commands, and they allowed for matrix calculations. Now there are thousands of commands, and there are many, many types of calculations that one can perform [3]. Simulink is a MATLAB “add-on” that allows one to simulate systems by combining blocks of various types. We will make use of Simulink as well.

During the course of this lab, the student will learn how to make calculations using MATLAB and will learn a little about simulating systems using the simulation tools provided by MATLAB and Simulink.
CHAPTER 1. PREFACE
Chapter 2

Getting Started

2.1 Some Very Basic Instructions

To start MATLAB, double click on the MATLAB icon on the computer’s desktop. This opens the main MATLAB window.

When working with MATLAB there are two ways of programming. One can either enter commands in the command window, or one can write functions using the M-file editor. In this first lab, we will take the easy way out and use the command window. At the end of the chapter, we introduce the editor. In all save the first lab, the editor must be used for writing MATLAB code.

The basic MATLAB interface is pretty easy to use. MATLAB does not require that variables be declared; MATLAB allocates space for the variable the first time the variable is used. To assign a value to a variable, one uses an equal sign. To set \( x \) equal to 2, one writes \( x = 2 \). When one does this, MATLAB replies with:

\[
x = 2
\]

Let us consider a few simple calculations. In order to have MATLAB raise a number, \( n \), to a power, \( m \) one writes \( n^m \). Thus, to calculate the 10th power of 2, one writes \( 2^{10} \). After being given this command, MATLAB responds with

\[
\text{ans} = 1024
\]
CHAPTER 2. GETTING STARTED

Note that ans is itself the name of a MATLAB variable, and one can manipulate it just as one can manipulate any other variable.

2.2 MATLAB as a Vector-enabled Calculator

Two features make MATLAB particularly useful. MATLAB knows how to work with vectors, and it knows how to produce all sorts of graphs. Let us start with vectors. Then we will move on to plotting graphs.

To define a vector, \( \mathbf{a} \)—an ordered set of numbers that will be represented by the letter \( \mathbf{a} \)—one types \( \mathbf{a} = [\text{num1 num2 num3 \ldots numN}] \). (One can insert commas between the different elements, but MATLAB does not require that one insert them.) MATLAB recognizes this as a command to allocate memory for \( \mathbf{a} \) and to assign the set of numbers listed to \( \mathbf{a} \). For example, giving MATLAB the command \( \mathbf{a} = [1 2 3 4 5] \) causes MATLAB to respond with

\[
\mathbf{a} = \\
1 \quad 2 \quad 3 \quad 4 \quad 5
\]

Once \( \mathbf{a} \) has been defined there are many ways that it can be processed.

Most MATLAB functions are vectorized; they can take vector arguments, and they will return vectors as answers. When using MATLAB, to calculate the sine of \( \pi \), one types \( \sin(\pi) \). To calculate the value of sine at the points \( \pi, \ldots, 5\pi \), one could give MATLAB five separate commands; that is not necessary, however. Instead, one can type \( \sin(\pi*a) \). (The asterisk denotes multiplication. MATLAB does not “understand” that if one writes \( ab \) one would like MATLAB to multiply to elements. One must always tell MATLAB to multiply them.) MATLAB will respond to the command with

\[
\text{ans} = \\
1.0e-015 * \\
0.1225 \quad -0.2449 \quad 0.3674 \quad -0.4899 \quad 0.6123
\]

Note that MATLAB did not answer 100% correctly. Though its answers are all very near zero, they are not precisely zero.
2.3 Ways of Manipulating Vectors

MATLAB allows the user to refer to a single element of a vector \( a \) by writing \( a(\text{ele}_\text{num}) \). MATLAB always starts numbering elements from element 1. (This is different from C. In C, arrays start from element 0.)

Continuing with the previous example, if one types \( \text{ans}(2) \), MATLAB responds with

\[
\text{ans} = \\
-2.4493\text{e-016}
\]

MATLAB will happily sum the elements of a vector—all one needs to do is type \( \text{sum(vect\_name)} \). Continuing with our example, if one types \( \text{sum(sin(pi*a))} \), MATLAB responds with

\[
\text{ans} = \\
3.6739\text{e-016}
\]

In mathematics, one can add vectors—and the same is true when using MATLAB. The following set of commands is legal, and the answer given is what MATLAB would reply with.

\[
\text{>> a} = [1 \ 2 \ 3] \\
\text{a} = \\
\begin{array}{ccc}
1 & 2 & 3 \\
\end{array} \\
\text{>> b} = [4 \ 5 \ 6] \\
\text{b} = \\
\begin{array}{ccc}
4 & 5 & 6 \\
\end{array} \\
\text{>> a+b} \\
\text{ans} = \\
\begin{array}{ccc}
5 & 7 & 9 \\
\end{array}
\]
It is often convenient to perform an operation on each element of a vector. MATLAB allows one to do this by preceding the operation by a period. Suppose that one defines the vector \( a \) by giving the command \( a = [2 \ 4 \ 6 \ 8 \ 10] \). Then to square each element, one writes \( a.^2 \). To this command MATLAB responds with

\[
\text{ans} = \\
4 \quad 16 \quad 36 \quad 64 \quad 100
\]

A final command that often comes in handy is the command:

\[
\text{low}_\text{lim} : \text{inc} : \text{upper}_\text{lim}
\]

This command causes MATLAB to create a vectors whose first element is \( \text{low}_\text{lim} \), whose elements increase in jumps of \( \text{inc} \), and whose final elements is less than or equal to \( \text{upper}_\text{lim} \) and is within \( \text{inc} \) of \( \text{upper}_\text{lim} \). Thus, giving MATLAB the command

\[
a = 1:2:10
\]

causes MATLAB to respond with

\[
a = \\
1 \quad 3 \quad 5 \quad 7 \quad 9
\]

(If \( \text{inc} \) is less than zero, then the vector’s elements decrease from \( \text{upper}_\text{lim} \) to \( \text{lower}_\text{lim} \).) If one defines a vector by using the command \( \text{low}_\text{lim} : \text{upper}_\text{lim} \), MATLAB interprets the command as \( \text{low}_\text{lim} : 1 : \text{upper}_\text{lim} \). That is, the default increment is one.

### 2.4 Plotting with MATLAB

MATLAB has many functions that allow one to plot data. We consider some of the simplest uses of the simplest of the plotting functions: \texttt{plot}. If one types \texttt{plot(a)}, MATLAB plots the values of \( a \) against the integers. For example, giving MATLAB the command \texttt{plot(1:2:10)} causes MATLAB to respond with Figure 2.1. (By default, \texttt{plot} interpolates between the points in a plot.)

If one gives MATLAB the command \texttt{plot(t,a)}, then MATLAB plots the vector \( a \) against the vector \( t \). Suppose that one gives MATLAB the commands
2.5. USING THE EDITOR

![Figure 2.1: A very simple plot](image)

```matlab
t = 0:0.1:3;
a = exp(-t);
plot(t, a)
```

Then MATLAB will respond with the graph of Figure 2.2. There are two points to note in the code given above. First of all the use of a semi-colon at the end of a line causes MATLAB to suppress the printing of the result of the assignment. When one is dealing with vectors with tens, hundreds, or thousands of elements this is very helpful. Secondly, the function `exp` calculates $e$ to the power given as the argument of the function. As the example demonstrates, the `exp` function is vectorized.

### 2.5 Using the Editor

One can open the MATLAB editor either by typing `edit` at the MATLAB command prompt or by clicking on the blank page on the toolbar on the MATLAB window. (If one would like to open a file in the editor, then one clicks on the folder on the toolbar, selects the relevant file, and then clicks on it. MATLAB will open the file in the editor.) The editor is context
sensitive and has many nice features. When one is finished writing a file, one saves it. MATLAB expects files that contain MATLAB commands to end with a `.m`. After saving a file, one can go to the command window, set the Current Directory to the directory in which the file was saved, and then type the file’s name (without the `.m` suffix). This causes MATLAB to perform the commands in the file. One can also cause MATLAB to execute the commands in the file by hitting F5 while in the editor window.

