Adaptive filters are used in many diverse applications, appearing in everything from military instruments to cellphones and home appliances. Adaptive Filtering: Fundamentals of Least Mean Squares with MATLAB® covers the core concepts of this important field, focusing on a vital part of the statistical signal processing area—the least mean square (LMS) adaptive filter. This largely self-contained text:

- Discusses random variables, stochastic processes, vectors, matrices, determinants, discrete random signals, and probability distributions
- Explains how to find the eigenvalues and eigenvectors of a matrix and the properties of the error surfaces
- Explores the Wiener filter and its practical uses, details the steepest descent method, and develops the Newton’s algorithm
- Addresses the basics of the LMS adaptive filter algorithm, considers LMS adaptive filter variants, and provides numerous examples
- Delivers a concise introduction to MATLAB®, supplying problems, computer experiments, and more than 110 functions and script files

Featuring robust appendices complete with mathematical tables and formulas, Adaptive Filtering: Fundamentals of Least Mean Squares with MATLAB® clearly describes the key principles of adaptive filtering and effectively demonstrates how to apply them to solve real-world problems.
Adaptive Filtering
MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book’s use or discussion of MATLAB® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® software.
Contents

Preface....................................................................................................................... xi
Author ...................................................................................................................... xiii
Abbreviations ............................................................................................................. xv
MATLAB® Functions ............................................................................................... xvii

Chapter 1  Vectors............................................................................................... 1
 1.1 Introduction ................................................................................................... 1
     1.1.1 Multiplication by a Constant and Addition and Subtraction .......... 1
     1.1.1.1 Multiplication by a Constant ..................................................... 1
     1.1.1.2 Addition and Subtraction ....................................................... 2
     1.1.2 Unit Coordinate Vectors .............................................................. 3
     1.1.3 Inner Product ............................................................................. 3
     1.1.4 Distance between Two Vectors ................................................... 5
     1.1.5 Mean Value of a Vector ............................................................... 5
     1.1.6 Direction Cosines ........................................................................ 7
     1.1.7 The Projection of a Vector .......................................................... 9
     1.1.8 Linear Transformations ............................................................... 10
 1.2 Linear Independence, Vector Spaces, and Basis Vectors ........... 11
     1.2.1 Orthogonal Basis Vectors ......................................................... 13
Problems ................................................................................................................. 13
Hints–Suggestions–Solutions .................................................................................... 14

Chapter 2  Matrices............................................................................................. 17
 2.1 Introduction ................................................................................................... 17
 2.2 General Types of Matrices ......................................................................... 17
     2.2.1 Diagonal, Identity, and Scalar Matrices ....................................... 17
     2.2.2 Upper and Lower Triangular Matrices ....................................... 17
     2.2.3 Symmetric and Exchange Matrices .......................................... 18
     2.2.4 Toeplitz Matrix ......................................................................... 18
     2.2.5 Hankel and Hermitian ............................................................... 18
 2.3 Matrix Operations ....................................................................................... 18
 2.4 Determinant of a Matrix ........................................................................... 21
     2.4.1 Definition and Expansion of a Matrix ....................................... 21
     2.4.2 Trace of a Matrix ...................................................................... 22
     2.4.3 Inverse of a Matrix .................................................................. 22
 2.5 Linear Equations ....................................................................................... 24
     2.5.1 Square Matrices ($n \times n$) ...................................................... 24
     2.5.2 Rectangular Matrices ($n < m$) .............................................. 26
     2.5.3 Rectangular Matrices ($m < n$) .............................................. 27
## Contents

2.5.4 Quadratic and Hermitian Forms ........................................ 29  
2.6 Eigenvalues and Eigenvectors ........................................... 31  
   2.6.1 Eigenvectors ..................................................... 32  
   2.6.2 Properties of Eigenvalues and Eigenvectors ............. 33  
Problems .............................................................................. 36  
Hints–Suggestions–Solutions .................................................. 37  

### Chapter 3  Processing of Discrete Deterministic Signals: Discrete Systems  

3.1 Discrete-Time Signals .................................................... 41  
   3.1.1 Time-Domain Representation of Basic  
      Continuous and Discrete Signals ............................... 41  
3.2 Transform-Domain Representation of Discrete Signals ...... 42  
   3.2.1 Discrete-Time Fourier Transform ......................... 42  
   3.2.2 The Discrete FT ................................................. 44  
   3.2.3 Properties of DFT ............................................... 46  
3.3 The z-Transform......................................................... 48  
3.4 Discrete-Time Systems .................................................. 52  
   3.4.1 Linearity and Shift Invariant .................................. 52  
   3.4.2 Causality .......................................................... 52  
   3.4.3 Stability ............................................................ 52  
   3.4.3 Transform-Domain Representation ....................... 57  
Problems .............................................................................. 60  
Hints–Suggestions–Solutions .................................................. 61  

### Chapter 4  Discrete-Time Random Processes  

4.1 Discrete Random Signals, Probability Distributions,  
   and Averages of Random Variables ................................. 63  
   4.1.1 Stationary and Ergodic Processes ........................... 65  
   4.1.2 Averages of RV .................................................. 66  
      4.1.2.1 Mean Value ............................................... 66  
      4.1.2.2 Correlation ................................................. 67  
      4.1.2.3 Covariance ............................................... 69  
4.2 Stationary Processes ..................................................... 71  
   4.2.1 Autocorrelation Matrix ....................................... 71  
   4.2.2 Purely Random Process (White Noise) ................. 74  
   4.2.3 Random Walk .................................................. 74  
4.3 Special Random Signals and pdf’s ................................. 75  
   4.3.1 White Noise .................................................... 75  
   4.3.2 Gaussian Distribution (Normal Distribution) ......... 75  
   4.3.3 Exponential Distribution ................................... 78  
   4.3.4 Lognormal Distribution .................................... 79  
   4.3.5 Chi-Square Distribution ..................................... 80  
4.4 Wiener–Khinchin Relations ......................................... 80  
4.5 Filtering Random Processes ........................................ 83
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6 Special Types of Random Processes</td>
<td>85</td>
</tr>
<tr>
<td>4.6.1 Autoregressive Process</td>
<td>85</td>
</tr>
<tr>
<td>4.7 Nonparametric Spectra Estimation</td>
<td>88</td>
</tr>
<tr>
<td>4.7.1 Periodogram</td>
<td>88</td>
</tr>
<tr>
<td>4.7.2 Correlogram</td>
<td>90</td>
</tr>
<tr>
<td>4.7.3 Computation of Periodogram and Correlogram Using FFT</td>
<td>90</td>
</tr>
<tr>
<td>4.7.4 General Remarks on the Periodogram</td>
<td>91</td>
</tr>
<tr>
<td>4.7.4.1 Windowed Periodogram</td>
<td>93</td>
</tr>
<tr>
<td>4.7.5 Proposed Book Modified Method for Better Frequency Resolution</td>
<td>95</td>
</tr>
<tr>
<td>4.7.5.1 Using Transformation of the rv’s</td>
<td>95</td>
</tr>
<tr>
<td>4.7.5.2 Blackman–Tukey Method</td>
<td>96</td>
</tr>
<tr>
<td>4.7.6 Bartlett Periodogram</td>
<td>100</td>
</tr>
<tr>
<td>4.7.7 The Welch Method</td>
<td>106</td>
</tr>
<tr>
<td>4.7.8 Proposed Modified Welch Methods</td>
<td>109</td>
</tr>
<tr>
<td>4.7.8.1 Modified Method Using Different Types of Overlapping</td>
<td>109</td>
</tr>
<tr>
<td>4.7.8.2 Modified Welch Method Using Transformation of rv’s</td>
<td>111</td>
</tr>
<tr>
<td>4.7.9 Proposed Book Modified Method for Better Frequency Resolution</td>
<td>113</td>
</tr>
<tr>
<td>Problems</td>
<td>113</td>
</tr>
<tr>
<td>Hints–Solutions–Suggestions</td>
<td>114</td>
</tr>
</tbody>
</table>

### Chapter 5

The Wiener Filter .................................................................................. 121

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>121</td>
</tr>
<tr>
<td>5.2 The LS Technique</td>
<td>121</td>
</tr>
<tr>
<td>5.2.1 Linear LS</td>
<td>122</td>
</tr>
<tr>
<td>5.2.2 LS Formulation</td>
<td>125</td>
</tr>
<tr>
<td>5.2.3 Statistical Properties of LSEs</td>
<td>130</td>
</tr>
<tr>
<td>5.2.4 The LS Approach</td>
<td>132</td>
</tr>
<tr>
<td>5.2.5 Orthogonality Principle</td>
<td>135</td>
</tr>
<tr>
<td>5.2.6 Corollary</td>
<td>135</td>
</tr>
<tr>
<td>5.2.7 Projection Operator</td>
<td>136</td>
</tr>
<tr>
<td>5.2.8 LS Finite Impulse Response Filter</td>
<td>138</td>
</tr>
<tr>
<td>5.3 The Mean-Square Error</td>
<td>140</td>
</tr>
<tr>
<td>5.3.1 The FIR Wiener Filter</td>
<td>142</td>
</tr>
<tr>
<td>5.4 The Wiener Solution</td>
<td>146</td>
</tr>
<tr>
<td>5.4.1 Orthogonality Condition</td>
<td>148</td>
</tr>
<tr>
<td>5.4.2 Normalized Performance Equation</td>
<td>149</td>
</tr>
<tr>
<td>5.4.3 Canonical Form of the Error-Performance Surface</td>
<td>150</td>
</tr>
<tr>
<td>5.5 Wiener Filtering Examples</td>
<td>151</td>
</tr>
<tr>
<td>5.5.1 Minimum MSE</td>
<td>154</td>
</tr>
<tr>
<td>5.5.2 Optimum Filter ($w^o$)</td>
<td>154</td>
</tr>
<tr>
<td>5.5.3 Linear Prediction</td>
<td>161</td>
</tr>
<tr>
<td>Problems</td>
<td>162</td>
</tr>
</tbody>
</table>
Chapter 6  

Eigenvalues of $R_x$: Properties of the Error Surface .......................... 171

6.1  The Eigenvalues of the Correlation Matrix ......................... 171

6.1.1  Karhunen–Loeve Transformation .............................. 172

6.2  Geometrical Properties of the Error Surface .................. 174

Problems ........................................................................... 178

Hints–Solutions–Suggestions .................................................. 178

Chapter 7  

Newton’s and Steepest Descent Methods ...................................... 183

7.1  One-Dimensional Gradient Search Method ....................... 183

7.1.1  Gradient Search Algorithm ......................................... 183

7.1.2  Newton’s Method in Gradient Search.......................... 185

7.2  Steepest Descent Algorithm ............................................. 186

7.2.1  Steepest Descent Algorithm Applied to Wiener Filter ... 187

7.2.2  Stability (Convergence) of the Algorithm .................. 188

7.2.3  Transient Behavior of MSE ....................................... 190

7.2.4  Learning Curve ......................................................... 191

7.3  Newton’s Method ............................................................ 192

7.4  Solution of the Vector Difference Equation ...................... 194

Problems ........................................................................... 197

Edition Problems .................................................................. 197

Hints–Solutions–Suggestions .................................................. 198

Chapter 8  

The Least Mean-Square Algorithm ............................................ 203

8.1  Introduction ................................................................ 203

8.2  The LMS Algorithm ...................................................... 203

8.3  Examples Using the LMS Algorithm .............................. 206

8.4  Performance Analysis of the LMS Algorithm .................. 219

8.4.1  Learning Curve ......................................................... 221

8.4.2  The Coefficient-Error or Weighted-Error Correlation Matrix .................................................... 224

8.4.3  Excess MSE and Misadjustment .............................. 225

8.4.4  Stability ................................................................. 227

8.4.5  The LMS and Steepest Descent Methods .................. 228

8.5  Complex Representation of the LMS Algorithm ............... 228

Problems ........................................................................... 231

Hints–Solutions–Suggestions .................................................. 232

Chapter 9  

Variants of Least Mean-Square Algorithm .................................. 239

9.1  The Normalized Least Mean-Square Algorithm .............. 239

9.2  Power Normalized LMS ................................................ 244
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3</td>
<td>Self-Correcting LMS Filter</td>
<td>248</td>
</tr>
<tr>
<td>9.4</td>
<td>The Sign-Error LMS Algorithm</td>
<td>250</td>
</tr>
<tr>
<td>9.5</td>
<td>The NLMS Sign-Error Algorithm</td>
<td>250</td>
</tr>
<tr>
<td>9.6</td>
<td>The Sign-Regressor LMS Algorithm</td>
<td>252</td>
</tr>
<tr>
<td>9.7</td>
<td>Self-Correcting Sign-Regressor LMS Algorithm</td>
<td>253</td>
</tr>
<tr>
<td>9.8</td>
<td>The Normalized Sign-Regressor LMS Algorithm</td>
<td>253</td>
</tr>
<tr>
<td>9.9</td>
<td>The Sign–Sign LMS Algorithm</td>
<td>254</td>
</tr>
<tr>
<td>9.10</td>
<td>The Normalized Sign–Sign LMS Algorithm</td>
<td>255</td>
</tr>
<tr>
<td>9.11</td>
<td>Variable Step-Size LMS</td>
<td>257</td>
</tr>
<tr>
<td>9.12</td>
<td>The Leaky LMS Algorithm</td>
<td>259</td>
</tr>
<tr>
<td>9.13</td>
<td>The Linearly Constrained LMS Algorithm</td>
<td>262</td>
</tr>
<tr>
<td>9.14</td>
<td>The Least Mean Fourth Algorithm</td>
<td>264</td>
</tr>
<tr>
<td>9.15</td>
<td>The Least Mean Mixed Norm LMS Algorithm</td>
<td>265</td>
</tr>
<tr>
<td>9.16</td>
<td>Short-Length Signal of the LMS Algorithm</td>
<td>266</td>
</tr>
<tr>
<td>9.17</td>
<td>The Transform Domain LMS Algorithm</td>
<td>267</td>
</tr>
<tr>
<td>9.17.1</td>
<td>Convergence</td>
<td>271</td>
</tr>
<tr>
<td>9.18</td>
<td>The Error Normalized Step-Size LMS Algorithm</td>
<td>272</td>
</tr>
<tr>
<td>9.19</td>
<td>The Robust Variable Step-Size LMS Algorithm</td>
<td>276</td>
</tr>
<tr>
<td>9.20</td>
<td>The Modified LMS Algorithm</td>
<td>282</td>
</tr>
<tr>
<td>9.21</td>
<td>Momentum LMS</td>
<td>283</td>
</tr>
<tr>
<td>9.22</td>
<td>The Block LMS Algorithm</td>
<td>285</td>
</tr>
<tr>
<td>9.23</td>
<td>The Complex LMS Algorithm</td>
<td>286</td>
</tr>
<tr>
<td>9.24</td>
<td>The Affine LMS Algorithm</td>
<td>288</td>
</tr>
<tr>
<td>9.25</td>
<td>The Complex Affine LMS Algorithm</td>
<td>290</td>
</tr>
<tr>
<td>9.26</td>
<td>Problems</td>
<td>291</td>
</tr>
<tr>
<td></td>
<td>Hints–Solutions–Suggestions</td>
<td>293</td>
</tr>
</tbody>
</table>

