Anywhere–Anytime
Signals and Systems Laboratory
From MATLAB to Smartphones
Synthesis Lectures on Signal Processing

Editor
José Moura, Carnegie Mellon University

Synthesis Lectures in Signal Processing publishes 80- to 150-page books on topics of interest to signal processing engineers and researchers. The Lectures exploit in detail a focused topic. They can be at different levels of exposition—from a basic introductory tutorial to an advanced monograph—depending on the subject and the goals of the author. Over time, the Lectures will provide a comprehensive treatment of signal processing. Because of its format, the Lectures will also provide current coverage of signal processing, and existing Lectures will be updated by authors when justified.

Lectures in Signal Processing are open to all relevant areas in signal processing. They will cover theory and theoretical methods, algorithms, performance analysis, and applications. Some Lectures will provide a new look at a well established area or problem, while others will venture into a brand new topic in signal processing. By careful reviewing the manuscripts we will strive for quality both in the Lectures’ contents and exposition.

Anywhere-Anytime Signals and Systems Laboratory: From MATLAB to Smartphones
Nasser Kehtarnavaz and Fatemeh Saki
2016

Smartphone-Based Real-Time Digital Signal Processing
Nasser Kehtarnavaz, Shane Parris, and Abhishek Sehgal
2015

An Introduction to Kalman Filtering with MATLAB Examples
Narayan Kovvali, Mahesh Banavar, and Andreas Spanias
2013

Sequential Monte Carlo Methods for Nonlinear Discrete-Time Filtering
Marcelo G.S. Bruno
2013

Processing of Seismic Reflection Data Using MATLAB™
Wail A. Mousa and Abdullatif A. Al-Shuhail
2011
Fixed-Point Signal Processing
Wayne T. Padgett and David V. Anderson
2009

Advanced Radar Detection Schemes Under Mismatched Signal Models
Francesco Bandiera, Danilo Orlando, and Giuseppe Ricci
2009

DSP for MATLAB™ and LabVIEW™ IV: LMS Adaptive Filtering
Forester W. Isen
2009

DSP for MATLAB™ and LabVIEW™ III: Digital Filter Design
Forester W. Isen
2008

DSP for MATLAB™ and LabVIEW™ II: Discrete Frequency Transforms
Forester W. Isen
2008

DSP for MATLAB™ and LabVIEW™ I: Fundamentals of Discrete Signal Processing
Forester W. Isen
2008

The Theory of Linear Prediction
P. P. Vaidyanathan
2007

Nonlinear Source Separation
Luis B. Almeida
2006

Spectral Analysis of Signals: The Missing Data Case
Yanwei Wang, Jian Li, and Petre Stoica
2006
Anywhere-Anytime
Signals and Systems Laboratory
From MATLAB to Smartphones

Nasser Kehtarnavaz and Fatemeh Saki
University of Texas at Dallas

SYNTHESIS LECTURES ON SIGNAL PROCESSING #14
ABSTRACT
A typical undergraduate electrical engineering curriculum incorporates a signals and systems course. The widely used approach for the laboratory component of such courses involves the utilization of MATLAB to implement signals and systems concepts. This book presents a newly developed laboratory paradigm where MATLAB codes are made to run on smartphones, which most students already possess. This smartphone-based approach enables an anywhere-anytime platform for students to conduct signals and systems experiments. This book covers the laboratory experiments that are normally covered in signals and systems courses and discusses how to run MATLAB codes for these experiments on smartphones, thus enabling a truly mobile laboratory environment for students to learn the implementation aspects of signals and systems concepts. A zipped file of the codes discussed in the book can be acquired via the website http://sites.fastspring.com/bookcodes/product/SignalsSystemsBookcodes.