For the time being we will make use of the editor to save our work and to do the work in a way that makes it easy to correct mistakes and to print the code that we have written.

Please make sure to save what you are working on at regular intervals.

2.6 Comments

Starting a line with a percent sign tells MATLAB that that line is a comment. As with all programming languages, it is very important to comment your code.

MATLAB has a help command. Typing `help com_name` causes MAT-
LAB to respond with information about the command. If one writes a file, places comments at the very beginning of the file, and types `help filename`, MATLAB prints the comments that appear at the beginning of the file. This allows the user to “expand” the MATLAB help facility.

### 2.7 Exercises

1. Recall that the Cauchy-Schwarz inequality states that for any two vectors, \( \vec{x} \) and \( \vec{y} \), the absolute value of the dot (scalar) product of the two vectors is less than or equal to the product of the norms of the two vectors. Moreover, equality is attained if and only one of the vectors is a multiple of the other vector.

   (a) Define the three “vectors”
   
   \[
   \begin{align*}
   \vec{a} &= [1 \ 2 \ 3 \ 4 \ 5] \\
   \vec{b} &= [1 \ 3 \ 5 \ 7 \ 9] \\
   \vec{c} &= [2 \ 4 \ 6 \ 8 \ 10] 
   \end{align*}
   \]
   
   • Calculate the dot products of \( \vec{a} \) with \( \vec{b} \), of \( \vec{a} \) with \( \vec{c} \), and of \( \vec{b} \) with \( \vec{c} \).
   
   • Calculate the norms of each of the vectors.
   
   • Verify the Cauchy-Schwarz inequality for the three dot products calculated above.

2. MATLAB treats the command \texttt{const ./ vect} to mean create a new vectors whose elements are \texttt{const} divided by the elements of \texttt{vect}. Using MATLAB and the commands we have seen, calculate

   (a) \( \sum_{k=0}^{100} k \)

   (b) \( \sum_{k=0}^{100} k^2 \)

   (c) \( \sum_{k=1}^{100} 1/k \)

   (d) \( \sum_{k=1}^{100} 1/k^2 \)

   (e) \( \sum_{k=1}^{10000} 1/k^2 \)

   (f) “Bonus question” To what number are the series of 2d and 2e tending?

3. “Bonus question” Derive the formula for

   \[
   \sum_{k=0}^{N} k^2.
   \]
You may wish to consult [1].

4. Plot five periods of the function \( \cos(2\pi t) \). Define a vector \( t \) with 100 elements the first of which is zero.

5. Write a file that expects the variable \( N \) to be defined already. Have the commands in the file cause MATLAB to calculate the sum

\[
\sum_{k=1}^{N} \frac{1}{k^2}.
\]

Add comments to the beginning of the file so that typing `help file_name` causes MATLAB to explain what the code does.
Chapter 3

Symbolic Calculations Using MATLAB

3.1 Overview

Though MATLAB was not really designed to handle symbolic calculations, as the need for such an ability became clear The MathWorks made an arrangement with the designers of Maple—which is a symbolic calculation package. MATLAB’s symbolic math toolbox is a sort of “front end” for Maple. In this lab, we learn how to perform symbolic calculations using MATLAB. The book [2] is a good general reference for this subject.

3.2 Getting Started

To define a symbolic variable, x, one can either use the command `syms x` or `x = sym('x')`. The first command is shorter; the second command is somewhat more versatile. One can define other symbols in terms of symbols that have already been defined.

If one gives MATLAB the commands

```matlab
x = sym('x')
y = 1/ (1 + x^2)
```

MATLAB responds with

```matlab
y =
1/(1+x^2)
```
18

CHAPTER 3. SYMBOLIC CALCULATIONS USING MATLAB

The extremely observant reader will note that when MATLAB printed its output here, it did not indent the answer. The symbolic value $1/(1+x^2)$ is printed flush with the margin. In general, MATLAB indents numerical values but prints symbolic values flush with the margin.

3.3 Plotting Symbolic Functions

MATLAB has a separate command for plotting symbolic functions: \texttt{ezplot}. The simplest way to use \texttt{ezplot} is to give the command \texttt{ezplot(y)}. MATLAB then plots the function over a default interval. With $y$ as defined above, giving the command \texttt{ezplot(y)} causes MATLAB to respond with Figure 3.1. If one would like to cause \texttt{ezplot} to plot from $x_{\text{min}}$ to $x_{\text{max}}$, one can use the command \texttt{ezplot(y, [xmin xmax])}.

3.4 Substitutions

MATLAB provides a command that allows us to substitute values, symbolic or otherwise, for symbolic values. The command that performs the substitu-
tion is called \texttt{subs}. To substitute 2 in the \( y \) we defined previously, one gives MATLAB the command \texttt{subs(y, 2)}. When given this command, MATLAB responds with

\[
\texttt{ans = } 0.2000
\]

Note that, based on the indentation, we find that MATLAB has given us a numeric value—and not a symbolic one. It is possible to tell MATLAB that our intention is to substitute the symbol 2—and not some representation of 2. (That is, we tell MATLAB to use an “ideal” 2 and to try to keep everything as ideal as possible.) To do this one uses \texttt{sym(’2’)} rather than 2. Giving MATLAB the command \texttt{subs(y, sym(’2’))} causes MATLAB to respond with

\[
\texttt{ans = } 1/5
\]

Note that the answer here is not indented. MATLAB \textit{knows} that the symbol it is giving is the correct and precise value it should be returning.

It is also possible to have MATLAB substitute one symbolic value for another. The command \texttt{subs(y, x^2)}, for example, causes MATLAB to respond with

\[
\texttt{ans = } 1/(1+x^4)
\]

\section*{3.5 Exercises}

1. Define the symbolic variable \( x \). Make use of this variable to define the symbolic function \( \sin(1/x) \). Then, plot the function from 0 to 2\( \pi \). What do you see? Why might the plotting routine be having trouble plotting this function?

2. (Odd and even functions.) To break a function, \( f(x) \), into odd and even parts, one can compute the two functions

\[
\begin{align*}
f_{\text{odd}}(x) &= \frac{f(x) - f(-x)}{2} \\
f_{\text{even}}(x) &= \frac{f(x) + f(-x)}{2}.
\end{align*}
\]
You will now perform these operations using MATLAB and examine the resulting functions.

(a) Define the symbolic variable \( x \).

(b) Define the symbolic function \( y = e^x \) (using the appropriate MATLAB commands).

(c) Define the symbolic function \( z = e^{-x} \) by making a substitution in the symbolic function you found in the previous section.

(d) Using the results of the previous two sections, calculate \( f_{\text{odd}}(x) \) and \( f_{\text{even}}(x) \) for the function \( y = e^x \).

(e) Use \texttt{ezplot} to plot the functions that you found over the region \([-1, 1]\).

3. Let us evaluate \( \sin(x) \) at the points \( \pi, \ldots, 5\pi \) again.

(a) Define the symbolic vector \( a \) by giving the command

\[
a = \text{sym}(1'):\text{sym}(5').
\]

This command should give you an array of five “ideal” numbers.

(b) Define the symbol \( p = \text{sym('pi')} \). This command gives you a variable, \( p \), that holds an ideal \( \pi \).

(c) Now have MATLAB calculate the values of \( \sin(\pi), \ldots, \sin(5\pi) \) using a single command.

(d) What is the command’s output? Why does it differ from the values found in §2.2.

4. (Calculus using the symbolic toolbox) The symbolic toolbox has many commands that allow one to calculate sums, derivatives and integrals of symbolic quantities. In this exercise we demonstrate a few of the toolbox’s abilities.