**Appendix 1: Suggestions and Explanations for MATLAB Use**

A1.1 Suggestions and Explanations for MATLAB Use

A1.1.1 Creating a Directory
A1.1.2 Help
A1.1.3 Save and Load
A1.1.4 MATLAB as Calculator
A1.1.5 Variable Names
A1.1.6 Complex Numbers
A1.1.7 Array Indexing
A1.1.8 Extracting and Inserting Numbers in Arrays
A1.1.9 Vectorization
A1.1.10 Windowing
A1.1.11 Matrices
A1.1.12 Producing a Periodic Function
A1.1.13 Script Files
A1.1.14 Functions
A1.1.15 Complex Expressions
A1.1.16 Axes
A1.1.17 2D Graphics
A1.1.18 3D Plots ..............................................................308
  A1.1.18.1 Mesh-Type Figures ........................308
A1.2 General Purpose Commands ...............................................309
  A1.2.1 Managing Commands and Functions................309
  A1.2.2 Managing Variables and Workplace .................309
  A1.2.3 Operators and Special Characters...............309
  A1.2.4 Control Flow............................................ 310
A1.3 Elementary Matrices and Matrix Manipulation..............311
  A1.3.1 Elementary Matrices and Arrays ...............311
  A1.3.2 Matrix Manipulation .....................................311
A1.4 Elementary Mathematical Functions ........................312
  A1.4.1 Elementary Functions..................................312
A1.5 Numerical Linear Algebra ...........................................313
  A1.5.1 Matrix Analysis........................................313
A1.6 Data Analysis ..........................................................313
  A1.6.1 Basic Operations..................................313
  A1.6.2 Filtering and Convolution .........................313
  A1.6.3 Fourier Transforms ...................................314
A1.7 2D Plotting ..............................................................314
  A1.7.1 2D Plots ................................................314

Appendix 2: Matrix Analysis ..................................................317
  A2.1 Definitions ........................................................317
  A2.2 Special Matrices ............................................319
  A2.3 Matrix Operation and Formulas .......................322
  A2.4 Eigendecomposition of Matrices .....................325
  A2.5 Matrix Expectations .......................................326
  A2.6 Differentiation of a Scalar Function with respect to a Vector ......327

Appendix 3: Mathematical Formulas ....................................329
  A3.1 Trigonometric Identities ................................329
  A3.2 Orthogonality .............................................330
  A3.3 Summation of Trigonometric Forms ...............331
  A3.4 Summation Formulas ................................331
    A3.4.1 Finite Summation Formulas .................331
    A3.4.2 Infinite Summation Formulas .................331
  A3.5 Series Expansions ..........................................332
  A3.6 Logarithms ..................................................332
  A3.7 Some Definite Integrals ................................332

Appendix 4: Lagrange Multiplier Method ..............................335

Bibliography ........................................................................337
Preface

This book is written for the applied scientists and engineers who want or need to learn the fundamentals about this subject, but are not experts in this specific field. It is also written to accompany a first graduate course in digital signal processing. In this book, we have selected the most important part, the least mean-square (LMS) adaptive filter, which is part of the statistical signal processing area. The adaptive filters have found their use in many and diverse fields such as communications, control, radar, sonar, and seismology.

The aim of this book is to present in a fundamental way the area of adaptive filtering concentrating to a specific type of adaptive filters and their variations, known as the LMS adaptive filter. All the adaptive filters are of the finite duration or finite impulse response filters (FIRs). Since the signals are random or nonrandom with additive random signals, noise, an introduction to random variables and stochastic processes is also included. Because many of the formulas contain vectors and matrices, appropriate chapters were included.

The book contains all the material necessary for the reader to study the content without the need to refer to other sources. It includes appendices to provide the needed extra supporting material. It also includes a number of MATLAB® functions and m-files for practising and verifying the material provided in the text. In addition, it provides more than 110 Book MATLAB functions and script files, which specifically were developed to verify the theory and proposed examples. It also provides many computer experiments to illustrate theories such as the Wiener and adaptive filtering. At the end of each chapter, numerous problems are provided to help the reader develop a deeper understanding of the presented material. Detailed solutions or hints and suggestions for solving all the problems are provided.

Chapter 1 covers vectors and their properties; Chapter 2 covers matrices and determinants; Chapter 3 provides a synopsis of the deterministic discrete systems and signals; Chapter 4 covers the fundamentals of the discrete random signals, probability distributions, and their properties as well as their simulations; Chapter 5 presents the Wiener filter and its practical uses; Chapter 6 explains how to find the eigenvalues and eigenvectors of a matrix and the properties of the error surfaces; Chapter 7 presents the steepest descent method and develops the Newton’s algorithm; Chapter 8 covers the basics of the celebrated LMS adaptive filter algorithm and presents numerous examples for different types of applications; and Chapter 9 covers the variants of the LMS adaptive filters. Appendix 1 explains the use of the MATLAB, Appendix 2 covers the fundamentals of the matrix theory, Appendix 3 presents some mathematical formulas, and Appendix 4 develops the method of Lagrange multiplier method.
MATLAB® is a registered trademark of The MathWorks, Inc. For product information, please contact:

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098, USA
Tel: +1 508 647 7000
Fax: +1 508 647 7001
E-mail: info@mathworks.com
Web: www.mathworks.com
Alexander D. Poularikas received his PhD from the University of Arkansas and became a professor at the University of Rhode Island. He was chairman of the engineering department at the University of Denver, Colorado, and then became chairman of the electrical and computer engineering department at the University of Alabama in Huntsville. He has published, coauthored, and edited 14 books. Dr. Poularikas served as an editor-in-chief of the Signal Processing Series (1993–1997) with Artech House and is now an editor-in-chief of the Electrical Engineering and Applied Signal Processing Series as well as the Engineering and Science Primers Series (1998 till present) with Taylor & Francis Group. He was a Fulbright scholar and is a lifelong senior member of the Institute of Electrical and Electronics Engineers (IEEE) and a member of Tau Beta Pi, Sigma Nu, and Sigma Pi. In 1990 and 1996, he received the IEEE Outstanding Educators Award, Huntsville Section.
Abbreviations

AMWM asymmetric modified Welch method
ARMA autoregressive moving average
BIBO bounded input bounded output
BT Blackman–Tukey
cdf cumulative distribution function
CRLB Cramer–Rao lower bound
DFT discrete Fourier transform
DTFT discrete-time Fourier transform
FFT fast Fourier transform
FIR finite impulse response
FT Fourier transform
ENSS error normalized step size
IDFT inverse discrete Fourier transform
IFFT inverse fast Fourier transform
iid independent and identically distributed
IIR infinite impulse response
LMMN least mean mixed norm LMS algorithm
LMF least mean fourth algorithm
LMS least mean-square algorithm
LS least squares
LSI linear shift invariance
LSE least-squares estimator
LTI linear time invariance
MMSE minimum mean-square error
MSE mean-square error
MVU minimum variance unbiased
NLMS normalized LMS algorithm
pdf probability density function
PNLMS power normalized LMS
pr probability
PSD power spectral density
QAM quadrature amplitude modulation
rv random variable
RVSS robust variable step size
RW random walk
SCLMSF self-correcting LMS filter
SCNLMSF self-correcting normalized LMS filter
SCWF self-correcting Wiener filter
SMWM symmetric modified Welch method
SNP signal-to-noise power
SPMWM symmetric predicted modified Welch method
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDLMS</td>
<td>transform domain LMS</td>
</tr>
<tr>
<td>VSLMS</td>
<td>variable step-size LMS</td>
</tr>
<tr>
<td>WN</td>
<td>white noise</td>
</tr>
<tr>
<td>WSS</td>
<td>wide-sense stationary process</td>
</tr>
</tbody>
</table>
MATLAB® Functions

\[ m, z \] = hist(x, a)  
\quad \text{counts in each bin; } z = \text{coordinates of the } a \text{ bins;}
\quad w = \max(x)/\text{length}(z); bp = m/(w*a) = \text{probability per bin;}

\[ [r, p, k] = \text{residue(nu, de)} \]
\quad \text{Example: } F(z) = z^2/(z^2 - 0.04), \quad nu = [1 \ 0 \ 0],
\quad de = [1 \ 0 \ -0.04], \quad r = [0.1000 \ -0.1000],
\quad p = [0.2000 \ -0.2000], \quad k = 1, \quad \text{hence } F(z) = 0.1(z/(z - 0.2)) -
\quad 0.1(z/(z + 0.2)) + 1, \quad \text{the inverse is } f(n) = 0.1(0.2)^n - 0.1(-0.2)^n +
\quad \delta(n); \quad r = \text{residues, } p = \text{poles}

\[ [r] = \text{xcorr(x, y, ‘biased’)} \]
\quad \text{Produces a biased cross-correlation; if } x = y, \quad r = \text{autocorrelation;}
\quad \text{the } r \text{ is an } 2N - 1 \text{ symmetric function with respect to } 0

\[ [r] = \text{xcorr(x, y, ‘unbiased’)} \]
\quad \text{Produces an unbiased cross-correlation; if } x = y,
\quad r = \text{autocorrelation}

\omega
\quad \text{Produces the lowercase omega, the same for the rest of the}
\quad \text{Greek letters}

\Omega
\quad \text{Produces the uppercase omega, the same for the rest of the}
\quad \text{Greek letters}

A(:,1)
\quad \text{Gives } [1 \ 4]^T \text{ for matrix } A = [1 \ 2; 4 \ 3], \text{ all rows of the first}
\quad \text{column}

A(2,:)
\quad \text{Gives } [2 \ 4] \text{ for the matrix } A = [1 \ 2; 4 \ 3], \text{ all columns for the}
\quad \text{second row}

\quad \text{a'*a}
\quad \text{a}^T \cdot b^T \text{ is a row vector } c = (a_1b_1, a_2b_2, \ldots, a_nb_n)

abs()
\quad \text{Gives the absolute value of the complex expression in the}
\quad \text{parentheses}

acos()
\quad \text{Produces the inverse of the cosine}

angle()
\quad \text{Gives the angle of a complex number or vector}

asin()
\quad \text{Produces the inverse of the sine}

atan()
\quad \text{Produces the inverse of the tangent}

axis([0 2 -2 5])
\quad \text{Creates axes with length 0 to 2 in the } x\text{-axis and } -2 \text{ to 5 in the}
\quad \text{y-axis}

bar(z,p)
\quad \text{Plots bars at the } x\text{-coordinates of } z \text{ and heights } p

bar(z,pb)
\quad \text{Plots the probability distribution with } z\text{ bars}

color designation
\quad ‘y’ = yellow, ‘m’ = magenta, ‘c’ = cyan, ‘r’ = red, ‘g’ = green,
\quad ‘b’ = blue, ‘w’ = white, ‘k’ = black

colormap(gray)
\quad \text{Produces black figure; see also } meshgrid()

conj()
\quad \text{Gives the complex conjugate of a complex number or expression}

conj(z)
\quad \text{Gives the conjugate value of the complex number } z,
\quad \text{conj}(3 - 0.5i) = 3.0000 + 5.0000i

contour(X,Y,Z,V,’g’)
\quad \text{V = [a \ b \ c] is a vector with values } a, b, \text{ and } c \text{ which can be any}
\quad \text{number; the contour will have green color}

(Continued)
\begin{verbatim}
contour(Z,[1.6 3.1 5])  Produces contours of the performance surface at heights 1.6, 3.1, and 5

contour(Z,30)          Creates 30 contours of the performance surface

conv(x,y)              Convolution between two vectors x and y; the output vector signal will have length(length(x) + length(y) − 1

det(A)                 Gives the determinant of A; we can also write det([1 2;4 3]) for 2 × 2 matrix

dimpulse(nu,de,5)      Inverts the z-transform. Example: z^2/(z^3 + 0.1),

                       nu = [0 1 0 0], de = [1 0 0 0.1], gives 5 terms of the time function

dimpulse(num,den,a)    Gives a numbers of the time function f(n), num = vector with the values of the numerator coefficients of the
                       z-function in descending order; den = same as for the num but for the denominator

dot                    a .* b is a column vector with elements c = (a_1b_1, a_2b_2,⋯,a_nb_n)

det(R)                 [Q,D] = eig(R), R = a correlation matrix, Q = each one of its columns equal to an
                       eigenvector, D = a diagonal matrix with the eigenvalues of the correlation matrix

diag(4)                Creates a 4 × 4 identity matrix, its main diagonal have unit value

fft(x,N)               Takes the DFT of the sequence (vector) x having N elements

figure(2)              Creates a second figure and the next plot will be introduced at this new figure

filter(b,a,x)          b = a vector of the input coefficients; a = a vector of the output coefficients; x = a vector
                       of the input to the system; y(n) = an output

                       y(n) + \sum_{m=0}^{\infty} a_{m+n} y(n-m) = \sum_{m=0}^{\infty} b_{m+n} x(n-m)

fliplr(x)              Flips a vector from left to right, x = a vector

floor(a)               Gives the number without the decimal, floor(5.68) = 5, floor(-2.6) = -3

freqz();              B = a vector of the b’s coefficients of the numerator of the
                       transfer function \( H(e^{j\omega}) \); A = a vector of the a’s coefficients of
                       the denominator of \( H(e^{j\omega}) \); N = number of bins in the frequency
                       domain; W = a frequency vector in rad/sample of the filter
                       \( H(e^{j\omega}) \) = transfer function

grid on               After plot (), we write grid on and a grid will appear with horizontal and vertical lines

hist(a,\alpha)         Creates a histogram with \( \alpha \) number of bins from the elements of the
                       vector a

hold on               We make a plot first; then we write hold on and then we proceed for the second plot on the same figure

ifft(X,N)             Takes the inverse DFT (IDFT) of the frequency sequence (vector) \{x\} with N elements

inv(A)                Gives the inverse of matrix A, \( A^{-1} \)

inv([1 2;4 3])        inv([1 2;4 3]) to invert a 2 × 2 matrix

legend(‘FT’)          Produces a legend identifying with the type of plotting, here FT

length(a)             Gives the number of the elements of the vector a
\end{verbatim}
log(a_i)  
Gives the natural logarithm of the number \( a_i \)