KEYWORDS
smartphone-based signals and systems laboratory; anywhere-anytime platform for signals and system courses; from MATLAB to smartphones
# Contents

Preface ................................................................. xi

1 Introduction to MATLAB .............................................. 1
  1.1 Starting MATLAB .............................................. 1
    1.1.1 Arithmetic Operations .................................... 1
    1.1.2 Vector Operations ......................................... 5
    1.1.3 Complex Numbers .......................................... 6
    1.1.4 Array Indexing ............................................ 7
    1.1.5 Allocating Memory .......................................... 8
    1.1.6 Special Characters and Functions ....................... 9
    1.1.7 Control Flow .............................................. 10
    1.1.8 Programming in MATLAB .................................. 12
    1.1.9 Sound Generation ......................................... 13
    1.1.10 Loading and Saving Data .................................. 14
    1.1.11 Reading Wave and Image Files ......................... 14
    1.1.12 Signal Display .......................................... 15
  1.2 MATLAB Programming Examples ................................. 15
    1.2.1 Signal Generation ......................................... 15
    1.2.2 Generating a Periodic Signal ............................. 17
  1.3 Lab Exercises ................................................ 18

2 Android Software Development Tools .............................. 21
  2.1 Installation Steps ............................................. 21
    2.1.1 Java JDK .................................................. 21
    2.1.2 Android Studio Bundle and Native Development Kit .... 22
    2.1.3 Environment Variable Configuration ..................... 23
    2.1.4 Android Studio Configuration ............................ 24
    2.1.5 Android Emulator Configuration .......................... 27
  2.2 Getting Familiar with Android Software Tools ............... 31
## From MATLAB Coder to Smartphone

3.1 MATLAB Function Design .................................. 43
3.2 Generating Signals via MATLAB on Smartphones ............... 45
  3.2.1 Test Bench ................................................. 46
  3.2.2 C Code Generation ...................................... 46
  3.2.3 Source Code Integration ................................. 48
3.3 Running MATLAB Coder C Codes on Smartphones ............... 52
3.4 References .................................................. 56

## Linear Time–Invariant Systems and Convolution

4.1 Convolution and Its Numerical Approximation .................. 57
4.2 Convolution Properties ....................................... 59
4.3 Convolution Experiments ...................................... 59
4.4 Lab Exercises ................................................ 89
  4.4.1 Echo Cancellation ........................................ 89
  4.4.2 Noise Reduction Using Mean Filtering .................... 90
  4.4.3 Impulse Noise Reduction Using Median Filtering ........ 91
4.5 Running MATLAB Coder C Codes on Smartphone ............... 91
4.6 Running in Real-Time on Smartphones .......................... 95
  4.6.1 MATLAB Function Design ................................. 95
  4.6.2 Test Bench ................................................ 95
  4.6.3 Modifying a Real Time Shell ............................. 99
4.7 References .................................................. 105

## Fourier Series

5.1 Fourier Series Numerical Computation .......................... 108
5.2 Fourier Series and Its Applications ............................. 109
5.3 Lab Exercises ................................................ 123
  5.3.1 RL Circuit Analysis ...................................... 123
  5.3.2 The Doppler Effect ...................................... 124
  5.3.3 Synthesis of Electronic Music ............................ 124
5.4 References .................................................. 127

## Continuous-Time Fourier Transform

6.1 CTFT and Its Properties ..................................... 129
6.2 Numerical Approximations of CTFT ............................ 130
Preface

A typical undergraduate electrical engineering curriculum incorporates a signals and systems course where students normally first encounter signal processing concepts of convolution, Fourier series, Fourier transform, and discrete Fourier transform. For the laboratory component of such courses, the conventional approach has involved a laboratory environment consisting of computers running MATLAB codes. There exist a number of lab textbooks or manuals for the laboratory component of signals and systems courses based on MATLAB, e.g., An Interactive Approach to Signals and Systems Laboratory by Kehtarnavaz, Loizou, and Rahman; Signals and Systems Laboratory with MATLAB by Palamides and Veloni; Signals and Systems: A Primer with MATLAB by Sadiku and Ali; and Signals and Systems by Mitra.