(a) The command \texttt{symsum} causes MATLAB to sum a symbolic expression. Give MATLAB the command \texttt{symsum(1/sym('n')^2,1,inf)}.

That is, have MATLAB calculate

\[
\sum_{n=1}^{\infty} \frac{1}{n^2}.
\]

Compare this with the values found in exercises 2d and 2e of Chapter 2.
(b) Calculate the indefinite integral of $1/(1+t^2)$ by using the symbolic integration command, \texttt{int}.

(c) Calculate the integral
\[
\int_{-\infty}^{\infty} e^{-x^2/2} \, dx
\]
by using the command \texttt{int(funct, low_lim, upper_lim)}. Please note that \texttt{inf} is the “ideal” infinity and \texttt{-inf} is the ideal negative infinity.
CHAPTER 3. SYMBOLIC CALCULATIONS USING MATLAB
Chapter 4

Convolution Using the Symbolic Toolbox

4.1 Overview

In this chapter we describe how to use the symbolic toolbox to calculate the convolution of two functions. As we will see, such calculations seem to be on the border of what the symbolic toolbox is capable of performing.

4.2 An Easy to Perform Convolution

Let us consider the convolution of $1/(1 + t^2)$ with itself. That is, let us calculate

$$\int_{-\infty}^{\infty} \frac{1}{1 + \tau^2} \frac{1}{1 + (t - \tau)^2} d\tau.$$

To do this, we give MATLAB the commands

```matlab
syms t tau
f = 1/(1 + t^2)
z = int(subs(f, tau)*subs(f, t-tau), tau, -inf, inf)
z = simplify(z)
figure(1)
ezplot(f)
figure(2)
ezplot(z)
```

MATLAB responds with

$$f =$$
CHAPTER 4. CONVOLUTION USING THE SYMBOLIC TOOLBOX

Figure 4.1: The graph of the function $1/(1 + t^2)$

\[ \frac{1}{1 + t^2} \]

\[ z = \frac{2\pi t^2}{t^4 + 4t^2} \]

\[ z = \frac{2\pi}{t^2 + 4} \]

and Figures 4.1 and 4.2. (The command `simplify(z)` causes MATLAB to simplify the symbolic expression given by $z$, and the command `figure(num)` causes MATLAB to open a figure window which it numbers `num`.)

The calculation of the integral makes use of the `int` command. This command performs symbolic integration. As we have used it, the first argument is the function being integrated. The second argument is the vari-
Figure 4.2: The graph of the function $1/(1 + t^2)$ convolved with itself
able over which the integration is to be performed. The third and fourth arguments are the limits of integration. There are many simpler forms of the command. Making use of the MATLAB help command, one can find out quite a bit about the command. (The general form of the help command is help command_name. To find out more information about int, type help int.) MATLAB responds to the above commands with all the functions—in functional form—and with the graphs of the functions.

4.3 An Exercise

Now we perform a convolution that is more difficult to perform using MATLAB. We consider the convolution of the unit step function with itself. MATLAB does not have a built-in unit step function. Thus, we must design it ourselves. We start by putting together a signum function—a function that returns a one if its input is positive and a minus one if its input is negative. To do this, we give MATLAB the commands

```
syms x
sig = x / abs(x)
```

This looks a bit nasty, but it is pretty clear that for all positive numbers its output is one and for all negative numbers its output is minus one. Try using ezplot to plot the function. If using the simplest form does not work, try providing ezplot with the region as well.

Next, we define the step function in terms of the function we have already created. It is clear that the command

```
st = (sig + 1)/2
```

will do the trick. Please plot this function as well.

Having defined the step function, we must calculate the convolution. To do this we give the commands

```
syms xi
ramp = int(subs(st,xi) * subs(st,x - xi), xi, -inf, inf)
```

At this point (at least when using MATLAB 6.5), MATLAB flatly refuses to allow ezplot to plot the function. One work-around is to use the subs command to substitute values into the symbolic function and then to plot the values. Giving MATLAB the commands

```
t = [-9.9:0.2:9.9];
a = subs(ramp, t);
plot(t,a)
```
4.4 Exercises

1. Please calculate the convolution of §4.2 analytically. (You may want to use residues or Fourier transforms.)

2. Using the symbolic toolbox and the tricks of this chapter, have MATLAB calculate and plot the convolution of a unit pulse,

\[ u(t) = \begin{cases} 
1, & 0 \leq t < 1 \\
0, & \text{otherwise}
\end{cases} \]

with itself.
CHAPTER 4. CONVOLUTION USING THE SYMBOLIC TOOLBOX
Chapter 5

Solving Differential Equations

5.1 Overview

In this chapter we describe several ways of solving differential equations using MATLAB. We make use of the symbolic toolbox when possible, and we also use numerical methods when desirable. Finally, we show how to use these tools to evaluate the behavior of a linear time-invariant (LTI) system.

5.2 Solving Ordinary Differential Equations Symbolically

MATLAB has a command, `dsolve`, that solves ordinary differential equations (ODEs) symbolically. One form of the `dsolve` command is

\[ \text{dsolve('ODE','initial conditions')} \]

In the ODE string, \( D \) is used to represent the first derivative and \( D^n \) represents the \( n \)th derivative. To solve the differential equation \( y'(t) = ay(t) \) subject to the initial condition \( y(0) = c \) and assign the solution to \( y \), one gives the commands

```matlab
syms a c
y = dsolve('Dy = a*y','y(0) = c')
```

MATLAB replies with

\[ y = c \exp(a \cdot t) \]

It is possible to solve much more complicated differential equations this way.
5.3 An Introduction to Matrices

Our next major goal is to solve differential equations numerically. In order to do this, we will make use of matrices as well as vectors. If one would like MATLAB to “create” the $M \times N$ dimensional matrix

$$ A = \begin{pmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MN} \end{pmatrix}, $$

One gives MATLAB the command

```
A = [a11 ... a1N; ... ; aM1 ... AMN]
```

That is, one lists the elements of each row separating the elements by spaces (or commas) and separating the rows by semicolons.

If one gives MATLAB the command $A = [1 \ 0; \ 0 \ 1]$, MATLAB responds with

$$ A =
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix} $$

Though many of the commands provided by MATLAB do not care whether a vector is a column vector or a row vector, when using matrices the difference is critical. If one defines a vector using the command $a = [1 \ 2 \ 3]$, one has defined a three-element row vector. To define a column vector, one separates each element by using a semicolon. If, for example, one gives MATLAB the command $v = [1; \ 2; \ 3]$, then MATLAB responds with

$$ v =
\begin{pmatrix}
1 \\
2 \\
3
\end{pmatrix} $$

One can use MATLAB as a simple matrix calculator without defining the matrices and vectors “variables” first. If, for example, one needs to calculate

$$ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} $$

one gives MATLAB the commands
[1 2 3; 4 5 6; 7 8 9] * [1; 2; 3]
and MATLAB responds with

\[ \text{ans} = \begin{array} {ccc} 14 \\ 32 \\ 50 \end{array} \]

(Once again, note that when one would like MATLAB to multiply to objects, one must tell MATLAB to multiply the objects by using an asterisk (*)�)

To access the elements of a matrix, one gives the matrix’s name and the position of the element to be accessed. To access \( A_{mn} \), one types \( A(m,n) \). To access the elements of a vector, type the vectors name and the number of the element. In order to access the second element of the \text{ans} vector, for example, one types \text{ans}(2). To this command, MATLAB responds with

\[ \text{ans} = 32 \]

(It is worth noting that this new \text{ans} variable has overwritten the old \text{ans} vector.)

MATLAB has many, many commands for matrix manipulation. To list just a few:

- \text{inv(A)} which inverts the (previously defined) matrix \( A \).
- \( A^2 \) which calculates \( A^2 \).
- More generally, \( A^N \) which calculates \( A^N \).
- \( A' \) which calculates the conjugate transpose of the matrix \( A \).