\( \log_{10}(a_i) \)  
Gives the logarithm of the number \( a_i \) with base 10

matrix: \( A(4:6,8:15) \)  
Produces a \( 3 \times 8 \) matrix from the elements of \( A \) that contains the rows 4 to 6 and columns 8 to 15

matrix: \( A \cdot n \)  
Raises every element in the power of \( n \)

mean(a)  
Gives the mean value of the elements of the vector \( a \)

meshgrid(x,y)  
Example: \( x = 0:0.1:4; \) \( y = 0:0.1:4; \) \( [X,Y] = \text{meshgrid}(x,y); \)
\( Z = 4 \cdot X.^2 + Y.^3; \) \% (the functions we want to produce);
\( \text{surf}(X,Y,Z); \) colormap(gray); \% black figure; see color above

norm(x)  
Gives the norm of \( x \), the square root of the sum of the square of the vector \( x \) coefficients number; the contours will have color green

\( \text{plot}(x,y) \)  
Plots a blue color graph \( x \) versus \( y \)

\( \text{plot}(x,y,'k') \)  
Plots a black color graph, ‘g’=green, ‘r’=red

\( r = [0.1000 \ -0.1000], \ p = [0.2000 \ -0.2000], \ k = 1, \) hence

\( \text{rand}(1,n) \)  
Produces a vector of \( n \) elements that are white noise with 0.5 mean value; \( \text{rand}(1,n) - 0.5 \) is a vector of \( n \) elements which are uniformly distributed with zero mean

\( \text{randn}(1,n) \)  
produces a vector of \( n \) elements that are normally distributed

\( \text{rank}(A) \)  
Gives the rank of the matrix \( A \), the number of independent columns of the matrix

\( \text{roots}(a) \)  
Give the roots of the above polynomial; \( a \) is a vector of the coefficients of the polynomial, e.g.
\( a \lambda^3 + b \lambda^2 + c \lambda + d = 0 \), then we write \( a = [a \ b \ c \ d] \)

\( \text{sign}() \)  
Examples: \( \text{sign}(0.2) = 1, \ \text{sign}(-2) = -1, \ \text{sign}(0) = 0, \)
\( \text{sign}([-2 \ 0.1 \ 0 \ -0.3 \ -5]) = [-1 \ 1 \ 0 \ -1 \ -1] \)

\( \text{sqrt}() \)  
Takes the square root of a number

\( \text{std}(x) \)  
Standard deviation = square root of the variance, \( x \) = a data vector

\( \text{stem}(x,y) \)  
Produces a set of lines of magnitude of the set \( y \)

\( \text{stem}(x,y,'k','filled') \)  
Produces a black color graph with the top circles filled with black color

\( \text{subplot}(4,1,3) \)  
Produces a subplot on the third row (3) of a figure that is made of four rows (4) and one column (1);

\( \text{subplot}(5,2,3) \)  
Produces a figure of 5 rows and 2 columns and plots the third subplot

\( \text{sum}(a) \)  
Sums the elements of a vector \( a \)

\( \text{sum}(X,1) \)  
Sums all rows of the matrix \( X \)

\( \text{sum}(X,2) \)  
Sums all the columns of the matrix \( X \)

\( \text{surf}(X,Y,Z) \)  
Produces a three-dimensional figure; see also meshgrid();

\( \text{title('Figure')} \)  
Produces the title “Figure” on the top of the figure

\( \text{trace}(A) \)  
Sums the main diagonal of the matrix \( A \)

\( \text{transpose()} \)  
Transposes the matrix

\( \text{var}(a) \)  
Gives the variance of the elements of vector \( a \)

\( \text{var}(x) \)  
Gives the variance of the data vector \( x \)

(Continued)
MATLAB® Functions

\begin{verbatim}
xcorr(x,y)     Gives two-sided correlation of the two vectors; if x = y, the
              one-sided correlation starts from length(x) to
              length(x) + length(y) - 1; if x < y, MATLAB adds zeros to the
              smaller, the one-sided correlation starts from length(x) and
              ends at 2length(y) - 1

xlabel('x_{cD}') On the x-axis will appear \( x_{cD} \)
xlabel('x-axis')  On the x-axis will appear x-axis
ylabel('y-label') On the y-axis will appear y-axis
\end{verbatim}
Vectors

1.1 INTRODUCTION

In this text we present vectors with boldfaced lowercase Latin letters, in columnar format. Matrices are presented with boldfaced capital Latin letters and numbers with lowercase Latin letters. For example, for three different types of three element vectors, we write

\[
x = \begin{bmatrix} 1 \\ 2.3 \\ -2 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad z = \begin{bmatrix} 2.33 \\ 4 + j5 \\ -j \end{bmatrix}
\]

(1.1)

The elements \(y_1, y_2, \) and \(y_3\) are constants. For any one of the above vectors, we could also write its equivalent form, for example, \(x^T = [1 \ 2.3 \ -2]\) or \(x^T = (1, 2.3, -2)\), where the letter \(T\) stands for the word transpose. The MATLAB® column vector format is written in the command window as follows:

```matlab
>> x = [1 2.3 -2]'; % column vector, accent indicates transpose;
>> x = [1; 2.3; -2]; % column vector;
>> x = [1 2.3 -2]; % row vector; () are not used in MATLAB for
> % vector manipulations;
```

We can consider a column vector as a special matrix with \(m\) rows, in this case 3, and \(n\) columns, in this case 1.

1.1.1 Multiplication by a Constant and Addition and Subtraction

1.1.1.1 Multiplication by a Constant

Using two vectors from (1.1), we write \(a_1 x^T = [a_1 1 \ a_1 2.3 \ -a_1 2]\) and \(a_1 z^T = [a_1 2.33 \ a_1 4 + a_1 j5 \ -a_1 j]\), where \(a_1\) is a constant. Equivalent expressions in MATLAB are the vectors \(a_1 x\) and \(a_1 z\) (of course, in MATLAB all the elements must be numbers). We named the column vector \(a_1 x\), though we could have given it any other name. Hence, in the MATLAB command window, we write

```matlab
>> a1*x = [a1*1 a1*2.3 -a1*2]';
>> a1*z = [a1*2.33 a1*4+a1*j*5 -a1*j]';
```
1.1.1.2 Addition and Subtraction

We obtain summation and/or subtraction for the two vectors \( \mathbf{a} \) and \( \mathbf{b} \) as follows (each operation must be done separately):

\[
\mathbf{z} = \mathbf{a} \pm k \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \pm k \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 \pm k b_1 \\ a_2 \pm k b_2 \\ a_3 \pm k b_3 \end{bmatrix}
\]  

(1.2)

Having the vectors \( \mathbf{a} \) and \( \mathbf{b} \) in numerical forms, the MATLAB operations are given as follows:

\[
>> \mathbf{a} = [a_1, a_2, a_3]'; \mathbf{b} = [b_1, b_2, b_3]'; \% \text{column vectors;}
\]

\[
>> \mathbf{z} = \mathbf{a} + k_1 \mathbf{b}; \% \text{\( \mathbf{z} \) is a column vector;}
\]

or

\[
>> \mathbf{z} = \mathbf{a} - k_1 \mathbf{b}; \% \text{\( \mathbf{z} \) is a column vector;}
\]

A vector \( \mathbf{a} \) with two elements, for example, can be considered as a line in the \( x-y \) plane starting at the origin and ending at the point \((a_1, a_2)\). If a vector has \( n \) elements, the vector starts from the origin of an \( n \)-dimensional space and ends at the point \((a_1, a_2, \ldots, a_n)\).

If \( \mathbf{a} \), \( \mathbf{b} \), and \( \mathbf{c} \) are \( n \)-dimensional vectors, belonging to \( n \)-dimensional vector space, and \( v_1 \) and \( v_2 \) are constants, then the rules in Table 1.1 are applicable.

Figure 1.1 shows two vectors in two-dimensional vector space, their sum, and their difference.

---

### Table 1.1

**Properties of Vectors**

1. \( \mathbf{a} + \mathbf{b} \) is an \( n \)-dimensional vector
2. \( \mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a} \)
3. \( (\mathbf{a} + \mathbf{b}) + \mathbf{c} = \mathbf{a} + (\mathbf{b} + \mathbf{c}) \)
4. \( \mathbf{a} + \mathbf{0} = \mathbf{a} \) (\( \mathbf{0} \) is \( n \)-dimensional)
5. \( \mathbf{a} + (-\mathbf{a}) = \mathbf{0} \)
6. \( v_1 \mathbf{a} \) is an \( n \)-dimensional vector
7. \( v_1 (\mathbf{a} + \mathbf{b}) = v_1 \mathbf{a} + v_1 \mathbf{b} \)
8. \( (v_1 + v_2)\mathbf{a} = v_1 \mathbf{a} + v_2 \mathbf{a} \)
9. \((v_1 + v_2)\mathbf{a} = (v_1 v_2)\mathbf{a} \)
10. \( 1(\mathbf{a}) = \mathbf{a} \)
11. \( \mathbf{a} + \mathbf{b} = \mathbf{a} \) implies \( \mathbf{b} = \mathbf{0} \) (\( \mathbf{0} \) is \( n \)-dimensional)
12. \( \mathbf{a} + \mathbf{b} = \mathbf{0} \) implies \( \mathbf{b} = -\mathbf{a} \) (\( \mathbf{0} \) is \( n \)-dimensional)
13. \( \mathbf{0} \mathbf{a} = \mathbf{0} \)
14. \( \mathbf{a} \mathbf{0} = \mathbf{0} \) (both zeros are \( n \)-dimensional)
15. If \( v_1 \mathbf{a} = \mathbf{0} \), then \( v_1 = \mathbf{0} \) or \( \mathbf{a} = \mathbf{0} \)
16. \( -(-\mathbf{a}) = \mathbf{a} \)

---


## 1.1.2 Unit Coordinate Vectors

For two-dimensional and three-dimensional vector spaces, for example, the unit vectors along the \( x \)-, \( y \)-, and \( z \)-axes are as follows:

\[
e_x = [1 \ 0]^T, \ e_y = [0 \ 1]^T; \quad e_x = [1 \ 0 \ 0]^T, \ e_y = [0 \ 1 \ 0]^T, \ e_z = [0 \ 0 \ 1]^T
\]

Therefore, the vector \( \mathbf{a} = [a_1 \ a_2 \ a_3] \) can also be presented in the following form:

\[
\mathbf{a} = a_1 \mathbf{e}_x + a_2 \mathbf{e}_y + a_3 \mathbf{e}_z \quad \text{or} \quad \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}
\]

### 1.1.3 Inner Product

The **inner** product of two vectors with the same number of elements is given by

\[
\alpha = \text{number} = \mathbf{a}^T \mathbf{b} = a_1b_1 + a_2b_2 + \cdots + a_nb_n = \sum_{i=1}^{n} a_ib_i
\]

\[
>> \alpha = \mathbf{a}^T \mathbf{b}; % \text{MATLAB format}
\]

\[
>> \alpha = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}^* \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}; % \text{MATLAB format}
\]

The MATLAB operation is as follows:

\[
>> \text{ab} = \mathbf{a}^T \mathbf{b};
\]
where \( \mathbf{a} \) and \( \mathbf{b} \) are column vectors, as we have decided to present the vectors in this text, and \( \mathbf{a}^T \mathbf{b} \) is a number and is the inner product of the two vectors. There are also other mathematical definitions of the inner product, for example, \( \mathbf{a} \cdot \mathbf{b} = \langle \mathbf{a}, \mathbf{b} \rangle \). However, we use the above definition since it resembles the MATLAB format.

For example, for the vectors \( \mathbf{a}^T = [2 \; 1 + j \; 2* j] \) and \( \mathbf{b}^T = [j \; -2* j \; 4] \), the inner product is given in MATLAB as follows:

\[
\begin{align*}
\text{>> ab = a' * b} \\
\text{>> 2.0000 - 8.0000i}
\end{align*}
\]

Although the semicolon is written as a row vector in the presentation of the vector in MATLAB, it produces a column vector. For example, \( \mathbf{x} = [1; \; 3] \) is a column vector. The prime in MATLAB indicates a transposition, which means that it changes a row vector to a column vector and a column vector to a row vector.

Based on the inner product of vectors, we can find the square of a vector’s magnitude using the inner product. The magnitude square and the magnitude of a vector (column) \( \mathbf{a} \) are, respectively, given by

\[
\begin{align*}
|\mathbf{a}|^2 &= \mathbf{a}^T \mathbf{a} = (a_1^2 + a_2^2 + \cdots + a_n^2) \\
\|\mathbf{a}\| &\equiv |\mathbf{a}| = (a_1^2 + a_2^2 + \cdots + a_n^2)^{1/2} = \left( \sum_{i=1}^{n} a_i^2 \right)^{1/2}
\end{align*}
\]

The magnitude is also known as one of the definitions of the norm of a vector and it is given by the mathematical identification \( \|\mathbf{a}\| \). In this text, we use as a norm the metric equivalent, the square root of the squares of the elements of vectors; thus, both are equivalent.