The motivation for writing this lab textbook or manual has been to provide an alternative laboratory approach to the above conventional laboratory approach by using smartphones as a truly mobile anywhere-anytime platform for students to run signals and systems codes written in MATLAB. This approach eases the requirement of using a dedicated laboratory room for signals and systems courses and would allow students to use their own laptops and smartphones as the laboratory platform to learn the implementation aspects of signals and systems concepts.

The challenge in developing this alternative approach has been to limit the programming language required from students to MATLAB and not require them to know any other programming language. MATLAB is extensively used in engineering departments and students are often expected to use it for various courses they take during their undergraduate studies.

The above challenge is met here by using the smartphone software tools that are publicly available. The software development environments of smartphones (both Android and iPhone) are free of charge and students can download and place them on their own laptops to be able to run signals and systems algorithms written in MATLAB on their own smartphones. In this book, we have developed the software shells that allow students to take MATLAB codes written on a laptop and run them on their own smartphones as apps. This initial edition of the book uses Android smartphones, however, it is to be noted that the same approach can be performed on iPhone smartphones as discussed in another book entitled Smartphone-based Real-time Digital Signal Processing by Kehtarnavaz, Parris, and Sehgal.

The book chapters correspond to the following labs for a semester-long lab course, considering that labs 4 through 7 often constitute the laboratory component of signals and systems courses: (1) introduction to MATLAB programming; (2) smartphone development tools; (3) use of MATLAB Coder to generate C codes from MATLAB and running C codes on smartphones; (4) linear time-invariant systems and convolution; (5) Fourier series; (6) continuous-time Fourier transform; and (7) digital signals and discrete Fourier transform.
Note that a zipped file of the codes discussed in the book can be acquired from the website http://sites.fastspring.com/bookcodes/product/SignalsSystemsBookcodes. Also, it is worth stating that this book is only meant as an accompanying lab book to signals and systems textbooks and is not meant to be used as a substitute for these textbooks.

As a final note, we wish to express our appreciation to the Erik Jonsson School of Engineering and Computer Science at the University of Texas at Dallas for supporting the development of this laboratory book.

Nasser Kehtarnavaz and Fatemeh Saki
August 2016
Chapter 1

Introduction to MATLAB

MATLAB is a programming environment that is widely used to solve engineering problems. There are many online references on MATLAB that one can read to become familiar with this programming environment. This chapter is only meant to provide an overview or a brief introduction to MATLAB.

Screenshots are used to show the steps to be taken and configuration options to set when using the Windows operating system.

1.1 STARTING MATLAB

Assuming MATLAB is installed on the laptop or computer used, select MATLAB from the Start bar of Windows, as illustrated in Figure 1.1. After starting MATLAB, a window called MATLAB desktop appears, see Figure 1.2, which contains other subwindows or panels. The panel Command Window allows interactive computation to be conducted. Suppose it is desired to compute \(3 + 4 \times 6\). This is done by typing it at the prompt command denoted by `>>`, see Figure 1.3. Since no output variable is specified for the result of \(3 + 4 \times 6\), MATLAB returns the value in the variable \(\texttt{ans}\), which is created by MATLAB. Note that \(\texttt{ans}\) is always overwritten by MATLAB, so, if the result is used for another operation, it needs to be assigned to a variable, for example \(x = 3 + 4 \times 6\).

In practice, a sequence of operations is usually performed to achieve a desired output. Often, a so-called m-file script is used for this purpose. An m-file script is a simple text file where MATLAB commands are listed. Figure 1.4 shows how to start a new script. In the HOME menu, locate the New Script tab or locate it under New → Script, or Ctrl+N to create a blank script under the panel EDITOR. When a new script is opened, it looks as shown in Figure 1.5. A script can be saved using a specified name in a desired location. An m-file script needs to be saved with “.m” extension. When such a file is run, MATLAB reads the commands and executes them as though there were typed sequentially. The following section provides further details on the MATLAB commands and operations.