For still more matrix operations, give MATLAB the command \text{help elmat} or \text{help matfun}.

### 5.4 Looping in MATLAB

When one is programming, it is often necessary to perform a set of instructions many times. One way to do this is to use some form of loop. MATLAB has a very simple \text{for} loop.

The structure of the \text{for} loop is
The expression is a row vector. If one starts a loop with for i = 1:10, then one is “asking” MATLAB to loop through the following commands until the matching end is hit. On the first loop, i will be one. On the tenth it will be 10. Note that indenting the contents of the loop is not necessary, but it makes the code easier to read. The MATLAB editor generally indents the contents of a loop.

As a simple example, to have MATLAB calculate ten factorial, one can give MATLAB the commands

```matlab
out = 1;
for i = 2:10
    out = out * i;
end
out
```

MATLAB will respond with

```matlab
out =
    3628800
```

which is indeed 10!.

### 5.5 Solving Differential Equations Numerically–A Very Brief Introduction

Consider the equation

\[
\frac{d}{dt}y(t) = ay(t) + x(t), \quad y(0) = \alpha.
\]

Let \( x_k = x(kT_s) \), \( k \geq 0 \). Considering the definition of the derivative, it is not unreasonable to hope that if \( T_s \) is sufficiently small, then the sequence
5.6. CONVERTING A HIGH-ORDER ODE INTO A FIRST-ORDER ODE

\{y_k\} defined by

\[
\frac{\approx \frac{dy}{dt}}{T_s} = ay_{k+1} + x_{k+1}, \quad y_0 = \alpha, \quad k \geq 0
\]

will tend to \( \{y(kT_s)\} \). That is, for small \( T_s \) we should find that \( y_k \approx y(kT_s) \). If \( T_s \) is small enough, the sequence should provide a good approximation to the solution of the differential equation.

Consider, for example, the equation \( y' = y, \quad y(0) = 1 \). Clearly, the exact solution of this equation is \( y(t) = e^t \). Rewriting our approximation, we find that the sequence we are looking for satisfies

\[
y_{k+1} = y_k + T_s y_{k+1}, \quad y_0 = 1.
\]

Let us take \( T_s = 0.1 \).

The following code calculates the approximation. In this code \( y_k \) is the \( k + 1 \)th element of the vector \( y \). This is done because MATLAB does not allow a vector to start from element zero; all vectors start from element 1.

\[
y(1) = 1;
\]

for \( i = 2:11 \)
\[
y(i) = y(i-1)/(1 - 0.1);
\]

end

plot(0:0.1:1, y, 0:0.1:1, exp(0:0.1:1))

legend('The Approximation','The Exact Solution')

MATLAB responds to the commands with the plot of Figure 5.1. (The \texttt{legend} command causes MATLAB to add a legend to the plot. Use the MATLAB \texttt{help} command to find out more about the \texttt{legend} command.)

5.6 Converting a High-order ODE into a First-order ODE

Suppose that one would like to write the second order ODE

\[
y'' + y' + y = 0, \quad y(0) = 1, y'(0) = 0
\]

as a set of first order ODEs. One can proceed as follows. Define:

\[
\vec{x}(t) = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix}.
\]
Figure 5.1: A comparison of the exact and the approximate solutions of the differential equation.
One finds that:
\[
\frac{d}{dt} \vec{x}(t) = \begin{pmatrix} 0 & 1 \\ -1 & -1 \end{pmatrix} \vec{x} \quad \vec{x}(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.
\]

This equation has the general form
\[
\frac{d}{dt} \vec{x} = A \vec{x} \quad \vec{x}(0) = \vec{\alpha}
\]
and one should be able to approximate the solution to this equation by making use of the vector difference equation
\[
\vec{x}_{k+1} = \vec{x}_k + T_s A \vec{x}_{k+1}, \quad \vec{x}_0 = \vec{\alpha}.
\]

Solving for \(\vec{x}_{k+1}\), we find that
\[
(I - T_s A)\vec{x}_{k+1} = \vec{x}_k, \quad \vec{x}_0 = \vec{\alpha},
\]
or that
\[
\vec{x}_{k+1} = (I - T_s A)^{-1} \vec{x}_k, \quad \vec{x}_0 = \vec{\alpha}.
\]

The following MATLAB code implements this algorithm and compares the results of the algorithm with the exact solution as calculated using \texttt{dsolve}.

When calculating the approximate solution, we take \(T_s = 0.05\) s and we calculate the solution out to \(t = 20\) s.

```matlab
A = [0 1; -1 -1];
x_init = [1; 0]
Ts = 0.05;
N = 20/Ts;

UpdateMatrix = inv([1 0; 0 1] - A * Ts);

x = x_init;
y(1) = x(1);
for i = 1 : N
    x = UpdateMatrix * x;
y(i+1) = x(1);
end

syms z t
z = dsolve('D2z + Dz + z = 0','z(0) = 1, Dz(0) = 0')
```
CHAPTER 5. SOLVING DIFFERENTIAL EQUATIONS

Figure 5.2: A comparison of the exact and the approximate solutions of the differential equation.

time = [0 : N] * Ts;
z_eval = subs(z, t, time);

plot(time, y, '-', time, z_eval, '--')
legend('The numerical solution', 'The solution as provided by dsolve')

The code’s output is given in Figure 5.2

5.7 ODEs, Transfer Functions, and the Step Response

Let us use the knowledge we now have to calculate the step response of the block whose transfer function is

\[ G(s) = \frac{1}{\tau s + 1} \]

in two different ways. First, we solve the relevant equations analytically using the \texttt{dsolve} command. Then, we solve the equations numerically.
5.7.1 The Analytical Solution

Let \( Y(s) \) be the Laplace transform of the output of the system, and let \( X(s) \) be the Laplace transform of the input of the system. Then, we know that

\[
\frac{Y(s)}{X(s)} = \frac{1}{\tau s + 1}.
\]

Cross-multiplying, we find that

\[
\tau s Y(s) + Y(s) = X(s).
\]

Converting back to a differential equation, we find that

\[
\tau y'(t) + y(t) = x(t).
\]

Recall that when calculating the step response of a system we assume that all relevant initial conditions are zero and the input, \( x(t) \), is 1 for \( t \geq 0 \).

We find that in order to calculate the step response, we must solve the ODE

\[
\tau y'(t) + y(t) = 1
\]

subject to the initial condition \( y(0) = 0 \). Making use of the symbolic toolbox by giving MATLAB the commands

```matlab
syms tau y t
step = dsolve('tau * Dy + y = 1', 'y(0) = 0')
```

causes MATLAB to respond with

```matlab
step = 
1-exp(-1/tau*t)
```

5.8 The Numerical Solution

To solve this equation numerically, we make use of the approximation

\[
\frac{y_{k+1} - y_k}{T_s} \approx y'((k + 1)T_s).
\]

Making use of this approximation, we find that the ODE can be approximated by the difference equation

\[
\frac{\tau}{T_s}(y_{k+1} - y_k) + y_{k+1} = 1,
\]

\( y_0 = 0 \).

Translating this equation into MATLAB code and choosing \( T_s = 0.01 \) and \( \tau = 0.5 \), we find that giving MATLAB the commands
Figure 5.3: A comparison of the exact step response and the approximate step response

\[
\begin{align*}
tau &= 0.5 \\
Ts &= 0.01 \\
y(1) &= 0 \\
\text{for } i = 2:1000 \\
&\quad y(i) = \frac{tau}{tau+Ts} * y(i-1) + \frac{Ts}{Ts + tau}; \\
\text{end} \\
t &= [0:999] * Ts;
\end{align*}
\]

causes MATLAB to store the samples of the (approximate value of the ) of the step response in the array \( y \). Giving MATLAB the commands

\[
\begin{align*}
\text{plot}(t, y, t, (1 - \exp(-t/tau))) \\
\text{legend('Numerical Solution', 'Exact Solution')}
\end{align*}
\]

causes MATLAB to respond with Figure 5.3. We find that the approximation is quite accurate.
5.9 Exercises

1. Use the `dsolve` command to solve the differential equation $y''(t) = -y(t)$ subject to the initial conditions $y(0) = 1, y'(0) = 0$. Use the MATLAB `help` command (type `help dsolve`) if you need help with the commands syntax. Plot the solution using the `ezplot` command.