In MATLAB, the above operations are given as follows:

\[
\begin{align*}
\text{>> mas = a' * a;} \% \text{mas is the sum of the squares of the individual vector elements;} \\
\text{>> ma = sqrt(a' * a);} \% \text{ma is the magnitude of the vector a; sqrt( ) is} \\
\text{>> a MATLAB function providing the square root of} \\
\text{>> \% the number in parentheses;} \\
\text{or} \\
\text{>> mas = sum(a.*a);} \% \text{a.*a is a vector obtained by multiplying the} \\
\text{>> \% corresponding elements and the sum( ) adds them;} \\
\text{>> \% the dot,, indicates multiplication of corresponding} \\
\text{>> \% elements; the same can be done for two} \\
\text{>> \% different vectors of the same number of} \\
\text{>> \% elements;} \\
\text{>> ma = sqrt(sum(a.*a));} \% \text{ma is the magnitude of the vector a;}
\end{align*}
\]
1.1.4 Distance between Two Vectors

**Definition 1.1.1:** Let \( \mathbf{a} = [a_1, a_2 \ldots a_n] \) and \( \mathbf{b} = [b_1, b_2 \ldots b_n] \) be any two \( n \)-dimensional vectors of equal length. Then, the distance (nonnegative number) between these two vectors is given by the following expression:

\[
d(\mathbf{a}, \mathbf{b}) = \| \mathbf{a} - \mathbf{b} \| = \left[ (a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2 \right]^{1/2}
\]

(1.5)

\[
= \left[ \sum_{i=1}^{n} (a_i - b_i)^2 \right]^{1/2}
\]

\[\triangle\]

**Example 1.1.1**

If the scalar \( k > 0 \), find the magnitude of the vector \( k\mathbf{a} \).

**Solution:**

\[
\|k\mathbf{a}\| = \|k\mathbf{a}\| = \left[ (k^2a_1)^2 + (k^2a_2)^2 + \cdots + (k^2a_n)^2 \right]^{1/2}
\]

\[
= k^2 \left( a_1^2 + a_2^2 + \cdots + a_n^2 \right)^{1/2} = k \|\mathbf{a}\| = k \|\mathbf{a}\| \]

The equivalent MATLAB format is as follows:

\[
>> c = \text{sqrt}(\text{sum}(k*(a'*a))); \% c = number;
\]

\[\square\]

1.1.5 Mean Value of a Vector

**Definition 1.1.2 (mean vector):** The mean value of an \( n \)-element vector is defined by

\[
\overline{\mathbf{a}} = \frac{1}{n}(a_1 + a_2 + \cdots + a_n) = \frac{1}{n} \sum_{i=1}^{n} a_i
\]

(1.6)

\[\triangle\]

If the values of the elements come out from a space of random numbers, then the mean value defined above gives an approximate value of the exact mean defined in the space. More will be discussed about random numbers and variables in Chapter 4.

Let us assume that a person picks up 9 resistors of 5 ohms each from a box containing a large number of resistors. A second person picks up another 9 resistors. Using MATLAB, we obtain the following:

\[
>> a = 5+\text{randn}(1,9); \% \text{randn()} \text{ is a MATLAB function creating a set of}
\]

\[
\text{numbers with mean value 0 and are normally (Gaussian)}
\]

\[
\text{\% distributed;}
\]
Adaptive Filtering

>> \%a is a row vector with 9 elements with mean value close to 5 and normally distributed;

>> 6.1892 4.9624 5.3273

>>b = 5+randn(1,9);

>>5.1746 4.8133 5.7258 4.4117 7.1832 4.8636
>>5.1139 6.0668 5.0593

>>ma = mean(a); \%mean(a) gives the mean value of the elements of the vector a, mean() is a MATLAB function;

>>4.9820

>>mb = mean(b);

>>5.3791

Another way to find the mean value of a vector is the following:

>>ma = sum(a)/length(a); \%length(a) is a MATLAB function giving the number of the elements of the vector;

Observe that the two means are not close to each other and to the mean value 5, which we would expect. This is due to the small number we picked from the space of normally distributed numbers. If we had considered a number greater than 1000, the mean value would have been closer to 5 and closer to each other. Figure 1.2 shows the distribution of 7000 numbers having a mean value close to 5. The mean value of these 7000 numbers is 5.0052. If we look at the mean value of another set of 7000 normally distributed numbers, we would find the mean value to be very close to 5 but not exactly the same as the one above. Therefore, if we repeat this experiment many times and plot the distribution of the means, we will observe a normal distribution around the value 5, but a very narrow normal distribution.

The normal distribution in Figure 1.2 may be obtained as follows:

>>x = 5+randn(1,7000); \%we must add the semicolon because otherwise all 7000 numbers will appear in the command window;
>>hist(x,40); \%hist(x,40) is a MATLAB function which produces the distribution, here the normal, from the vector
>> \%x with 40 bins;
>> \%the color is always blue;
>>h = findobj(gca,'Type','path');
>>set(h,'FaceColor','w','EdgeColor','k'); \%creates a figure with \%white, \'w\', color
>> \%the bins and
Vectors

> their lines black, 'k';
> other colors are: green, 'g'; red, 'r';
> yellow, 'y';
> cyan, 'c'; blue, 'b';
> magenta, 'm';
>> xlabel('Values of the numbers produced');
>> ylabel('Number of values falling within a bin');

1.1.6 Direction Cosines

A three-dimensional vector \( \mathbf{a} \), its components, and the three unit vectors along the three axes (x, y, and z) are shown in Figure 1.3. The cosines of the three angles created by the vector and the three axes are as follows:

\[
\cos \theta_1 = \frac{a_1}{\| \mathbf{a} \|}; \quad \cos \theta_2 = \frac{a_2}{\| \mathbf{a} \|}; \quad \cos \theta_3 = \frac{a_3}{\| \mathbf{a} \|}; \quad \| \mathbf{a} \| = \| \mathbf{a} \| 
\]  

(1.7)

Example 1.1.2

We will find the cosines of the vector \( \mathbf{a}^T = \begin{bmatrix} 1 & 1/2 \end{bmatrix} \); we will also write the vector in mathematical form: \( \mathbf{a}^T = (1, 1/2) \).
Adaptive Filtering

Solution: The direction cosines and their sum square are as follows:

\[
\cos \theta_1 = \frac{a_1}{\|a\|} = \frac{1}{\sqrt{1^2 + (1/2)^2}} = \frac{2}{\sqrt{5}}, \quad \theta_1 = 26.5^\circ; \quad \cos \theta_2 = \frac{a_2}{\|a\|} = \frac{1/2}{\sqrt{5}/2}
\]

\[
= \frac{1}{\sqrt{5}}, \quad \theta_2 = 63.5^\circ;
\]

\[
\cos^2 \theta_1 + \cos^2 \theta_2 = \left(\frac{2}{\sqrt{5}}\right)^2 + \left(\frac{1}{\sqrt{5}}\right)^2 = 1
\]

We can use the following MATLAB program:

```matlab
>>a = [1 1/2];
>>cos1 = a(1)/sqrt(sum(a.*a));%cosine of theta one;
>>th1r = acos(cos1);%theta one in radians;
>> %acos() is a MATLAB function that gives
>> %the inverse of a
>> %cosine; similar: asin(), atan();
>>th1 = th1r*180/pi;%theta one in degrees;
>>th1
>>26.5651;%theta one in degrees;
```

The cosine of the angle between two vectors \( \mathbf{a} \) and \( \mathbf{b} \) is

\[
\cos \theta = \frac{\mathbf{a}^T \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} \quad (1.8)
\]

Because \( |\cos \theta| \leq 1 \), the inner product between the two vectors satisfies the Cauchy–Schwarz inequality:

\[
|\mathbf{a}^T \mathbf{b}| \leq \|\mathbf{a}\| \|\mathbf{b}\| \quad (1.9)
\]
Another useful inequality is the following:

\[2|a^Tb| \leq \|a\|^2 + \|b\|^2\]  
(1.10)

### 1.1.7 The Projection of a Vector

To find the projection point \( p \) of the vector \( b \) on \( a \) (Figure 1.4), we observe that the magnitude of the vector \( p = \alpha a \) is some multiple of the vector \( a \). The line from the end of vector \( b \) to vector \( a \), which ends at the end point of the vector \( p \), is perpendicular to vector \( a \). Hence, we have

\[ p = \alpha a, \quad (b - \alpha a) \perp a \quad \text{or} \quad a^T(b - \alpha a) = 0 \quad \Rightarrow \quad \alpha = \frac{a^Tb}{a^Ta} \]  
(1.11)

Therefore, the projection of \( b \) onto the line through 0 and the end of vector \( a \) is

\[ p = \alpha a = \frac{a^Tb}{a^Ta} \]  
(1.12)

**Example 1.1.3**

Find the orthogonal projection of \( b^T = [2 \ 7 \ 2] \) onto vector \( a^T = [5 \ 7 \ 4] \).

**Solution:** The magnitude of the projected vector is \( \|b_p\| = (a^Tb)/(a^Ta)^{1/2} = (10 + 49 + 8)/(25 + 49 + 16)^{1/2} = 7.0624 \). The direction is given by the unit vector \( a/\|a\| = (5,7,4)/(25 + 49 + 16)^{1/2} = [0.5270 \ 0.7379 \ 0.4216] \). Hence, \( b_p = b_p[0.5270 \ 0.7379 \ 0.4216] = 7.0624[0.5270 \ 0.7379 \ 0.4216] = [3.7219 \ 5.2113 \ 2.9775] \) and also \( b_p = (a^Tb)/(a^Ta)a = 0.7444a \).

**Definition 1.1.3 (projection theorem):** Let \( \mathcal{V} \) be a vector space, \( S \) be a subspace of \( \mathcal{V} \), and \( v \) be any vector in \( \mathcal{V} \). The vector \( v_p \in S \) is a minimizing vector if and only if \( e = v - v_p \) is orthogonal to \( S \) (\( e \) is perpendicular to all vectors in \( S \)).

![Figure 1.4](image-url)
1.1.8 Linear Transformations

Functions from vector spaces to vector spaces are called transformations (or maps or operators).

Examples of transformations are as follows:

**Shift**

\[ L(x) = x + a \]

Let \( a = [2 \ 1]^T \) and \( x = [1 \ 1]^T \); the shift of \( x \) is shown in Figure 1.5 (shown by the circle). \( L \) is an operator, indicating a transformation.

**Rotation and shift**

Let the line \( l \) be described by \( 0.5[1 \ 1]^T + (1-0.5)[1 \ 2]^T \) or \( [0.5 \ 0.5]^T + [-0.5 \ 1]^T \). If \( L = [1 \ 1; 1 \ -1] \), a \( 2 \times 2 \) matrix, then the image of the line \( l \) is

\[
L[l] = \begin{bmatrix} 1 & 1 \\ -1 & 0.5 \end{bmatrix} \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.5 \\ -1.5 \end{bmatrix}
\]

The original line and its image, after the transformation, are shown in Figure 1.6. Observe that \( 0.5 + (1-0.5) = 1 \).

**Axis**

Let the operator be of the form:

\[
L = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}
\]

The basis for a rectangular coordinate system is \( \{e_1, e_2\} = \{[1 \ 0]^T, [0 \ 1]^T\} \). After applying the operator, we have

![Figure 1.5](image-url)
FIGURE 1.6

FIGURE 1.7

\[ L\{e_1\} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \]

and

\[ L\{e_2\} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \]

Therefore,

\[ L\{ae_1 + be_2\} = aL\{e_1\} + bL\{e_2\} = a\begin{bmatrix} 2 \\ 1 \end{bmatrix} + b\begin{bmatrix} 1 \\ 2 \end{bmatrix} \]

Figure 1.7 shows the transformation for \( a = 2 \) and \( b = 1 \).

1.2 LINEAR INDEPENDENCE, VECTOR SPACES, AND BASIS VECTORS

A set of \( n \) vectors \( v_1, v_2, \ldots, v_n \) is said to be **linearly independent** if

\[ \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n = 0 \]  \hspace{1cm} (1.13)
implies that all the $\alpha_i$'s are 0. If a set of nonzero $\alpha_i$'s can be found so that (1.13) holds, then the set of the vectors is said to be **linearly dependent**. If $v_1, v_2, \ldots, v_n$ is a set of linearly dependent vectors, then one of the vectors can be expressed as a linear combination of the remaining vectors. Hence, we have

$$v_1 = \beta_2 v_2 + \cdots + \beta_n v_n$$

The $\beta_i$'s are some set of scalars. If a set of vectors is $n$-dimensional, there can be only $n$ independent vectors. In other words, any set that contains more than $n$ vectors will always be linearly dependent.

**Example 1.2.1**

Find out if the vectors $a^T = [1 \ 4 \ 1]$ and $b^T = [1 \ 0 \ 1]$ are linearly dependent or independent.

*Solution*: Applying (1.13), we obtain

$$\begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 \alpha_1 + 1 \alpha_2 \\ 4 \alpha_1 + 0 \alpha_2 \\ 1 \alpha_1 + 1 \alpha_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Two vectors are equal if their corresponding elements are equal. Hence, $\alpha_1 + \alpha_2 = 0$ and $4\alpha_1 = 0$, which results in $\alpha_1 = \alpha_2 = 0$. This indicates that the vectors are independent. If we add the vector $v_3^T = [0 \ 1 \ 0]$, we find that the vectors are dependent (see Problem 1.2.1).

Given a set of $N$ vectors $V = \{v_1, v_2, v_3, \ldots, v_N\}$, consider the set of all vectors $V$ that may be formed by the linear combination of the vectors $v = \sum_{i=1}^{N} \alpha_i v_i$ for different sets of $\alpha_i$'s. This set forms a vector space, and the vectors $v_i$ are said to span the **vector space** $\mathbb{V}$. In addition, if the vectors $v_i$ are linearly independent, then it is said that they form a **basis** for the vector space. The number $N$ is referred to as the **dimension** of the space. For example, the set of all real vectors of the form $v = [v_1, v_2, v_3, \ldots, v_N]^T$ forms an $N$-dimensional vector space, denoted by $\mathbb{R}^N$, which is spanned by the basis vectors:

$$e_1 = [1 \ 0 \ 0 \ \cdots \ \ 0]^T$$

$$e_2 = [0 \ 1 \ 0 \ \cdots \ \ 0]^T$$

$$\vdots$$

$$e_N = [0 \ 0 \ 0 \ \cdots \ \ 1]^T$$
On this basis (basis is not unique), any vector $v$ in $\mathbb{R}^N$ may be uniquely decomposed as follows:

$$v = \sum_{i=1}^{N} v_i e_i$$

**Definition 1.2.1:** Let $\mathcal{S}$ and $\mathcal{S}'$ denote any two vector spaces. Then the following definitions apply:

1. The vector space consisting of all vectors common to both vector spaces is called the **intersection** of the two spaces and is written as $\mathcal{S} \cap \mathcal{S}'$ (see Figure P1.2.1).
2. The vector space consisting of all vectors $a + b$, where $a$ lies in $\mathcal{S}$ and $b$ in $\mathcal{S}'$, is called the sum or the **union** of the two spaces and is written as $\mathcal{S} \cup \mathcal{S}'$.