1.1.1 ARITHMETIC OPERATIONS

There are four basic arithmetic operators in m-files:
Figure 1.1: MATLAB appearance in the Windows Start bar.
1.1. STARTING MATLAB

**Command History:**
View or run previously executed functions.

**Command Window:**
Write MATLAB functions at prompt command line (>>).

**Current Directory:**
You can change it to any desired one.

**Workspace:**
View variables that are created and stored during a MATLAB session.

**Current Folder:**
View the files in the current directory.

**Figure 1.2:** MATLAB interface window.

**Figure 1.3:** A simple computation in the Command Window.
4 1. INTRODUCTION TO MATLAB

Figure 1.4: Starting a new m-file script in MATLAB.

Figure 1.5: An m-file script docked in the EDITOR panel.
1.1. STARTING MATLAB

1.1.1 ADDITION, SUBTRACTION, MULTIPLICATION, AND DIVISION

In MATLAB, the following operators are used for basic arithmetic operations:

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division (for matrices, it also means inversion)

The following three operators work on an element-by-element basis:

- `.*` multiplication of two vectors, element-wise
- `./` division of two vectors, element-wise
- `.^` raising all the elements of a vector to a power

As an example, to evaluate the expression \( a^3 + \sqrt{bd} - 4c \), where \( a = 1.2, b = 2.3, c = 4.5, \) and \( d = 4 \), type the following commands in the Command Window to get the answer (`ans`):

```matlab
>> a=1.2;
>> b=2.3;
>> c=4.5;
>> d=4;
>> a^3+sqrt(b*d)-4*c
ans =
-13.2388
```

Note the semicolon after each variable assignment. If the semicolon is omitted, the interpreter echoes back the variable value.

1.1.2 VECTOR OPERATIONS

Consider the vectors \( \mathbf{x} = [x_1, x_2, \ldots, x_n] \) and \( \mathbf{y} = [y_1, y_2, \ldots, y_n] \). The following operations indicate the resulting vectors:

\[
\mathbf{x} \cdot \mathbf{y} = [x_1y_1, x_2y_2, \ldots, x_ny_n]
\]
\[
\mathbf{x} / \mathbf{y} = \left[ \frac{x_1}{y_1}, \frac{x_2}{y_2}, \ldots, \frac{x_n}{y_n} \right]
\]
\[
\mathbf{x} \wedge \mathbf{p} = [x_1^p, x_2^p, \ldots, x_n^p]
\]

Considering that the boldfacing of vectors/matrices are not used in .m files, in the notation adopted in this book, no boldfacing of vectors/matrices is shown to retain notation consistency with .m files.
6  1. INTRODUCTION TO MATLAB

The arithmetic operators + and − can be used to add or subtract matrices, vectors, or scalars. Vectors denote one-dimensional arrays and matrices denote multi-dimensional arrays. For example:

\[
\begin{align*}
>> \text{x} &= [1,3,4] \\
>> \text{y} &= [4,5,6] \\
>> \text{x}+\text{y} \\
\text{ans} &= \\
5 & \text{ 8 10}
\end{align*}
\]

In this example, the operator + adds the elements of the vectors x and y, element by element, assuming that the two vectors have the same dimension, in this case, or 1 × 3 one row with three columns. An error occurs if one attempts to add vectors having different dimensions. The same applies for matrices.

To compute the dot product of two vectors (in other words, \( \sum_i x_i y_i \)), use the multiplication operator "*" as follows:

\[
\begin{align*}
>> \text{x}*\text{y}' \\
\text{ans} &= \\
4 & 15 24
\end{align*}
\]

Note the single quote after y denotes the transpose of a vector or a matrix.