2. Use the `dsolve` command to solve the differential equation $y''(t) = -2y'(t) - y(t)$ subject to the initial conditions $y(0) = 1, y'(0) = 0$. Plot the solution using the `ezplot` command.

3. Define the matrix

$$A = \begin{pmatrix} 1 & 2 & 5 \\ 3 & 7 & 9 \\ 1 & 5 & 9 \end{pmatrix}$$

and the vector $\vec{v}$,

$$\vec{v} = \begin{bmatrix} 1 \\ 5 \\ 9 \end{bmatrix}$$

Calculate $A\vec{v}$, $A^2\vec{v}$, and $A^{-1}\vec{v}$.

4. Use MATLAB to approximate the solution of the equation $y''(t) = -y(t)$ subject to the initial conditions $y(0) = 1, y'(0) = 0$. Compare the approximate solution to the exact one found in exercise 1.

5. Use MATLAB to calculate the step response of the system whose transfer function is

$$G(s) = \frac{1}{s^2 + s + 1}$$

(a) Derive the differential equation satisfied by the step response.

(b) Use the `dsolve` command to solve the ODE.

(c) Next, approximate the solution of the ODE using the techniques we have studied.

(d) Compare the solutions of parts 5b and 5c.
Chapter 6

Fourier Series and the Gibbs Phenomenon

6.1 The Fourier Series of the Square Wave

Let the function $sq(t)$ be defined by

$$sq(t) = \begin{cases} 1 & m \leq t < m + 1/2 \\ -1 & m + 1/2 \leq t < m + 1 \end{cases}$$

The Fourier series associated with this function is

$$sq(t) = b_0 + \sum_{k=1}^{\infty} b_k \cos(2\pi kt) + \sum_{k=1}^{\infty} a_k \sin(2\pi kt).$$

The Fourier coefficients of this 1-periodic function are given by

$$b_0 = \int_{-1/2}^{1/2} sq(t) \, dt$$
$$b_k = 2 \int_{-1/2}^{1/2} \cos(2\pi kt) sq(t) \, dt, \quad k > 0$$
$$a_k = 2 \int_{-1/2}^{1/2} \sin(2\pi kt) sq(t) \, dt, \quad k \geq 0.$$

Because $sq(t)$ is odd, it is clear that $b_k = 0$ for all $k \geq 0$. From symmetry it is clear that:

$$a_k = 4 \int_{0}^{1/2} \sin(2\pi kt) \, dt = 4 \left. \frac{-\cos(2\pi kt)}{2\pi k} \right|_{t=0}^{t=1/2} = \begin{cases} \frac{4}{\pi k} & k \in \text{odd} \\ 0 & k \in \text{even} \end{cases}.$$
Thus, we find that the Fourier series associated with \( \text{sq}(t) \) is

\[
\sum_{k=0}^{\infty} \frac{4 \sin(2\pi(2k+1)t)}{\pi(2k+1)}.
\]

We would now like to examine the extent to which this series truly represents the square wave.

### 6.2 A Quick Check

Before proceeding to analyze the series we have found, it behooves us to check that the calculations were performed correctly. For a 1-periodic function like \( \text{sq}(t) \), Parseval’s equation states that

\[
\int_{-1/2}^{1/2} \text{sq}^2(t) \, dt = b_0^2 + \frac{1}{2} \sum_{k=1}^{\infty} (a_k^2 + b_k^2).
\]

In our case this means that

\[
1 = \frac{1}{2} \sum_{k=0}^{\infty} \left( \frac{4}{\pi(2k+1)} \right)^2 = \frac{8}{\pi^2} \sum_{k=0}^{\infty} \frac{1}{(2k+1)^2}.
\]

We can check that this sum is correct using MATLAB. One way to perform a quick check is to give MATLAB the commands

```matlab
L = [1:2:10001];
(8/pi^2) * sum(1./L.^2)
```

MATLAB responds with

```
ans =
1.0000
```

which is an indication that we may have performed all of the calculations correctly.

A second way to have MATLAB check this computation is to give MATLAB the commands

```matlab
syms k
(sym('8')/sym('pi')^2) * symsum(1/(2*k+1)^2,k,0,Inf)
```

MATLAB responds to these commands with

```
ans =
1
```

indicating that the series does, indeed, sum to 1.
6.3 “Seeing” the Sum

It is not difficult to have MATLAB calculate the Fourier series and display its values. Consider the following code:

```matlab
 t = [-500:500] * 0.001;
 total = zeros(size(t));
 for k = 1 : 2 : 101
     total = total + (4/pi) * sin(2*pi*k*t) / k;
 end
 plot(t,total)
```

This code defines a “time” vector, \( t \), with 1001 elements. It then defines a vector to hold the sum, \( \text{total} \), and proceeds to sum the first 51 terms in the Fourier series. The code causes MATLAB to produce the plot shown in Figure 6.1. The “ringing” produced by Gibb’s phenomenon is clearly visible.
6.4 The Experiment

Let

\[ \text{saw}(t) = t \]

for \(-1/2 < t \leq 1/2\) and continue \(\text{saw}(t)\) periodically outside of this region.

1. Calculate the Fourier coefficients associated with \(\text{saw}(t)\).

2. Check Parseval’s equation for \(\text{saw}(t)\) both numerically and symbolically.

3. Sum the first 3 terms in the Fourier series and plot the Fourier series as a function of time.

4. Sum the first 10 terms in the Fourier series and plot the Fourier series as a function of time.

5. Sum the first 50 terms in the Fourier series and plot the Fourier series as a function of time.

When summing the Fourier series, make sure that the time samples you take are sufficiently closely spaced that you clearly see Gibb’s phenomenon.
Chapter 7

Linearity and Nonlinearity

7.1 Linearity

Suppose that when one inputs $x_1$ to a system, the output of the system is $y_1$, and when one inputs $x_2$ to a system, the output of the system is $y_2$. A system is said to be linear if from these two facts one can conclude that if the input to the system is $\alpha_1 x_1 + \alpha_2 x_2$, then the output of the system will be $\alpha_1 y_1 + \alpha_2 y_2$. Linear systems are said to satisfy the principle of superposition.

7.2 Simulink

To “see” when a system is linear and when it is not, we make use of Simulink—a MATLAB “add-on.” Simulink is, essentially, a graphical user interface (GUI) to MATLAB. It allows one to drag-and-drop blocks to build up a system.

To open Simulink, one can type “simulink” at the MATLAB command prompt. Alternatively one can click on the Simulink icon on the toolbar in the main MATLAB window.

After performing either of these actions, the Simulink Library Browser will open. After it has opened, click on the blank page in this window’s toolbar to open a new Simulink “page.” To start working, one drags and drops items from the library browser into the worksheet, one connects the items, and then one runs the simulation.
7.3 A Simple Example

We start by building a system that amplifies a sine wave by a factor of two. To do this, go to the browser, click on the “sources” tab (in the Simulink “blockset”), and then drag a “sine wave” over to the untitled worksheet.

Having actually put something in the worksheet, it is probably best to save the worksheet; do so. Note that Simulink saves its worksheets with a .mdl extension. (Make sure to save worksheets regularly while working on them.)

Next, go to the “Math Operations” tab, click on it and drag a gain block to the worksheet. To connect the sine wave to the gain block select the sine wave block, hold down the control key and left-click on the gain block. (This is the general procedure for connecting blocks.)