An important special case occurs when the two spaces have in common only the $0$ vector, which is represented by $\mathcal{S} \cap \mathcal{S}' = 0$. This type of union is known as the **direct sum** of the two spaces and is written as $\mathcal{S} \oplus \mathcal{S}'$. From Figure P1.2.1, we observe that

$$\dim(\mathcal{S} \oplus \mathcal{S}') = \dim(\mathcal{S}) + \dim(\mathcal{S}')$$

### 1.2.1 Orthogonal Basis Vectors

**Definition 1.2.2:** The vector spaces $\mathcal{S}$ and $\mathcal{S}'$ are said to be **mutually orthogonal** if every vector in $\mathcal{S}$ is orthogonal to every vector in $\mathcal{S}'$.

**Definition 1.2.3:** Let $\mathcal{S}$ be a subspace of $\mathcal{S}^{(n)}$ [$\mathcal{S} \in \mathcal{S}^{(n)}$]. Then the set of all vectors in $\mathcal{S}^{(n)}$ that are perpendicular to $\mathcal{S}$ is known as the **orthogonal complement** of $\mathcal{S}$, denoted by $\mathcal{S}^\perp$.

### PROBLEMS

1.1.1 Show that the inner product is distributive over addition (subtraction).
1.1.2 Show that if a vector is multiplied by a scalar, its magnitude changes and not its direction.
1.1.3 Prove (1.8).
1.1.4 Prove (1.10).
1.2.1 Show that the three vectors in Example 1.2.1 are dependent.
1.2.2 Find the dimensions of the general union $\mathcal{S} \cup \mathcal{S}'$. 
**Adaptive Filtering**

**FIGURE P1.2.1**

**HINTS–SUGGESTIONS–SOLUTIONS**

1.1.1  
\[
\begin{align*}
a^T(b \pm c) &= a_1(b_1 \pm c_1) + a_2(b_2 \pm c_2) + \cdots + a_n(b_n \pm c_n) \\
&= (a_1b_1 \pm a_2b_2 \pm \cdots \pm a_nb_n) + (a_1c_1 \pm a_2c_2 \pm \cdots \pm a_nc_n) = a^Tb \pm a^Tc
\end{align*}
\]

1.1.2 The magnitude of the vector \(\alpha b\) is
\[
\|\alpha b\| = \left[ (\alpha b_1)^2 + (\alpha b_2)^2 + \cdots + (\alpha b_n)^2 \right]^{1/2}
\]
\[
= \alpha \left[ (b_1)^2 + (b_2)^2 + \cdots + (b_n)^2 \right]^{1/2} = \alpha \|b\|
\]

If \(\alpha > 0\), we have extension, and if \(\alpha < 1\), we have contraction.

1.1.3 From the law of trigonometry of a triangle with two sides \(a\) and \(b\) and their angle between them \(\theta\), we have
\[
|b - a|^2 = |b|^2 + |a|^2 - 2|b||a| \cos \theta.
\]
If the angle is 90°, we have the Pythagoras theorem. We can also write the above expression in the following form:
\[
|b - a|^2 = (b - a) \cdot (b - a) = b^2 - 2b^T a + a^2 = b^2 + a^2 - 2b^T a + 2|b||a| \cos \theta
\]

Canceling the same quantities from both sides of the above equation, we recognize (1.8).

1.1.4 If \(a = \pm b\), the equality holds. For any two vectors, the magnitude of their sum or difference is a positive scalar. Hence, \(|b \pm a|^2 \geq 0\). Expanding the norm (see Problem 1.1.3), we obtain
\[
|b \pm a|^2 = |b|^2 + |a|^2 \pm 2|b||a| \cos \theta = |b|^2 + |a|^2 \pm 2a^Tb \geq 0
\]

From the above expression, the inequality easily follows.
1.2.1 By equating the vectors, we obtain the set of equations
\[ \alpha_1 + \alpha_2 + 0\alpha_3 = 0; \ 4\alpha_1 + 0\alpha_2 + \alpha_3 = 0 \] and \[ \alpha_1 + \alpha_2 + 0\alpha_3 = 0. \] Solving the set, we obtain \( \alpha_2 = -\alpha_1, \alpha_3 = -4\alpha_1. \) Setting arbitrarily \( \alpha_1 = 1, \) we obtain \( \mathbf{a} = \mathbf{b} + 4\mathbf{c}. \)

1.2.2

\[
\dim(\mathcal{S} \cup \mathcal{S}) = (s - m) + m + (t - m) = s + t - m
\]

or
\[
\dim(\mathcal{S} \cup \mathcal{S}) = \dim(\mathcal{S}) + \dim(\mathcal{S}) - \dim(\mathcal{S} \cap \mathcal{S})
\]
2 Matrices

2.1 INTRODUCTION

A generalization of a vector, that is, an ordered array of numbers, determines the magnitude and direction of the vector. A further generalization is that of an \( n \times m \) matrix \( A = (a_{ij}) \), which is shown below:

\[
A = \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1m} \\
a_{21} & a_{22} & \cdots & a_{2m} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nm}
\end{bmatrix}
\]  

(2.1)

with a typical element \( a_{ij} \), where \( i = 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, m \).

2.2 GENERAL TYPES OF MATRICES

We shall present two- and three-element matrices for convenience. However, the properties are identical for any size.

2.2.1 DIAGONAL, IDENTITY, AND SCALAR MATRICES

Diagonal: \( A = \begin{bmatrix} a_{11} & \cdots & 0 \\ 0 & \ddots & \vdots \\ \cdots & \cdots & a_{nn} \end{bmatrix} \); Identity: \( I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \); Scalar: \( A = \begin{bmatrix} a \\ 0 \\ a \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = aI \)

(2.2)

We shall also write diag \( \{a_{11}, a_{22}, \ldots, a_{nn}\} \) in short.

2.2.2 UPPER AND LOWER TRIANGULAR MATRICES

Lower triangular: \( A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \); Upper triangular: \( A = \begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix} \)

(2.3)
2.2.3 **Symmetric and Exchange Matrices**

Symmetric: \[ A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \]

\[ A = A^T \]  

(2.4)

Exchange: \[ J = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \]

2.2.4 **Toeplitz Matrix**

An \( n \times n \) matrix is said to be Toeplitz if all the elements along each of the diagonals have the same value, that is, \( a_{ij} = a_{i+j,j+1} \) for all \( i < n \) and \( j < n \).

Toeplitz: \[ A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{11} & a_{12} \\ a_{23} & a_{21} & a_{11} \end{bmatrix} \]  

(2.5)

2.2.5 **Hankel and Hermitian**

A matrix is said to be Hankel if its elements along the diagonals that are perpendicular to the main diagonal are equal. A complex matrix is said to be Hermitian if it is conjugate symmetric.

Hankel: \[ A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{13} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \]

Hermitian: \[ A = \begin{bmatrix} 1 & 1+2j & -j \\ 1-2j & 2 & 4j \\ j & -4j & 6 \end{bmatrix} = (A^*)^T = A^H \]  

(2.6)

>>a = [1 1+2*j -j;1-2*j 2 4*j;j -4*j 6];
>>conj(transpose(a))
1.0000 1.0000 + 2.0000i 0 - 1.0000i
1.0000 - 2.0000i 2.0000 0 + 4.0000i
0 + 1.0000i 0 - 4.0000i 6.0000

2.3 **Matrix Operations**

Table 2.1 presents some of the algebraic operation of matrices together with the MATLAB operations.
Example 2.3.1

Find the solution of the system using the backslash approach in MATLAB and regular inversion approach.

\[ 5x + 2y = 1 \]
\[ 2x + 2y = 4 \]

**Solution:** If we subtract the second equation from the first, we find \( 3x = -3 \) or \( x = -1 \). Substituting \( x = -1 \) in any of the two equations, we find \( y = 3 \), and thus, the
unknown \( \mathbf{b} = [x \ y]' = [-1 \ 3]' \). We obtain the same value of \( \mathbf{b} \) using MATLAB backslash approach; if we set

\[
\mathbf{A} = \begin{bmatrix}
5 & 2 \\
2 & 2
\end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix}
1 \\
4
\end{bmatrix}
\]

then in MATLAB we write \( \mathbf{b} = \mathbf{A}^{-1} \mathbf{c} \). Since the solution of the above system is equal to \( \mathbf{b} = \mathbf{A}^{-1} \mathbf{c} \), in MATLAB Command window we write \( \mathbf{b} = \text{inv(}A\text{)}*\mathbf{c} \) to find \( \mathbf{b} \).

\[\blacksquare\]

### Example 2.3.2

Let \( \mathbf{A} = \begin{bmatrix} 1 & -1 & 1 \\ -3 & 2 & 1 \end{bmatrix} \) and \( \mathbf{B} = \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix} \). Find \( \mathbf{AB}, (\mathbf{AB})', \) and \( \mathbf{B}' \mathbf{A}' \).
Solution:

\[ AB = \begin{bmatrix} 1 \cdot 1 + (-1) \cdot 2 + 1 \cdot 1 & 1 \cdot 2 + (-1) \cdot 4 + 1 \cdot 2 \\ -3 \cdot 1 + 2 \cdot 2 + 1 \cdot 1 & -3 \cdot 2 + 2 \cdot 4 + 1 \cdot 2 \\ -2 \cdot 1 + 1 \cdot 2 + 0 \cdot 1 & -2 \cdot 2 + 1 \cdot 4 + 0 \cdot 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 0 & 0 \end{bmatrix} \]

\[(AB)^T = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 4 & 0 \end{bmatrix} \]

\[ B^T A^T = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \\ -3 & 2 & 1 \\ -2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 4 & 0 \end{bmatrix} = (AB)^T \]

\[ \begin{align*}
A &= \begin{bmatrix} 1 & -1 & 1 \\ -3 & 2 & 1 \\ -2 & 1 & 0 \end{bmatrix}, & B &= \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 1 & 2 \end{bmatrix} \\
\Rightarrow A \times B &= \begin{bmatrix} 0 & 0 \\ 2 & 4 \\ 0 & 0 \end{bmatrix} \\
\Rightarrow (A \times B)' &= \begin{bmatrix} 0 & 2 & 0 \\ 0 & 4 & 0 \end{bmatrix} \\
\Rightarrow B' \times A' &= \begin{bmatrix} 0 & 2 & 0 \\ 0 & 4 & 0 \end{bmatrix}
\end{align*} \]

We can invert a matrix by using the MATLAB function, \( \text{inv}(A) \).

2.4 DETERMINANT OF A MATRIX

2.4.1 Definition and Expansion of a Matrix

The determinant of a square matrix is written as follows:

\[ \det(A) = |A| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}; \quad |A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} - a_{13}a_{21}a_{32} + a_{13}a_{22}a_{31} \] (2.7)

The determinant of a \( 3 \times 3 \) matrix, which is true for any dimensions, is given by

\[ \begin{vmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{vmatrix} = (-1)^{1+1} d_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} + (-1)^{1+2} d_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + (-1)^{1+3} d_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \]

\[ + (-1)^{1+3} d_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \] (2.8)
Note: In the above $3 \times 3$ determinant, we selected to expand the first row, but we could have taken any row or column. The exponents correspond to the row and column numbers that correspond to the element of the row we select. The remaining rows and columns produce the minor matrices whose determinant is taken.

Note: The MATLAB function for finding the determinant is `det(A)`.

Note: An $n \times n$ matrix $A$ can be inverted to find $A^{-1}$, if and only if $\det(A) \neq 0$. In MATLAB we write `inv(A)`.

Additional properties of the determinant are given below, assuming that both, $A$ and $B$, are $n \times n$ matrices.

$$
\det(AB) = \det(A) \det(B) = \det(B) \det(A)
$$

$$
\det(A^T) = \det(A)
$$

$$
\det(aA) = a^n \det(A)
$$

$$
\det(A^{-1}) = \frac{1}{\det(A)} \quad (A \text{ must be invertible})
$$

2.4.2 Trace of a Matrix

A trace of an $n \times n$ matrix $A$ is given by

$$
\text{tr}(A) = \sum_{i=1}^{n} a_{ii}; \quad \text{tr}(A^{-1}BA) = \text{tr}[A^{-1}(BA)] = \text{tr}[(BA)A^{-1}]
$$

$$
= \text{tr}[(B(AA^{-1})] = \text{tr}(B)
$$

In the Command window of MATLAB, we write `trace(A)` to obtain the trace of the matrix $A$.

2.4.3 Inverse of a Matrix

**Definition 2.4.1:** An $n \times n$ matrix $A$ is **nonsingular**, if it possesses a inverse or an equivalent its determinant is not 0, otherwise it is known as **singular**. Its inverse $A^{-1}$ has the property

$$
A^{-1}A = AA^{-1} = I
$$

where:

$I$ is the identity matrix

Additional properties of invertible matrices are as follows:

$$
(AB)^{-1} = B^{-1}A^{-1}; \quad (A^{-1})^T = (A^T)^{-1}; \quad (A^{-1})^{-1} = A; \quad |A^{-1}| = \frac{1}{|A|}
$$
The inverse of a nonsingular matrix can be computed by the formula:

\[ A^{-1} = \frac{\text{adj}(A)}{|A|}, \quad \text{adj}(A) = C^T, \quad C_{ij} = \text{cofactor} = (-1)^{i+j} M_{ij} \quad (2.13) \]

where:
- \( C_{ij} \) is known as the **cofactor** of the element \( a_{ij} \)
- \( M_{ij} \) is the determinant of the **minor** matrix resulting by eliminating the \( i \) row and the \( j \) column of the matrix \( A \)

### Example 2.4.1

Find the inverse of the matrix \( A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 1 \\ 3 & 4 & 5 \end{bmatrix} \).