An element-by-element multiplication of two vectors (or two arrays) is computed by the following operator:

\[
\begin{align*}
>> \text{x}.*\text{y} \\
\text{ans} &= \\
4 & 15 24
\end{align*}
\]

That is, \( x.*y \) means \([1 \times 4, 3 \times 5, 4 \times 6] = [4 \text{ 15 24}]\).

1.1.3 COMPLEX NUMBERS

MATLAB supports complex numbers. The imaginary number is denoted with the symbol \( i \) or \( j \), assuming that these symbols have not been used any other place in the program. It is critical to avoid such symbol conflicts for obtaining correct outcome. Enter the following and observe the outcomes:
1.1. STARTING MATLAB

```matlab
>> z = 3 + 4i \ % note the multiplication sign '.*' is not needed after 4
>> conj(z) \ % computes the conjugate of z
>> angle(z) \ % computes the phase of z
>> real(z) \ % computes the real part of z
>> imag(z) \ % computes the imaginary part of z
>> abs(z) \ % computes the magnitude of z
```

One can also define an imaginary number with any other user-specified variables. For example, in the following manner:

```matlab
>> img = sqrt(-1)
>> z = 3 + 4*img
>> exp(pi*img)
```

1.1.4 ARRAY INDEXING

In m-files, all arrays (vectors) are indexed starting from 1; in other words, x(1) denotes the first element of the array x. Note that arrays are indexed using parentheses (.) and not square brackets [.] , as done in C/C++. To create an array featuring the integers 1 through 6 as elements, enter:

```matlab
>> x = [1, 2, 3, 4, 5, 6]
```

Alternatively, use the notation `:`:

```matlab
>> x = 1:6
```

This notation creates a vector starting from 1 to 6, in steps of 1. If a vector from 1 to 6 in steps of 2 is desired, then type:

```matlab
>> x = 1:2:6
ans =
    1    3    5
```

Also, examine the following code:
One can easily extract numbers in a vector. To concatenate an array, the example below shows how to use the operator "[ ]":

>> x=[1:3 4 6 100:110]

To access a subset of this array, type the following:

>> x(3:7)
>> length(x) % gives the size of the array or vector
>> x(2:2:length(x))

### 1.1.5 ALLOCATING MEMORY

One can allocate memory for one-dimensional arrays (vectors) using the command or function `zeros`. The following command allocates memory for a 100-dimensional array:

```matlab
>> y=zeros(100,1);
>> y(30)
ans =
 0
```

`zeros(n,m)` creates an \( n \times m \) matrix with all 0 elements. One can allocate memory for two-dimensional arrays (matrices) in a similar fashion. The command or function

```matlab
>> y=zeros(4,5)
```

defines a 4 by 5 matrix.

Similar to the command `zeros`, the command `ones` can be used to define a vector containing all ones,
1.1. STARTING MATLAB

```matlab
>> y=ones(1,5)
ans= 
1 1 1 1 1
```

1.1.6 SPECIAL CHARACTERS AND FUNCTIONS

Some common special characters used in m-files are listed in Table 1.1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pi</code></td>
<td>π (3.14.....)</td>
</tr>
<tr>
<td><code>^</code></td>
<td>indicates power (for example, 3^2 = 9)</td>
</tr>
<tr>
<td><code>NaN</code></td>
<td>not-a-number, obtained when encountering undefined operations, such as 0/0</td>
</tr>
<tr>
<td><code>Inf</code></td>
<td>represents +∞; indicates the end of a row in a matrix; also used to suppress printing on the screen (echo off)</td>
</tr>
<tr>
<td><code>%</code></td>
<td>comments anything to the right of % is ignored by the .m file interpreter and is considered to be comments</td>
</tr>
<tr>
<td><code>'</code></td>
<td>denotes transpose of a vector or a matrix; also used to define strings, for example, <code>str1 = 'DSP'</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td>denotes continuation; three or more periods at the end of a line continue current function to next line</td>
</tr>
</tbody>
</table>

Some special functions that are widely used are listed in Table 1.2.