Go to the “Sinks” tab, click on it, and drag a Scope block from the right-hand panel to the worksheet. Then connect the gain block to the scope. Double click on the gain block, and use the dialog box that opens to change the gain of the gain block from 1 to 2. Double click on the sine wave block, and change its frequency from 1 radian per second to $2\pi$ radians per second. (Enter $2 \times \pi$ as the frequency.) Double click on the scope to actually open a scope window. Finally hit the “play” button on top of the worksheet window, and a sine wave should appear in the scope window.

The sine wave may not be very pretty. Simulink is numerical software, and it is not always good at “guessing” how many samples of a function the user needs. In our case, in order to improve the quality of the sine wave, one can click on the sine wave block again and change the sample time from zero—which is “continuous-time”—to 0.001 s—which gives lots of samples in each period of the sine wave. Make the change and hit play again. How does the sine wave look now?

7.4 Testing Linearity

We now test the linearity of the gain block. To do this, build the system of Figure 7.1. The following tips should prove helpful.

- To copy a block, hold down the control key, left-click on the block and drag the copy to wherever it is needed.

- The summing block (the circle with the pluses inside) is located in the “Math Operations” library.

- Double clicking on the summing block opens up a dialog box that allows
Figure 7.1: A system to test the linearity of the gain block
one to change the sums to differences. That is how one produces a differencing block.

- To add a connection to a “wire,” hold down the control key, click on the spot on the wire to which you would like to add a connection, and drag the cursor to the input of the item to which you would like to connect. Release the cursor when a double cross-hair is shown over the block’s input.

- In order to open a scope window, double click on the scope of interest.

- If one does not see the whole signal on a scope, the problem is probably that the window is limiting the number of samples that it saves. To remove this restriction, go to the scope window. Go to the “parameters” tab (the second tab from the left) and click on it. A dialog box will open. Click on the “data history” tab, and unclick the “limit data points to last” box.

- Make the frequencies of the two sine waves different.

### 7.5 A Nonlinear System

Save the model that you have build for the linear system. Pick another block that should be nonlinear—perhaps the “sign” block in the “Math Operations” sub-library—and replace all the gains with this block. Run the simulation again. What is the output of the differencing block? How do your results show that the new block is not linear—that it is non-linear?

### 7.6 Exercise

1. Use the symbolic toolbox to show that the squaring operation is not linear. That is, use the symbolic toolbox to show that

\[(\alpha x + \beta y)^2 \neq \alpha x^2 + \beta y^2.\]

You may wish to use the commands pretty and collect to make the results of the symbolic calculations easier to read and understand. (Use the help command to find out how to use these new commands.)
Chapter 8

Continuous-time Linear Systems

8.1 Overview

One of the first, if not the first, MATLAB toolboxes was the control theory toolbox. By using the commands in this toolbox it is possible to define transfer function objects and to examine the properties of the systems described by the transfer functions.

8.2 Defining a Transfer Function Object

To define a transfer function object that corresponds to a transfer function that is rational function—a function that is the quotient of polynomials—one gives MATLAB the command \texttt{tf(num,den)}. The vector \texttt{num} contains the coefficients of the numerator polynomial and the vector \texttt{den} contains the coefficient of the denominator polynomial. The first element of each vector is the coefficient of the highest power in the polynomial, and each element afterwards corresponds to the next lower power of \(s\). Suppose that one has a system for which

\[ G(s) = \frac{1}{0.1s + 1}. \]

Giving MATLAB the command

\[
G = \text{tf([1],[0.1 1])}
\]

cause MATLAB to respond with

Transfer function:
8.3 Analyzing a Transfer Function

There are many ways that MATLAB can help analyze a transfer function. Supposing that we have defined the transfer function object $G$ as we did above. To examine $G$’s impulse response, one need only give MATLAB the command `impulse(G)`. To this command, MATLAB responds by producing Figure 8.1. (“Bonus question.” Why is the height of the response initially 10?) To examine the step response all one need do is give MATLAB the command `step(G)`. To this command, MATLAB responds by producing Figure 8.2. By using the magnifying glasses in the toolbar above the figures, it is possible to zoom in on parts of a figure or to zoom out from parts of a figure.

MATLAB makes it easy to plot Bode plots as well. The command `bode(G)` will cause MATLAB to produce the Bode plots that correspond
Figure 8.2: The step response of the system whose transfer function is $G(s) = \frac{1}{0.1s + 1}$
CHAPTER 8. CONTINUOUS-TIME LINEAR SYSTEMS

Figure 8.3: The block diagram of the system in which \( G(s) \) and \( H(s) \) are cascaded to the system. If all that one is interested in is the magnitude plot, MATLAB provides one with the command `bodemag`.

For more information about any given command, use the MATLAB `help` command.

8.4 Transfer Function Manipulations

Suppose that one is working with the system of Figure 8.3 where

\[
G(s) = \frac{1}{0.1s + 1} \quad \text{and} \quad H(s) = \frac{s}{0.1s + 1}.
\]

As we have already seen, \( G(s) \) is the transfer function of a low-pass filter. It is easy to see that \( H(s) \) is the transfer function of a high-pass filter. Giving MATLAB the command

\[
H = \text{tf}([1 \ 0], [0.1 \ 1])
\]

causes MATLAB to define the transfer function of the second block.

How will the system behave? If one cascades two blocks, then the transfer function of the cascaded blocks is the product of the transfer functions of the blocks. MATLAB knows how to multiply transfer functions, and giving MATLAB the command \( T = G \times H \), causes MATLAB to respond with

Transfer function:
\[
s
\]
\[
\frac{s}{0.01 s^2 + 0.2 s + 1}
\]

Giving MATLAB the command `bodemag(T)`, cause MATLAB to produce Figure 8.4. We find that the cascaded blocks lead to a bandpass filter.
Figure 8.4: The magnitude plot of the system in which $G(s)$ and $H(s)$ are cascaded
8.5 Systems with Feedback

Considering the system of Figure 8.5 and starting from $V_{\text{out}}(s)$, it is easy to see that:

$$V_{\text{out}}(s) = G(s)(V_{\text{in}}(s) - H(s)V_{\text{out}}(s)).$$

Rearranging terms, we find that

$$(1 + G(s)H(s))V_{\text{out}}(s) = G(s)V_{\text{in}}(s).$$

We find that

$$\frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = \frac{G(s)}{1 + G(s)H(s)}.\tag{1}$$

This is the transfer function of the system of Figure 8.5.

MATLAB is capable of computing the transfer function of the system with feedback from the transfer functions of its constituent parts. For example, if $G(s) = 1/(s + 1)$ and $H(s) = 2$, then giving MATLAB the commands

```matlab
G = tf([1],[1 1]);
H = tf([2],[1]);
T = G / (1 + G*H)
```

causes MATLAB to respond with

Transfer function:

$$\frac{s + 1}{s^2 + 4s + 3}$$
This is somewhat odd, as a simple calculation shows that the transfer function of the system with feedback is

\[ T(s) = \frac{G(s)}{1 + G(s)H(s)} = \frac{1}{s + 3}. \]

If one considers the answer provided by MATLAB somewhat more carefully, one finds that

\[ \frac{s + 1}{s^2 + 4s + 3} = \frac{s + 1}{(s + 1)(s + 3)} = \frac{1}{s + 3}. \]

That is, MATLAB found the correct answer but expressed it in an odd fashion. MATLAB took a first order transfer function and made it look like a second order transfer function.

### 8.6 Exercises

1. What kind of system is described by the transfer function

   \[ G(s) = \frac{1}{\tau s + 1}? \]

2. Analyze the system whose transfer function is

   \[ G(s) = \frac{s}{s^2 + s + (2\pi 100)^2}. \]

   (a) Define the transfer function object, \( G \).
   
   (b) Find the impulse response of the system.
   
   (c) Explain the frequency of the oscillations that you see.
   
   (d) Find the step response of the system.
   
   (e) Explain why the step response of the system starts from zero and ends at zero.