**Solution:** The determinant of the matrix is 6. The first column cofactors are as follows:

\[ c_{11} = (-1)^{1+1} |M_{11}| = + \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix} = 4 \]
\[ c_{21} = (-1)^{2+1} |M_{21}| = - \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix} = 2 \]
\[ c_{31} = (-1)^{3+1} |M_{31}| = + \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = 2 \]

Following the same procedure for the rest of the cofactors, we obtain

\[ C = \begin{bmatrix} 4 & 7 & 8 \\ 2 & -4 & 2 \\ 2 & 5 & -4 \end{bmatrix}; \quad A^{-1} = \frac{\text{adj}(A)}{|A|} = \frac{C^T}{6} = \begin{bmatrix} -0.6667 & 0.3333 & 0.3333 \\ -1.1667 & -0.6667 & 0.8333 \\ 1.3333 & 0.3333 & -0.6667 \end{bmatrix} \]

```matlab
>> a = [1 2 3;2 0 1;3 4 5];
>> inv(a)
an =
-0.6667 0.3333 0.3333
-1.1667 -0.6667 0.8333
1.3333 0.3333 -0.6667
```
2.5 LINEAR EQUATIONS

A system of equations of the form

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m &= b_2 \\
    \vdots \notag \\
    a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m &= b_n
\end{align*}
\]

which can be written in the matrix form:

\[
A \begin{bmatrix} x_1 \\
                 x_2 \\
                 \vdots \\
                 x_m \end{bmatrix} = \begin{bmatrix} b_1 \\
                                b_2 \\
                                \vdots \\
                                b_n \end{bmatrix}
\]

(2.14)

2.5.1 SQUARE MATRICES \((n \times n)\)

The unknown vector \(x\) can be found using inverse matrix \(A\), assuming that \(A\) has nonzero determinant (nonsingular matrix). Hence,

\[
x = A^{-1}b
\]

(2.15)

If \(A\) is singular, then there may be no solution (the equation is inconsistent) or many solutions.

There is a convenient way to represent the vector \(b\) as a linear combination of the columns of the matrix \(A\) as follows:

\[
b = \sum_{i=1}^{m} x_i a_i
\]

(2.16)

**Example 2.5.1 (Regression Line)**

Let us consider the data on students’ weight and height given in the following table:

<table>
<thead>
<tr>
<th>Student number:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight ((y)):</td>
<td>152</td>
<td>164</td>
<td>167</td>
<td>156</td>
<td>150</td>
</tr>
<tr>
<td>Height ((x)):</td>
<td>157</td>
<td>149</td>
<td>160</td>
<td>151</td>
<td>144</td>
</tr>
</tbody>
</table>

We are requested to plot a straight line that is the most representative of the random variations of these two measurements (Figure 2.1).

**Solution:** The requested line is known as linear regression line, and for each pair of values, the line is defined by the equation:

\[
y_i = b_0 + b_1x_i + v_i
\]
Matrices

We, therefore, must find the unknowns $b_0$ and $b_1$ so that we find the appropriate line. The constants $v_i$'s are the errors introduced in the data.

To determine the $b$'s, we must use the least squares method such that the quantity

$$\sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2$$

becomes the smallest possible. Therefore, we write

$$\frac{\partial}{\partial b_0} \left[ \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \right] = 0 \text{ or } nb_0 + \left( \sum_{i=1}^{n} x_i \right) b_1 = \sum_{i=1}^{n} y_i$$

$$\frac{\partial}{\partial b_1} \left[ \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \right] = 0 \text{ or } \left( \sum_{i=1}^{n} x_i \right) b_0 + \left( \sum_{i=1}^{n} x_i^2 \right) b_1 = \sum_{i=1}^{n} x_i y_i$$

The above two linear equations in $b_0$ and $b_1$ are called the normal equations. The above equations take the matrix form:

$$\begin{bmatrix}
\sum_{i=1}^{n} x_i \\
\sum_{i=1}^{n} x_i^2
\end{bmatrix}
\begin{bmatrix}
b_0 \\
b_1
\end{bmatrix} =
\begin{bmatrix}
\sum_{i=1}^{n} y_i \\
\sum_{i=1}^{n} x_i y_i
\end{bmatrix}$$

**FIGURE 2.1**

We, therefore, must find the unknowns $b_0$ and $b_1$ so that we find the appropriate line. The constants $v_i$'s are the errors introduced in the data.

To determine the $b$'s, we must use the least squares method such that the quantity

$$\sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2$$

becomes the smallest possible. Therefore, we write

$$\frac{\partial}{\partial b_0} \left[ \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \right] = 0 \text{ or } nb_0 + \left( \sum_{i=1}^{n} x_i \right) b_1 = \sum_{i=1}^{n} y_i$$

$$\frac{\partial}{\partial b_1} \left[ \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \right] = 0 \text{ or } \left( \sum_{i=1}^{n} x_i \right) b_0 + \left( \sum_{i=1}^{n} x_i^2 \right) b_1 = \sum_{i=1}^{n} x_i y_i$$

The above two linear equations in $b_0$ and $b_1$ are called the normal equations. The above equations take the matrix form:

$$\begin{bmatrix}
\sum_{i=1}^{n} x_i \\
\sum_{i=1}^{n} x_i^2
\end{bmatrix}
\begin{bmatrix}
b_0 \\
b_1
\end{bmatrix} =
\begin{bmatrix}
\sum_{i=1}^{n} y_i \\
\sum_{i=1}^{n} x_i y_i
\end{bmatrix}$$
Using the inverse matrix approach and MATLAB, we obtain the values of the \( b \)'s, which are shown in Figure 2.1.

```matlab
>> x = [157 149 160 151 144];
>> y = [152 164 167 156 150];
>> b = inv([5 sum(x); sum(x) sum(x.^2)])*[sum(y) ; sum(x.*y)];
>> %it was found that \( b_0 = 73.4730, b_1 = 0.5541 \);
>> x1 = 130:5:180;
>> y1 = 73.4730+0.5541*x1;
>> plot(x,y,'.k');hold on;plot(x1,y1,'k');
>> xlabel('Height');ylabel('Weight');
```

2.5.2 **Rectangular Matrices** \((n < m)\)

If the rows are fewer than the columns, \( n < m \), indicates that there are fewer equations than unknowns. If the equations are consistent, there are many vectors that satisfy the equations. Specifically, to find a unique solution, the rank of \( A \) must be equal to \( n \). All the \( n \times n \) matrices of the matrix \( A \) must have nonzero matrix. Then the solution (known as the **minimum norm** solution) is as follows:

\[
\begin{align*}
    x_0 &= A^H(AA^H)^{-1}b \\
    &= A^H(AA^H)^{-1}b
\end{align*}
\]  

(2.17)

where:

\( H \) stands for the conjugate transpose of the matrix (Hermitian)

For real matrices, we substitute \( H \) with \( T \) (transpose). The matrix

\[
A^+ = A^H(AA^H)^{-1}
\]

(2.18)

is known as the **pseudoinverse** of the matrix \( A \) for the undetermined problem.

**Example 2.5.2**

Find the solution for the system given below:

\[
\begin{align*}
    x_1 + x_2 + x_3 + x_4 &= 1 \\
    x_1 - x_2 + x_3 - x_4 &= 1
\end{align*}
\]

**Solution:** Any \( 2 \times 2 \) matrix of

\[
A = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1
\end{bmatrix}
\]

has nonzero determinant. This indicates that the rank of \( A \) is 2. Hence, we can use (2.17). Therefore, we obtain
\[ x = A^T (AA^T)^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}^{-1} \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.0 \\ 0.5 \\ 0.0 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \]

\[ A = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}, \quad \text{adj}(A) = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}^T = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}, \quad |A| = 16 \]

\[ A^{-1} = \frac{\text{adj}(A)}{|A|} = \frac{1}{16} \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 4 \\ 0 \\ 1 \end{bmatrix} \]

\[ \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0.5 \\ 0.5 \end{bmatrix} \]

\[ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}; 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}; \]

\[ >> A = [1 1 1 1;1 -1 1 -1]; \]

\[ >> x = A'*inv(A*A')*[1 1]'; \]

\[ x = 0.5000 \\
0.5000 \\
0 \]

\[ \begin{align*}
2.5.3 & \textbf{ Rectangular Matrices (} m < n \textbf{)} \\
& \text{For this case, in general, no solution exists. The equations in this case are } \textit{inconsistent} \text{ and the solution is said to be } \textit{overdetermined}. \text{ In this case, we cannot expand an arbitrary vector } b \text{ as a function of the column vectors of } A \text{ as in (2.16). Our next action is to find the coefficients } x_i \text{ that produce the best approximation to } b: \]

\[ \hat{b} = \sum_{i=1}^{m} x_i a_i \]  

(2.19)

The most common approach is to use the \textit{least squares} solution. This is equivalent to finding a vector \( x \) that minimizes the norm of the error vector:

\[ \|e\|^2 = \|b - Ax\|^2 \]  

(2.20)
As shown in Figure 2.2, the least squares solution has the property that the error

\[ e = b - Ax \]

is \textit{orthogonal} to every vector \( a \) on the plane, which is used in the approximation for \( b \), that is, the columns of \( A \). This orthogonality implies that

\[ A^H e = 0 \quad (2.21) \]

or

\[ A^H e = A^H b - A^H Ax = 0 \quad \text{or} \quad A^H Ax = A^H b \quad (2.22) \]

The above two expressions are known as the \textbf{normal equations}. For real matrices, \( H \) (Hermitian) is substituted by the transposition \( T \). If the columns of \( A \) are linearly independent (\( A \) has full rank), then the matrix \( A^H A \) is invertible and the \textbf{least squares solution} is

\[ x_0 = (A^H A)^{-1} A^H b \quad ; \quad A^+ = (A^H A)^{-1} A^H \quad (2.23) \]

Matrix \( A^+ \) is known as \textbf{pseudoinverse} for the \textbf{overdetermined} system. In addition, the best approximation \( \hat{b} \) to \( b \) is given by the \textbf{projection} of the vector \( b \) onto the subspace spanned by the vectors \( a_i \) (Figure 2.2):

\[ \hat{b} = Ax_0 = A(A^H A)^{-1} A^H b \quad ; \quad \hat{b} = P_A b \quad ; \quad P_A = A(A^H A)^{-1} A^H \quad (2.24) \]
Matrix $P_A$ is known as the projection matrix. The minimum error square is given by (see Problem 2.5.2)

$$\min \| e \|^2 = b^H e = b^H b - b^H A x_0$$

(2.25)

**Example 2.5.3 (Overdetermined Case)**

Find the vector $x_0$ for the system as well as the minimum error. The system is

$$2x_1 - x_2 = 1$$
$$2x_1 + x_2 = 1$$
$$x_1 + x_2 = 1$$

**Solution:** Using MATLAB, we obtain

```matlab
>> a = [2 1; 2 1; 1 1];
>> b = [1; 1; 1];
>> xo = inv(a'*a)*a'*b;
>> bap = a*inv(a'*a)*a'*b;
>> mes = b'*b - b'*bap;
```

The values are $\hat{b} = bap = [1.3333; 0.6667; 0.6667]$; $A^TA = [6 3; 3 6]$; $(A^TA)^{-1} = [0.2222 -0.1111; -0.1111 0.2222]$; $x_o = [0.6667 0]$; $\min(e^2) = 0.3333$.

**2.5.4 Quadratic and Hermitian Forms**

A **quadratic form** of a real symmetric $n \times n$ matrix $A$ is the scalar defined by (see Problem 2.5.3)

$$Q_A(x) = x^T A x = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j$$

(2.26)

$$Q_A(x) = x^H A x = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i^* x_j, \; x = \text{complex}$$

**Example 2.5.4**

Find the projection of the vector $a = [1 \ 4 \ -2]^T$ onto the plane spanned by $x_1 = [1 \ 2 \ 1]^T$ and $x_2 = [2 \ 1 \ 0]^T$.

**Solution:** Using MATLAB, we obtain
TABLE 2.3
Some Additional Properties of Matrices

\[ A^T = A \]
\[ A^T A = I; A^H A = I \]
\[ AB \] is symmetric if \( AB = BA \).
\[ A^{-1} \] is symmetric if \( A \) is symmetric.
\[ A = A^H \]
\[ A^H = A^{-1} \]
\[ A = -A^T \]
The \( n \times n \) matrix \( A \) is skew symmetric.
\[ Q_A(x) = x^T A x = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j \]
\( Q_A \) (scalar) is the \textbf{quadratic form} of a real symmetric \( n \times n \) matrix \( A \).
\[ Q_A(x) > 0 \]
\( A \) is \textbf{positive definite}.
\[ Q_A(x) \geq 0 \]
\( A \) is \textbf{positive semidefinite}. \( Q_A(x) \leq 0 \) is negative semidefinite.
\[ Q_A(x) < 0 \]
\( A \) is \textbf{negative definite}. \( Q_A(x) = 0 \) is negative definite.

**Grammian**

An \( n \times n \) matrix \( A \) is Grammian if it is symmetric and positive semidefinite.

\( A = P^T P \)
If \( P \) is nonsingular, then \( A \) (\( n \times n \)) is positive definite.

\( P = P^2 \)
\( P \) is an \textbf{idempotent} matrix and also a \textbf{projection} matrix.

\( P_A = A(A^T A)^{-1} A^T \)
\( A \) is an \( n \times m \) matrix with rank(\( A \)) = \( m < n \). Then, \( P_A \) is idempotent and symmetric and consequently an orthogonal projection matrix.