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sqrt</code></td>
<td>indicates square root, for example, <code>sqrt(4) = 2</code></td>
</tr>
<tr>
<td><code>abs</code></td>
<td>absolute value</td>
</tr>
<tr>
<td><code>length</code></td>
<td><code>length(x)</code> gives the dimension of the array x</td>
</tr>
<tr>
<td><code>sum</code></td>
<td>finds sum of the elements of a vector</td>
</tr>
<tr>
<td><code>find</code></td>
<td>finds indices of nonzero</td>
</tr>
</tbody>
</table>

Here is an example of the function `length`,
1. INTRODUCTION TO MATLAB

>> x=1:10;
>> length(x)
ans =
    10

The function `find` returns the indices of a vector that are non-zero. For example,
`I = find(x>4)` finds all the indices of x greater than 4. Thus, for the above example:

>> find(x>4)
ans =
      5 6 7 8 9 10

1.1.7 CONTROL FLOW

m-files have the following control flow constructs:

• `if` statements
• `switch` statements
• `for` loops
• `while` loops
• `break` statements

The constructs `if`, `for`, `switch`, and `while` need to terminate with an `end` statement. Examples are provided below:

if

>> x=-3;
if x>0
    str='positive'
elseif x<0
    str='negative'
elseif x== 0
    str='zero'
else

1.1. STARTING MATLAB

```
str='error'
end
```

See the value of "str" after running the above code.

```
while
  x=-10;
  while x<0
    x=x+1;
  end
```

See the value of x after running the above code.

```
for loop
  >>x=0;
  for j=1:10
    x=x+j;
  end
```

The above code computes the sum of all the numbers from 1 to 10.

```
break
  With the break statement, one can exit early from a for or a while loop. For example,
  >> x=-10;
  while x<0
    x=x+2;
    if x == -2
      break;
    end
  end
```

Some of the supported relational and logical operators are listed below.
12 1. INTRODUCTION TO MATLAB

Relational Operators

Table 1.3: Relational operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=</td>
<td>less than equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>==</td>
<td>equal</td>
</tr>
<tr>
<td>~=</td>
<td>not equal</td>
</tr>
</tbody>
</table>

Logical Operators

Table 1.4: Logical operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>NOT</td>
</tr>
</tbody>
</table>

1.1.8 PROGRAMMING IN MATLAB

Open a new script file as displayed in Figures 1.4 and 1.5. Save it first in a desired directory. Then write your MATLAB code and press Run button from the EDITOR panel.

For instance, to write a program to compute the average (mean) of a vector x, the program should use the vector x as its input and return the average value. To write this program, follow the steps outlined below.

Type the following in an empty window:

```matlab
x=1:10
L=length(x);
sum=0;
for j=1:L
    sum=sum+x(j);
end
y=sum/L % the average of x
```
From the **EDITOR** panel, go to *save* → *Save As* and enter *average.m* for the filename. Then click on the **Run** button to run the program. Figure 1.6 shows the MATLAB interactive window after running the program.

![MATLAB Interactive Window](image)

**Figure 1.6:** m-file script interactive window after running the program *average*.

### 1.1.9 SOUND GENERATION

Assuming the computer used has a sound card, one can use the function `sound` to play back speech or audio files through its speakers. That is, the function `sound(y,FS)` sends the signal in a vector `y` (with sample frequency `FS`) out to the speaker. Stereo sounds are played on platforms that support them, with `y` being an N-by-2 matrix.

Type the following code and listen to a 400 Hz tone:
14 1. INTRODUCTION TO MATLAB

```
>> t=0:1/8000:1;
>> x=cos(2*pi*400*t);
>> sound(x,8000);
```

Now generate a noise signal by typing:

```
>> noise=randn(1,8000);  % generate 8000 samples of noise
>> sound(noise,8000);
```

The function `randn` generates Gaussian noise with zero mean and unit variance.