3. Analyze the system whose transfer function is

   \[ G(s) = \frac{-s + 1}{2s^2 + s + 3}. \]

   (a) Use the \texttt{ltiview} command to open the \texttt{ltiviewer}—the linear time-invariant system viewer. Use the \texttt{help} command to discover how this command is used.
(b) Have the ltiviewer display the step and impulse responses of the system.
(c) Explain why the step response of the system is initially negative.

4. Consider the system of Figure 8.5. Let $H(s) = 1$. Have MATLAB calculate the step response of the system described by the figure when:

(a) $G(s) = 0.125/(s^2 + s)$
(b) $G(s) = 0.25/(s^2 + s)$
(c) $G(s) = 0.5/(s^2 + s)$
(d) $G(s) = 1/(s^2 + s)$

What effect does the increased gain have on the system’s performance? (Please address both the system’s rise time and the amount of overshoot in the system’s output.)
Chapter 9

Non-minimum-phase Systems

9.1 Overview

Generally speaking, the transfer functions we see have both their poles and their zeros in the left half-plane. For causal systems, having a pole in the right half-plane is a *big* problem. Such a pole indicates that the system is not stable.

Systems with zeros in the right half-plane are said to be *non-minimum-phase systems*. What can one say about such systems?

One point that should be immediately obvious is that if $G(s)$ has a zero in the right half-plane, then $1/G(s)$ has a pole in the right half-plane and is not the transfer function of a stable system. This means that one cannot build a perfect open-loop compensator for $G(s)$. What else can one say about non-minimum-phase systems?

9.2 The Step Response of a Non-minimum-phase System

Suppose that one has a stable system whose transfer function is $G(s)$ and whose DC gain, $G(0)$, is positive. Suppose that in addition, $G(a) = 0$ for some $a > 0$; suppose that $G(s)$ has a zero in the right half-plane. Then we can show that at some point the step response of the system must go negative.

On an intuitive level, inputting a value of 1 to a system *should* cause the system to head towards the positive numbers. Assuming that a system’s initial conditions are zero, inputting a value of 1 should cause the system to head in the positive “direction.” We now show that this is not neces-
Non-minimum-phase systems behave in an anomalous fashion.

The output of the system whose transfer function is $G(s)$ to a unit step, $v_{\text{step}}(t)$, satisfies

$$V_{\text{step}}(s) = \frac{1}{s}G(s).$$

From the final value theorem, we know that

$$\lim_{t \to \infty} v_{\text{step}}(t) = \lim_{s \to 0^+} s \left( \frac{1}{s}G(s) \right) = \lim_{s \to 0^+} G(s) = G(0) > 0.$$

Thus, we know that the step response converges to a positive value.

We also know that

$$0 = \left. \frac{1}{s}G(s) \right|_{s=a} = \frac{G(a)}{a} = \int_{0}^{\infty} e^{-at}v_{\text{step}}(t) \, dt.$$

We know that for large enough $t$ the step response is greater than zero, and the exponential is always greater than zero. In order for the integral to equal zero, the step response must be negative at some point.

### 9.3 The Experiment: Part I

Define a transfer function object whose transfer function is given by

$$G(s) = \frac{-s/2 + 1}{s^2 + s/2 + 1}$$

Use the step command to plot the step response of the system. Does the step response agree with the predictions made by our theory?

### 9.4 Short-term Vs. Long-term Behavior

In §9.2, we found that if a stable system has a zero or zeros in the right half-plane, then the system’s step response must be negative for some positive value of $t$. In the example of §9.3, we found that the step response started off negative and then turned positive. Must this always be the case?
Let us consider transfer functions with a zero or zeros at \( a > 0 \), with all other zeros and poles in the left half-plane, and with positive DC gain. Such transfer functions must be of the form

\[
G(s) = (a - s)^m F(s)
\]

where \( F(s) \) has all of its poles and zeros in the left-half plane and \( F(0) > 0 \).

The initial value theorem states that if \( H(s) \) is the Laplace transform of \( h(t) \), then

\[
h(0^+) = \lim_{s \to \infty} sH(s).
\]

In particular, if the degree of the denominator of \( G(s) \) is greater than that of the numerator of \( G(s) \), then the initial value of the step response of the system described by \( G(s) \) will be

\[
\text{initial value} = \lim_{s \to \infty} \frac{1}{s} G(s) = 0.
\]

The Laplace transform of \( f'(t) \) is \( sF(s) - f(0^+) \). If \( f(0^+) \) is zero, then the Laplace transform of \( f'(t) \) is \( sF(s) \). Considering the step response again, we find that the Laplace transform of the derivative of the step response is \( G(s) \). Assuming that the difference in the degrees of the numerator is exactly one, we find that the initial value of the derivative is

\[
v'_{\text{step}}(0) = \lim_{s \to \infty} sG(s) = \lim_{s \to \infty} s(a - s)^m F(s).
\]

As the degrees of the numerator and the denominator of \( sG(s) \) are the same, the limit will be a non-zero number. As the sign of \( F(s) \) is positive for large value of \( s \), the sign of the number will be the same as that of \((a - s)^m\). If \( m \) is even, the sign will be positive, and if \( m \) is odd, the sign will be negative. That is, if \( m \) is even, the derivative of the step response is initially positive, and step response initially increases from 0, then turns negative, and eventually tends to \( G(0) \). If \( m \) is odd, then the step response decreases initially.

\section{9.5 The Experiment: Part II}

\subsection*{9.5.1 The Step Response}

Define a transfer function object that corresponds to a system whose transfer function is

\[
G(s) = \frac{(1 - s/2)^2}{s^3 + 6.25s^2 + 7s + 1}
\]

Have MATLAB plot the step response of the system. Does the plot agree with theory developed? Explain.
9.5.2 The Derivative of the Step Response

For the $G(s)$ given above, we find that the degree of the denominator is one greater than the degree of the numerator. As we have already seen, this implies that the step response of the system is zero when $t = 0$ and that the derivative of the step response is $G(s)$.

In order to understand the step response somewhat better, we would like to cause MATLAB to plot the derivative of the step response. To do this, we need to cause MATLAB to plot the inverse Laplace transform of $G(s)$. This, however, is precisely the impulse response of the system. As we saw in §8.3, MATLAB has a command, `impulse`, that plots the impulse response. Have MATLAB plot the derivative of the step response.

Now, explain the connection between the plot of the derivative of the step response and the plot of the step response. How does the first plot explain what we see in the second plot?

9.6 The Experiment: Part III

It is possible to arrive at the transfer function

$$G(s) = \frac{(1 - s/2)^2}{s^3 + 6.25s^2 + 7s + 1} \quad (9.1)$$

by considering a system like that of Figure 8.5 and taking

$$G(s) = \frac{(1 - s/2)^2}{s^3 + 6s^2 + 8s} \quad (9.2)$$

and $H(s) = 1$.

Define the transfer function object $G$ to have the transfer function given by (9.2). Let $T$ be defined by $T = G/(1+G)$. Note that the resulting transfer function does not seem to match that of (9.1). Actually, MATLAB has simply “inflated” the transfer function (as it did in §8.5).

Give MATLAB the command `step(T)`. Examine the region near $t = 0$ carefully. Does the step response of the system rise above zero before dipping below zero? Try to explain why the step response as given by MATLAB does not quite match the step response it gave in the previous section. This example shows that sometimes using the correct form for a function can be important to numerical methods used to evaluate the function.
9.7 Exercises

1. Explain why if the denominator of a function, $G(s)$, is of higher degree than the numerator of $G(s)$, then

$$\lim_{s \to \infty} G(s) = 0.$$ 

2. Let $u(t)$ be the unit step function. Let $g(t)$ be a system’s impulse response, and let $G(s)$, the Laplace transform of $g(t)$, be the system’s transfer function. By definition, the step response of a system is the output of the system when the input is $u(t)$. For our system,

$$v_{\text{step}}(t) = \int_{-\infty}^{\infty} u(t - \tau) g(\tau) \, d\tau.$$ 

Differentiate this function, and show that $v'_{\text{step}}(t) = g(t)$. (You may assume that any calculation that looks “legal” is.)
Chapter 10

Analog Filter Design Using MATLAB

10.1 Overview

MATLAB provides many filter design tools. Most of the tools are aimed at
digital filter design, but some of the tools also support analog filter design.
We briefly consider some analog filter design tools.