**Note:** \( T \) can be substituted with \( H \).

\[ a = [1 \ 4 \ -2]^T, \ x_1 = [1 \ 2 \ 1]^T, \ x_2 = [2 \ 1 \ 0]^T \]

\[ X = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 0 \end{bmatrix}, \ X^T X = \begin{bmatrix} 6 & 4 \\ 4 & 5 \end{bmatrix}, \ \left[ X^T X \right]^{-1} = \begin{bmatrix} 0.3571 & -0.2857 \\ -0.2857 & 0.4286 \end{bmatrix} \]

\[ P = X(X^T X)^{-1} X^T = \begin{bmatrix} 0.9286 & 0.1429 & -0.2143 \\ 0.1429 & 0.7143 & 0.4286 \\ -0.2143 & 0.4286 & 0.3571 \end{bmatrix} \]

\[ P = PP, \ \hat{a} = \text{projected} = Pa = \begin{bmatrix} 1.9286 \\ 2.1429 \\ 0.7857 \end{bmatrix} \]
The coefficients of the linear combination of the column vectors of \( X \) that represent
the projected vector of \( a \) is given by
\[
b = (X^T X)^{-1} X^T a = \begin{bmatrix} 0.7857 \\ 0.5714 \end{bmatrix}, \quad a = 0.7857x_1 + 0.5714x_2
\]

The error is given by
\[
e = a - \hat{a} = (I - P)a = \begin{bmatrix} -0.9286 \\ 1.8571 \\ -2.7857 \end{bmatrix}, \quad \hat{a}^T e = 2.2204e - 015 \equiv 0
\]

It is apparent that the error vector is perpendicular to the projected one. The error
vector is also perpendicular to \( x_1 \) and \( x_2 \), which span the plane on which the pro-
jected vector lies. For example, \( x_1^T e = 8.8818e - 016 \) as it was expected.

\[\square\]

### 2.6 Eigenvalues and Eigenvectors

Let \( A \) be an \( n \times n \) matrix and consider the following set of linear equations:
\[
A q = \lambda q \quad \text{or} \quad (A - \lambda I) q = 0
\]

For the above equation to have nonzero vector \( q \), the determinant of the matrix
\( A - \lambda I \) must be 0, \( |A - \lambda I| = 0 \). The determinant gives an \( n \)-th order polynomial
with respect to \( \lambda \), and thus, its solution gives \( n \) roots of the unknown \( \lambda \), \( \lambda_i \) for
\( i = 1, 2, 3, \ldots, n \). These values are known as **eigenvalues** and the corresponding
vectors are the **eigenvectors**.

For example, for a \( 2 \times 2 \) matrix, we write
\[
\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \lambda q_1 \\ \lambda q_2 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]

\[
\begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{or} \quad \det \begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} = 0
\]

which gives us two values for the unknown \( \lambda \), \( \lambda_i \) for \( i = 1, 2 \), which are known as the
**eigenvalues**.

Let us assume that the \( a \)'s in the above expression have the following values:
\( a_{11} = a_{22} = 2, a_{12} = a_{21} = 1 \). Therefore, the above polynomial becomes:
\( \lambda - 4\lambda + 3 = 0 \). Hence, the roots of the polynomial (eigenvalues) are as follows:
\[
\lambda = \frac{4 \pm \sqrt{4^2 - 4 \times 3}}{2 \times 1} \Rightarrow \lambda_1 = 1, \lambda_2 = 3
\]
Adaptive Filtering

>> roots([1 -4 3]); % roots() is a MATLAB function that
% gives the roots of the polynomials;
% observe the arrangement of the coefficients of
% the polynomial;

ans =
 3
 1

2.6.1 Eigenvectors

It is desired to find the eigenvectors of the matrix $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ that has the eigenvalues 1 and 3. From (2.27), we obtain

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = 1 \times \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2q_1 + q_2 \\ q_1 + 2q_2 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

$$\Rightarrow 2q_1 + q_2 = q_1 \Rightarrow q_1 = -q_2 \Rightarrow q_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

The normalized eigenvector $q_1$ corresponding to the eigenvalue $\lambda = 1$ is

$$q_1 = \frac{1}{\sqrt{1^2 + (-1)^2}} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.7071 \\ 0.7071 \end{bmatrix}$$

Following the same procedure, the second eigenvector corresponding to the second eigenvalue $\lambda = 3$ is

$$q_1 = \frac{1}{\sqrt{1^2 + 1^2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}$$

The above results can be found using MATLAB as follows:

>> A = [2 1; 1 2];
>> [V,D] = eig(A)
>> V
V =
-0.7071    0.7071
0.7071    0.7071
>> D
D =
1 0
0 3
Based on the above results, we can analyze the matrix $A$ as follows:

$$A = V D V^{-1}$$

(2.28)

### 2.6.2 Properties of Eigenvalues and Eigenvectors

We will use the following data produced with the help of MATLAB to check some of the properties given below:

```
>> a = [2 1 1; 1 2 1; 1 1 2];
>> [v, d] = eig(a);
>> v
v =
0.4082 0.7071  0.5774
0.4082 -0.7071  0.5774
-0.8165  0    0.5774

>> d
d =
  1.0000  0    0
  0        1.0000 0
  0        0    4.0000
```

**Property 1.** The nonzero eigenvectors $q_1, q_2, \ldots, q_n$ corresponding to distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ are linearly independent.

**Property 2.** The eigenvalues of a Hermitian ($A^H = A; A^H = A^T$) matrix are real (see Problem 2.6.2).

**Property 3.** A Hermitian matrix is positive definite, if its eigenvalues are positive. If its eigenvalues are greater than or equal to 0, the matrix is semidefinite.

**Property 4.** The sum of the $n$ eigenvalues of an $n \times n$ matrix $A$ is equal to the sum of its diagonal terms (or trace of $A$). The product of the $n$ eigenvalues is equal to the determinant of $A$ (see Problem 2.6.3).

```
>> trace(a)
6
>> trace(d)
6.0000
```

**Property 5.** The eigenvectors corresponding to distinct eigenvalues of a Hermitian matrix are orthogonal (see Problem 2.6.4).

**Property 6 (spectral theorem).** Any Hermitian $n \times n$ matrix $A$ may be decomposed as

$$A = Q \Lambda Q^H = \sum_{i=1}^{n} \lambda_i q_i q_i^H$$

(2.29)
where there is a set of orthogonal eigenvectors with their corresponding eigenvalues.

\[
A = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}
\]

\[
[Q, L] = \text{eig}(A)
\]

\[
Q = \begin{bmatrix} 
0.4082 & 0.7071 & 0.5774 \\
0.4082 & -0.7071 & 0.5774 \\
-0.8165 & 0 & 0.5774
\end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 4
\end{bmatrix}
\]

Property 7. A real \( n \times n \) matrix \( A \) having linearly independent eigenvectors with its corresponding eigenvalues can be expressed in the form (see Problem 2.6.5):

\[
A = Q \Lambda Q^{-1}
\]  
(2.30)

Property 8. The inverse of Hermitian matrix \( A \) is of the form (see Problem 2.6.6):

\[
A^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i} q_i q_i^H
\]  
(2.31)

Property 9. \( B \) is an \( n \times n \) matrix with eigenvalues \( \lambda_i \) and eigenvectors \( q_i \).

\( A \) is a matrix related to \( B \) as follows:

\[
A = B + a I
\]
Then, \( A \) and \( B \) have the same eigenvectors and the eigenvalues of \( A \) are \( a + \lambda_j \) (see Problem 2.6.7).

**Property 10.** For a symmetric (eigenvectors form an orthonormal set) positive definite (\( A > 0 \), eigenvalues are positive) matrix \( A \), the equation

\[
x^T A x = 1
\]

defines an ellipse in \( n \) dimensions whose axes are in the direction of the eigenvectors \( q_i \) of \( A \) with the half-length of these axes being equal to \( 1/\sqrt{\lambda_i} \).

**Property 11.** The maximum value of the eigenvalues of an \( n \times n \) matrix \( A \) has the following relations:

\[
\lambda_{\text{max}} \leq \sum_{i=1}^{n} \lambda_i = \text{tr}(A)
\]

\[
\lambda_{\text{max}} \leq \max_{i} \sum_{j=1}^{n} a_{ij}
\]

**Property 12.** The eigenvalues of \( A^k \) are equal to \( \lambda_i^k \) and its eigenvectors are the same. Hence,

\[
A^k = (Q^{-1}AQ)(Q^{-1}AQ)\cdots(Q^{-1}AQ) = Q^{-1}A^kQ
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1.0000 & 0 \\
0 & 0 & 64.0000
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0.0000 \\
0.0000 & 1.0000 & 0.0000 \\
0.0000 & 0 & 64.0000
\end{bmatrix}
\]

**Example 2.6.1**

Identify the ellipse for the matrix \( A =\begin{bmatrix}2 & 1 \\ 1 & 2 \end{bmatrix} \).

**Solution:** The eigenvalues are \( \lambda_1 = 1 \) and \( \lambda_2 = 3 \) and the eigenvectors are \( q_1 = [1 \ -1]^T \) and \( q_2 = [1 \ 1]^T \). Figure 2.3 shows the ellipse with the eigenvectors and the minor and major axes.
2.1.1 Find the matrix $A = J J$.

If $C = \begin{bmatrix} 1 & 1 \\ 3 & 4 \end{bmatrix}$ and $C^{-1}$ = inverse of $C = \begin{bmatrix} 4 & -1 \\ -3 & 1 \end{bmatrix}$, find $CC^{-1}$.

2.4.1 Compute the determinant of the matrix $A$ along the first column and also find the trace.

$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 1 \\ 3 & 4 & 5 \end{bmatrix}$

2.4.2 Compute the inverse of the matrices:

$A = \begin{bmatrix} 1 & -2 \\ 2 & 4 \end{bmatrix}$; $B = \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}$

2.5.1 Find the solution of the system

$\begin{bmatrix} -1 & 0 & 2 & 1 \\ -1 & 1 & 0 & -1 \end{bmatrix} [x_1, x_2, x_3, x_4]^T = [1, 1]^T$

2.5.2 Prove (2.25).
2.5.3 Show that \( A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \) is positive definite and \( B = \begin{bmatrix} 4 & 0 \\ 0 & 0 \end{bmatrix} \) is semidefinite.

2.5.4 Prove that the coefficients of expansion of a vector projected on a subspace is equal to \( b = (X^T X)^{-1} X^T Y \).

2.6.1 Find the eigenvalues and the corresponding eigenvectors for the matrix
\[
A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}
\]

2.6.2 Verify Property 2 of the eigenvalues.

2.6.3 Verify Property 4 of the eigenvalues using a \( 2 \times 2 \) matrix.

2.6.4 Verify Property 5 of the eigenvalues.

2.6.5 Verify Property 7 of the eigenvalues.

2.6.6 Verify Property 8 of the eigenvalues.

2.6.7 Verify Property 9 of the eigenvalues.

**HINTS–SUGGESTIONS–SOLUTIONS**

2.1.1
\[
A = JJ = I \cdot C^{-1} C = I \Rightarrow CC^{-1} C = IC = C
\]

Hence, \( J^{-1} JJ = J^{-1} I = J^{-1} \) or \( J = J^{-1} \).

2.4.1
\[
|A| = (-1)^{1+1}(1) \begin{vmatrix} 0 & 1 \\ 4 & 5 \end{vmatrix} + (-1)^{2+1}(2) \begin{vmatrix} 2 & 3 \\ 4 & 5 \end{vmatrix} + (-1)^{3+1}(3) \begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix} = (0 \cdot 5 - 4 \cdot 1) - 2(2 \cdot 5 - 3 \cdot 4) + 3(2 \cdot 1 - 0 \cdot 3) = 6; \quad \text{tr}(A) = 1 + 0 + 5 = 6
\]

2.4.2
\[A^{-1} = \frac{1}{8} \begin{bmatrix} 4 & 2 \\ -2 & 1 \end{bmatrix} ; \quad \text{det}(B) = 0 \Rightarrow \text{there is no inverse}\]

2.5.1
\[x = [ -0.5000 \ 0.3333 \ 0.3333 \ -0.1667 ]^T\]

2.5.2
\[\min |x|^2 = e^H e = (b^H - x^H A^H) e = b^H e - x^H 0 = b^H b - b^H A x_0\]

2.5.3 For any vector \( x \) different than 0,
\[
Q_A(x) = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + 2x_2^2 > 0
\]

Therefore, matrix \( A \) is nonnegative.
\[
Q_B(x) = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 4x_1^2 > 0.
\]
However, for $x = [0 \ x_2]$, $Q_b = 0$, and thus, $A$ is positive semidefinite.

### 2.5.4

$$e^T X = \left[(I - P)Y\right]^T X = Y^T (I - P)^T X = Y^T (I^T - P^T) X = Y^T X - Y^T PX$$

$$= Y^T X - Y^T X(X^T X)^{-1} X^T X = Y^T X - Y^T X = 0; \ PX = X(X^T X)^{-1} X^T X$$

$$= XX^{-1} X^{-T} X^T X = X; \ \hat{Y} = PY = X(X^T X)^{-1} X^T Y = Xb \Rightarrow b = (X^T X)^{-1} X^T Y$$

The columns of $X$ span the subspace in which the projection $\hat{Y}$ of $Y$ lies.