1.1.10 LOADING AND SAVING DATA

One can load or store data using the commands `load` and `save`. To save the vector `x` of the above code in the file `data.mat`, type:

```
>> save('data.mat', 'x')
```

To retrieve the data saved, type:

```
>> load data
```

The vector `x` gets loaded in memory. To see memory contents, use the command `whos`:

```
>> whos
Variable            Dimension     Type
x                    1x8000        double array
```

The command `whos` gives a list of all the variables currently in memory, along with their dimensions and data type. In the above example, `x` contains 8,000 samples.

To clear up memory after loading a file, type `clear all` when done. This is important because, if one does not clear all the variables, conflicts can occur with other codes using the same variables.

1.1.11 READING WAVE AND IMAGE FILES

With MATLAB, one can read data from different file types (such as .wav, .jpeg, and .bmp) and load them in a vector.
1.2. MATLAB PROGRAMMING EXAMPLES

To read an audio data file with .wav extension, use the following command:

```matlab
>> [y,Fs]=audioread('filename')
```

This command reads a wave file specified by the string `filename` and returns the sampled data in `y` with the sampling rate of `Fs` (in hertz).

To read an image file, use the following command:

```matlab
>> [y]=imread('filename')
```

This command reads a grayscale or color image from the string `filename` and returns the image data in the array `y`.

1.1.12 SIGNAL DISPLAY

Graphical tools are available in MATLAB to display data in a graphical form. Throughout the book, signals in both the time and frequency domains are displayed using the function `plot`,

```matlab
>> plot(x,y)
```

This function creates a 2-D line plot of the data in `y` versus corresponding `x` values.

1.2 MATLAB PROGRAMMING EXAMPLES

In this section, several simple MATLAB programs are presented.

1.2.1 SIGNAL GENERATION

In this example, we see how to generate and display continuous-time signals in the time domain. One can represent such signals as a function of time. For simulation purposes, a representation of time `t` is needed. Note that the time scale is continuous while computers handle operations in a discrete manner. Continuous time simulation is achieved by considering a very small time interval. For example, if a 1-second duration signal in millisecond (msec) increments (time interval of 0.001 second) is considered, then one sample every 1 msec or a total of 1,000 samples are generated for the entire signal, leading to a continuous signal simulation. This continuous-time signal approximation or simulation is used in later chapters. It is important to note that a finite number of samples is involved in the simulation of a continuous-time signal, and thus to differentiate a continuous-time signal from a discrete-time signal, a much higher number of samples per second for a continuous-time signal needs to be used (very small time interval).
Figure 1.7: Continuous-time signals.

The signal $x_1(t)$ can be represented mathematically as follows:

$$x_1(t) = \begin{cases} 
0 & 0 \leq t < 1 \\
1 & 1 \leq t < 2 \\
0 & 2 \leq t < 3 
\end{cases} \quad (1.1)$$

To simulate this signal, use the MATLAB functions `ones` and `zeros`. The signal value is zero during the first second, which means the first 1,000 samples are zero. This portion of the signal is simulated with the function `zeros(1, 1000)`. In the next second (next 1,000 samples), the signal value is 2, and this portion is simulated by the function `2*ones(1, 1000)`. Finally, the third portion of the signal is simulated by the function `zeros(1, 1000)`. In other words, the entire duration of the signal is simulated by the following .m file function:

```matlab
x1 = [zeros(1, 1/dt) 2*ones(1, 1/dt) zeros(1, 1/dt)]
```

The signal $x_2(t)$ can be represented mathematically as follows:

$$x_2(t) = \begin{cases} 
2t & 0 \leq t < 1 \\
-2t + 4 & 1 \leq t < 2 \\
0 & 2 \leq t < 3 
\end{cases} \quad (1.2)$$