10.2 Designing a Butterworth Filter

To design an analog low-pass Butterworth filter using MATLAB, one uses
the command

\[ [b \ a] = \text{butter}(\text{order}, \ \text{cutoff\_freq}, \ 's') \]

This command tells MATLAB to design a low-pass Butterworth filter of
order \text{order} and cutoff frequency \text{cutoff\_freq}. The \text{'s'} tells MATLAB to
design an analog filter. (Without this command, MATLAB designs a digital
filter.) The vectors \text{a} and \text{b} hold the coefficients of the denominator and the
numerator (respectively) of the filter’s transfer function. Giving MATLAB
the commands

\[ [b \ a] = \text{butter}(4, 100, \ 's'); \]
\[ G = \text{tf}(b,a) \]

causes MATLAB to respond with
Figure 10.1: The magnitude plot of the fourth-order low-pass Butterworth filter with a cutoff frequency of 100 rad/s.

Transfer function:

\[
\frac{1e008}{s^4 + 261.3 s^3 + 3.414e004 s^2 + 2.613e006 s + 1e008}
\]

Giving MATLAB the command \texttt{bodemag(G,\{30,3000\})} causes MATLAB to respond with Figure 10.1. (The term \{30,3000\} in the \texttt{bodemag} command causes the command to plot the magnitude response from 30 rad/s out to 3,000 rad/s.)

Two points are worth noting. Looking at the magnitude plot, one sees that at 100 rad/s the response seems to have decreased by about 3 dB—as it should. Also, one notes that from 100 rad/s to 1,000 rad/s the response seems to drop by about 80 dB. As this is a fourth order filter its rolloff should be \(4 \times 20\) dB/dec.
10.3 Exercises

1. Design a fourth order high-pass Butterworth filter whose cutoff frequency is 1,000 rad/s.
   (a) What is the transfer function of the filter?
   (b) Plot the magnitude response of the filter.
   (c) What is the high-frequency rolloff?

2. Compare the performance of a fourth-order low-pass Chebyshev filter with a cutoff frequency of 1 kHz with the performance of Butterworth filter of the same order.
   (a) Use the MATLAB help command to learn how to use the cheby2 command.
   (b) Use the commands butter and cheby2 to design the two filters.
   (c) Plot the magnitude plots of the two filters.
   (d) Which of the filters has a narrower transition region?
   (e) Which of the filters has a “prettier” magnitude response?
Chapter 11

Using MATLAB to Calculate Transforms

11.1 The Fourier Transform

In principle, calculating Fourier transforms using the symbolic toolbox should not be too hard. In practice it seems to require a fair amount of thought and effort.

Consider the Fourier transform of a pulse—of a function of the form:

\[ \Pi_a(t) = \begin{cases} 1 & |t| \leq a/2 \\ 0 & \text{otherwise} \end{cases} \]

In principle, calculating the transform should be simple. All that one needs to do is to calculate the integral of \( e^{-2\pi jft} \Pi_a(t) \) with respect to \( t \). In practice, one must first define the pulse function.

One way to define the function is to give MATLAB the commands

```matlab
syms t a
sgn = t/abs(t)
stp = (sym('1') + sgn)/sym('2')
pulse = subs(stp,t+a/sym('2')) - subs(stp,t-a/sym('2'))
```

MATLAB responds with

```
pulse =
1/2*(t+1/2*a)/abs(t+1/2*a)-1/2*(t-1/2*a)/abs(-t+1/2*a)
```

If one would like to see this in a more human-readable form, one can give MATLAB the command `pretty(pulse)`. MATLAB responds with
The symbol pulse is just what we need. To calculate the Fourier transform of $\Pi_1(t)$, we give MATLAB the commands

\[
j = \text{sqrt}('\text{sym}('-1'))
pulse\_trans = \text{int}(\exp(-2\pi j f t) * \text{subs}(\text{pulse},a,1), t, -\infty, \infty)
\]

MATLAB responds with

\[
\frac{-1/2 i (-\exp(-i \pi f) + \exp(i \pi f))}{\pi f}
\]

To simplify this expression and present it in a more human-readable form, one can use the command \texttt{pretty(simple(pulse\_trans))}. The \texttt{simple} command looks for a simple form of \texttt{pulse\_trans}, and, as we have seen, the \texttt{pretty} command presents the results in a more human-readable form. The results MATLAB gives are

\[
\frac{\sin(\pi f)}{\pi f}
\]

This is both a pretty and a simple form of the transform.

To plot this function one could use the \texttt{ezplot} command. For some reason, this does not seem to give optimal results. It is also possible to give the commands

\[
t = -20.005:0.01:20.005;
out = \text{subs}(\text{pulse\_trans},t);
\text{plot}(t, out, t, \text{zeros(size(t)))}
\text{axis tight}
\]

These commands cause MATLAB to respond by producing Figure 11.1. (The reason for the somewhat odd-looking definition of $t$—the set of points at which the Fourier transform is to be evaluated—is to avoid having any element of $t$ equal to zero. Evaluating

\[
\frac{\sin(\pi f)}{\pi f}
\]

at $f = 0$ causes MATLAB to generate a divide-by-zero error.)

The command \texttt{axis tight} causes MATLAB to make the border of the figure fit the plotted values “tightly.” The command \texttt{zeros(size(obj))}
11.2. Z-TRANSFORMS

Calculating the Z-transform can be done by using the `symsum` command. This command calculates sums of symbolic random variables and is very similar to the `int` command.

Let us calculate the Z-transform of the sequence

\[ a_k = \begin{cases} 
  kT_s & k \geq 0 \\
  0 & \text{otherwise}
\end{cases} \]

To do this and cause MATLAB to output the results in an eye-pleasing fashion, we give MATLAB the commands

```
syms k Ts z
```
trans = symsum(Ts * k * z^(-k),k,0,inf);
pretty(trans)
MATLAB responds with

\[
\frac{Ts \cdot z}{2(-1 + z)}
\]

11.3 Exercises

1. Show that the expression

\[
\frac{t + 1/2 \cdot a}{1/2} - \frac{t - 1/2 \cdot a}{1/2} - \left| t + 1/2 \cdot a \right| - \left| -t + 1/2 \cdot a \right|
\]

is, in fact, equal to \( \Pi_a(t) \). You may want to examine the value of the expression when \( t < -a/2 \), when \( -a/2 \leq t < a/2 \), and when \( t \geq a/2 \).

2. Make use of the symbolic toolbox to calculate the Fourier transform of

\[
g(t) = \begin{cases} 
e^{-t} & t \geq 0 \\
0 & \text{otherwise} \end{cases}
\]

3. Make use of the symbolic toolbox to calculate the Fourier transform of

\[
g(t) = e^{-|t|}.
\]

It may be necessary to calculate the Fourier transform of \( g(t) \) by calculating the integral from \(-\infty\) to 0 and from 0 to \( \infty \) separately. Then, of course, one must add the two results to get the final result.

4. Make use of the techniques of §11.1 to calculate and plot the Fourier transform of \( \Pi_1(t) \cos(2\pi100t) \).

5. Make use of the symbolic toolbox to calculate the Z-transform of the sequences:

(a) \( a_k = \begin{cases} 
k^2 & k \geq 0 \\
0 & \text{otherwise} \end{cases} \)

(b) \( a_k = b^{-|k|} \)
Bibliography