A linearly increasing or decreasing vector can thus be used to represent the linear portions. The time vectors for the three portions or segments of the signal are $0:dt:1-dt$, $1:dt:2-dt$, and $2:dt:3-dt$. The first segment is a linear function corresponding to a time vector with a slope of 2; the second segment is a linear function corresponding to a time vector with a slope of -2 and an offset of 4; and the third segment is simply a constant vector of zeros. In other words,
the entire duration of the signal for any value of $dt$ can be simulated by the following .m file function:

$$x_2([2*(0:dt:(1-dt))-2*(1:dt:(2-dt))+4 \text{ zeros}(1,1/dt)]$$

Figures 1.8 and 1.9 show the MATLAB code and the plot of the above signal generation, respectively. Signals can be displayed using the function `plot(t, data)`. For proper plotting, first the correct $t$ vector needs to be generated. Here this is done by using the function `linspace`:

$$>> t=linspace(0,E,N)$$

![MATLAB code of a signal generation example.](image)

**Figure 1.8:** MATLAB code of a signal generation example.

This function generates a vector $t$ of $N$ points linearly spaced between and including $0$ and $E$, where $N$ is equal to $E/dt$.

### 1.2.2 GENERATING A PERIODIC SIGNAL

In this example, a simple periodic signal is generated. This example involves generating a periodic signal in textual mode and displaying it in graphical mode. The shape of the signal $\sin$.
INTRODUCTION TO MATLAB

Figure 1.9: Signal plots.

square, triangle, or sawtooth) can be modified, as well as its frequency and amplitude, by using appropriate control parameters. The MATLAB code and the plots generated by it are shown in Figures 1.10 and 1.11, respectively.

Now consider an m-file code to generate four types of waveforms using the functions sin, square, and sawtooth. To change the amplitude and frequency of the waveforms, two control parameters named Amplitude (A) and Frequency (f) are used. Waveform Type (w) is another parameter used for controlling the waveform type. With this control parameter, one can select from multiple inputs. Finally, the waveforms are displayed by using the function plot.

1.3 LAB EXERCISES

1. Write an m-file code to add all the numbers corresponding to the even indices of an array. For instance, if the array x is specified as x = [1, 3, 5, 10], then 13 (= 3 + 10) should be returned. Use the program to find the sum of all even integers from 1 to 1,000. Run your code. Also, rewrite the code where x is the input vector and y is the sum of all the numbers corresponding to the even indices of x.

2. Explain the operation performed by the following .m file:

```matlab
L=length(x);
for j=1:L
if x(j) < 0
x(j)=-x(j);
end
end
```
Figure 1.10: Periodic signal generation example.

Figure 1.11: Plot of a periodic sinusoid signal.
1. INTRODUCTION TO MATLAB

Rewrite this program without using a `for` loop.

3. Write an .m file code that implements the following hard-limiting function:

\[ x(t) = \begin{cases} 
0.2 & t \geq 0.2 \\
-0.2 & t < 0.2 
\end{cases} \]  \hspace{1cm} (1.3)

For \( t \), use 1,000 random numbers generated by using the function `rand`.

4. Write a MATLAB code to generate two sinusoid signals with the frequencies \( f_1 \) Hz and \( f_2 \) Hz and the amplitudes \( A_1 \) and \( A_2 \), based on a sampling frequency of 8,000Hz with the number of samples being 256. Set the frequency ranges from 100 to 400Hz and set the amplitude ranges from 20 to 200. Generate a third signal with the frequency \( f_3 = (\text{mod} (\text{lcm} (f_1, f_2), 400) + 100) \) Hz, where `mod` and `lcm` denote the modulus and least common multiple operation, respectively, and the amplitude \( A_3 \) is the sum of the amplitudes \( A_1 \) and \( A_2 \). Use the same sampling frequency and number of samples as specified for the first two signals. Display all the signals using the legend on the same waveform graph and label them accordingly.