#### 2.6.1

$$\begin{bmatrix} 2 - \lambda_1 & 1 \\ 1 & 2 - \lambda_2 \end{bmatrix} = (2 - \lambda)^2 - 1 = 0, \ \lambda_1 = 3, \lambda_2 = 1$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} q_{11} \\ q_{12} \end{bmatrix} = \begin{bmatrix} 3q_{11} \\ 3q_{12} \end{bmatrix} \Rightarrow 2q_{11} + q_{12} = 3q_{11}$$

$$\Rightarrow q_{11} = q_{12} \text{ which satisfies the second equation}$$

$$\Rightarrow q_1 = \frac{1}{\sqrt{8}} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \text{normalized eigenvector arbitrarily setting} \ q_{11} = 2$$

Similarly, $q_2 = \frac{1}{\sqrt{8}} \begin{bmatrix} 2 \\ -2 \end{bmatrix}$

#### 2.6.2

$$Aq_i = \lambda_i q_i \Rightarrow q_i^H A q_i = \lambda_i q_i^H q_i \quad (1)$$

$$\Rightarrow \left(q_i^H A q_i\right)^H = \left(\lambda_i q_i^H q_i\right)^H \Rightarrow q_i^H A^H q_i = \lambda^*_i q_i^H q_i$$

$$\Rightarrow q_i^H A q_i = \lambda^*_i q_i^H q_i$$

Comparing with (1) (A is Hermitian) $\lambda_i = \lambda^*_i$

#### 2.6.3

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}; \det \left(\begin{bmatrix} a - \lambda & b \\ c & d - \lambda \end{bmatrix}\right) = \lambda^2 - (\text{tr}) \lambda + \det; \lambda_{1,2} = \frac{\text{tr} \pm [\text{(tr)}^2 - 4 \det]^{1/2}}{2}$$

$$\lambda_1 + \lambda_2 = \frac{\text{tr} + [(\text{tr})^2 - 4 \det]^{1/2}}{2} + \frac{\text{tr} - [(\text{tr})^2 - 4 \det]^{1/2}}{2} = \frac{2\text{tr}}{2} = \text{tr}; \ \lambda_1 \lambda_2 = \det$$
2.6.4
\[ A q_i = \lambda_i q_i, \quad A q_j = \lambda_j q_j \quad (1) \]
\[ \Rightarrow q_j^H A q_i = \lambda_i q_j^H q_i \quad (2) \]
\[ \Rightarrow q_j^H A q_j = \lambda_j q_j^H q_j \quad (3) \]
\[ \Rightarrow \text{conjugate transpose} \quad (3) \]
\[ \Rightarrow q_j^H A q_i = \lambda_i q_j^H q_i \quad (4) (A \text{ is Hermitian and } \lambda_i \text{'s are real}) \]

subtract (4) from (2) \[ \Rightarrow (\lambda_i - \lambda_j) q_j^H q_i = 0 \text{, if } \lambda_i - \lambda_j \neq 0 \]
\[ \Rightarrow q_j^H q_i = 0 = \text{orthogonal} \]

2.6.5
\[ A q_k = \lambda_k q_k, \quad k = 1, 2, \ldots, n \Rightarrow \text{the } n \text{ equations are } A [ q_1, q_2 \cdots q_n ] \]
\[ = [ \lambda_1 q_1, \lambda_2 q_2 \cdots \lambda_n q_n ] \quad (1) \]
\[ \Rightarrow Q = [ q_1, q_2 \cdots q_n ], \Lambda = \text{diag} \{ \lambda_1, \lambda_2, \ldots, \lambda_n \} \]
Then, (1) becomes \[ A Q = Q \Lambda \Rightarrow A Q Q^{-1} = Q \Lambda Q^{-1} \Rightarrow A = Q \Lambda Q^{-1} \]

2.6.6
\[ A^{-1} = (Q \Lambda Q^H)^{-1} = (Q^H)^{-1} \Lambda^{-1} Q^{-1} = Q \Lambda^{-1} Q^H = \sum_{i=1}^{n} \frac{1}{\lambda_i} q_i q_i^H \]

2.6.7
If \( q_i \) is the eigenvector of \( B \), then \( A q_i = B q_i + a q_i = \lambda_i q_i + a q_i = (\lambda_i + a) q_i \)
3 Processing of Discrete Deterministic Signals
Discrete Systems

3.1 DISCRETE-TIME SIGNALS

Discrete-time signals are seldom found in nature. Numerous discrete signals are used to verify results or theories, for example, the daily change of the stock market in economics, the daily maximum temperature at a particular place for the climate change, and so on. However, most of the signals used by engineers and scientists are the outputs of transducers that, by nature, are continuous signals. Therefore, we are forced to digitize these signals and proceed to manipulate them at their digitized form. Hence, a continuous-time signal \( f(t) \) is digitized (sampled by a constant amount of time \( T \)) and becomes a set of numbers \( \{f(nT)\} \). The sampling theorem tells us that for those signals with finite spectrum (band-limited signals) and maximum frequency \( \omega_N \) (known as the Nyquist frequency), we must use a sampling frequency \( \omega_s = 2\pi/T \) or \( T = 2\pi/\omega_s \), which should be twice as large as the Nyquist frequency, or, equivalently, the sampling time \( T \) must be less than one-half of the Nyquist time, \( T_N = 2\pi/\omega_N \). In this chapter, we shall assume that all the detected signals are band limited. This is a reasonable assumption, because almost all of the signals have been transferred through a finite-frequency filter (low-pass filter, prefiltering), transducer.

3.1.1 TIME-DOMAIN REPRESENTATION OF BASIC CONTINUOUS AND DISCRETE SIGNALS

A set of basic continuous and discrete signals is included in Table 3.1.

Example 3.1.1

Draw the signal \( f(nT) = 2\delta(n + 2T) + 2u(n - T) - u(n - 5T) \).

Solution: Figure 3.1 shows the above signal.

◼
3.2 TRANSFORM-DOMAIN REPRESENTATION OF DISCRETE SIGNALS

### 3.2.1 Discrete-Time Fourier Transform
Any nonperiodic discrete-time signal $x(t)$ with finite energy has a discrete-time Fourier transform (DTFT), which is found by an approximation of the Fourier transform (FT). Any discrete signal that comes from discretizing a continuous signal has a periodic FT.

The transform is found as follows:

$$X(e^{j\omega}) = \mathcal{F}\{x(nT)\} = \mathcal{F}\{x(nT)\} = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt = \sum_{n=-\infty}^{\infty} x(nT)$$  \hspace{1cm} (3.1)
where the exact integral from \( nT - T \) to \( nT \) has been replaced by the approximate area \( T \times x(nT) \). When \( T = 1 \), the above equation takes the form:

\[
X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}
\] (3.2)

The relation \( X[e^{j(\omega+2\pi)}] = X(e^{j\omega}e^{j2\pi}) = X(e^{j\omega}) \) indicates that the spectrum of a discrete signal is periodic.

**Example 3.2.1**

Plot the amplitude and phase spectrum of the DTFT for the discrete signal \( x(n) = 0.95^nu(n) \), an exponential one.

**Solution:** The DTFT of the signal is as follows:

\[
X(e^{j\omega}) = \sum_{n=0}^{\infty} (0.95)^n e^{-j\omega n} = \sum_{n=0}^{\infty} (0.95e^{-j\omega})^n = \frac{1}{1-0.95e^{-j\omega}}
\]

The second summation above constitutes a geometric series of the form \( 1 + a + a^2 + a^3 + \cdots = 1/(1-a) \), where, in this case, \( a = 0.95\exp(-j\omega) \). The absolute value of \( a \) must be less than 1. The amplitude and phase are plotted in Figure 3.2a and b, respectively. If we had expended the frequency further, we would have seen the periodic format of the spectrum.

![Figure 3.2](image-url)
As a practical matter, we are always restricted to manipulate only a finite time-length signal. Let us assume that the number of the data in the time sequence is \( N \) at time-length apart. The transform sequence is characterized by the same number \( N \) of frequency data that are \( 2\pi/NT \) apart. Hence, we define the discrete FT (DFT) of a sequence of \( N \) samples \( \{x(nT)\} \) by the relation:

\[
X\left(k\frac{2\pi}{NT}\right) = X(k\Omega_b) = X(e^{j\Omega_b}) = \mathcal{F}_D\{x(nT)\} = T \sum_{n=0}^{N-1} x(nT)e^{-j\Omega_b T n}
\]

(3.3)

where:
- \( N \) is the number of samples (even number)
- \( T \) is the sampling time interval
- \((N-1)T\) is the signal length in the time domain
- \( \Omega_b = (\omega_b/N) = (2\pi/NT) \) is the frequency sampling interval (frequency bin)
- \( e^{-j\Omega_b T} = N \)th principal root of unity

The inverse DFT (IDFT) is given by

\[
x(nT) = \mathcal{F}_D^{-1}\left\{X\left(k\frac{2\pi}{NT}\right)\right\} = \mathcal{F}_D^{-1}\{X(k\Omega_b)\} = \frac{1}{NT} \sum_{k=0}^{N-1} X\left(k\frac{2\pi}{NT}\right)e^{j\frac{2\pi}{N}nk}
\]

(3.4)
For $N = 1$, the above two equations become

$$X\left(k \frac{2\pi}{N}\right) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}k} \quad 0 \leq k \leq N - 1$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(k \frac{2\pi}{N}\right)e^{j\frac{2\pi}{N}n} \quad 0 \leq n \leq N - 1$$

(3.5)

Note: Both sequences, time and frequency, are discrete.

Both the sequences are periodic (see Problem 3.2.1). This indicates that the DFT is associated with periodic sequences in both the time and frequency domains. To find the DFT of a time sequence and the inverse, we use the following two MATLAB functions:

```
>> X = fft(x, N);
>> x = ifft(X, N);
```

$\{X\}$ is an $N$-point sequence (vector) and $\{x(n)\}$ is the input time sequence (vector).

The middle point $N/2$ of $\{X\}$ is known as the fold-over frequency. Plotting the magnitude of $\{X\}$ versus frequency ($k2\pi/N$), where $2\pi/N$ is the bin width, we find that $\pi$ is the middle of the frequency spectrum, and $\pi$ to $2\pi$ is the reflection from 0 to $\pi$. Similarly, we find that $\pi/T$ is the fold-over frequency when $T$ is positive different than 0. This indicates that to approximate the spectrum of a continuous signal, we must make the sampling time $T$ as small as possible.

Example 3.2.2

Find the DFT for the function $x(n) = 0.9^n u(n)$ for $N = 4$.

Solution: Since the sample time is unity, we use (3.5). Hence, we find

$$X\left[0 \left(\frac{2\pi}{4}\right)\right] = \sum_{n=0}^{4-1} 0.9^n e^{-j\frac{2\pi}{4}n} = 1 + 0.9 + 0.9^2 + 0.9^3 = 3.4390$$

$$= 0.1900 - 0.1710j$$

$$X\left[1 \left(\frac{2\pi}{4}\right)\right] = \sum_{n=0}^{4-1} 0.9^n e^{-j\frac{2\pi}{4}n} = 1 + 0.9e^{-j\frac{2\pi}{4}} + 0.9^2e^{-j\frac{2\pi}{4}} + 0.9^3e^{-j\frac{2\pi}{4}+j\frac{2\pi}{4}3}$$

$$= 0.1810 - 0.0000j$$

$$X\left[2 \left(\frac{2\pi}{4}\right)\right] = \sum_{n=0}^{4-1} 0.9^n e^{-j\frac{2\pi}{4}n} = 1 + 0.9e^{-j\frac{2\pi}{4}1} + 0.9^2e^{-j\frac{2\pi}{4}2} + 0.9^3e^{-j\frac{2\pi}{4}3+1}$$

$$= 0.1900 + 0.1710j$$
The amplitude spectrum is 3.4390, 0.2556, 0.1810, and 0.2556 for \(k = 0, 1, 2, \text{ and } 3\). The phase spectrum is 0.0000, –0.7328, 0, and 0.7328 for \(k = 0, 1, 2, \text{ and } 3\). The phase spectrum is given in radians. Since the frequency bin is \(\Omega_b = 2\pi/NT = 2\pi/4 \times 1 = \pi/2\), the amplitude and frequency spectrums are located at 0, \(\pi/2\), \(\pi\) (fold-over frequency), and \(3\pi/2\) radians per unit length.

### 3.2.3 Properties of DFT

To abbreviate the notation, we write

\[
X(k\Omega_b) = X(k); \quad x(nT) = x(n); \quad e^{-j2\pi/N} = W
\]

Using the above notation, the DFT properties are as follows:

<table>
<thead>
<tr>
<th>Property</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linearity</td>
<td>(af(n) + bh(n) \leftrightarrow aF(k) + bH(k))</td>
</tr>
<tr>
<td>Symmetry</td>
<td>((1/N)F(n) \leftrightarrow f(-k))</td>
</tr>
<tr>
<td>Time shifting</td>
<td>(f(n-i) \leftrightarrow F(k)e^{-j2\pi ki/N} = F(k)W^{ki})</td>
</tr>
<tr>
<td>Frequency shifting</td>
<td>(f(n)e^{j\omega n} \leftrightarrow F(k - i))</td>
</tr>
<tr>
<td>Time convolution</td>
<td>(y(n) \leftrightarrow f(n) \ast h(n) \leftrightarrow F(k)H(k))</td>
</tr>
<tr>
<td>Frequency convolution</td>
<td>(f(n)h(n) \leftrightarrow 1/N \sum_{k=0}^{N-1} F(k)H(n-x))</td>
</tr>
<tr>
<td>Parseval’s theorem</td>
<td>(\sum_{n=-N}^{N-1}</td>
</tr>
<tr>
<td>Time reversal</td>
<td>(f(-n) \leftrightarrow F(-k))</td>
</tr>
<tr>
<td>Delta function</td>
<td>(\delta(n) \leftrightarrow 1)</td>
</tr>
<tr>
<td>Central ordinate</td>
<td>(f(0) = \frac{1}{N} \sum_{k=0}^{N-1} F(k); \quad F(0) = \sum_{n=-N}^{N-1} f(n))</td>
</tr>
</tbody>
</table>

### Example 3.2.3

Find the DFT of the signal \(x(t) = \exp(-t)u(t)\).

**Solution:** Because there is no practical way to digitize the infinite signal, we select for this example the range \(0 \leq t \leq 4\) and two sampling times \(T = 1\) and \(T = 0.5\). The results from the DFT will be compared with the exact FT spectrum of the signal. The FT of the signal is

\[
X(\omega) = \int_{-\infty}^{\infty} e^{-t}u(t)e^{-j\omega t}dt = \int_{0}^{\infty} e^{-(1+j\omega)t}dt = \frac{1}{1+j\omega} = \frac{1}{(1+\omega^2)^{1/2}}e^{-j\tan^{-1}(\omega)} = A(\omega)e^{j\phi(\omega)}
\]
To find the DFT with $T = 1$ and compare the results with the exact FT, we use the following m-file example3_2_3 Book MATLAB program. At the Command window, having the Book MATLAB files in the MATLAB path (see Appendix 1), you write example3_2_3 and on return create Figure 3.3.

```
%example3_2_3
 t = 0:1:4;
 x = exp(-t); %x = vector with 5 elements;
 dftx1 = fft(x,32); %asks for 32 frequency bins, MATLAB
   %will pad x with 32-5 zeros;
 w = 0:2*pi/32:2*pi-(2*pi/32);
 FT1 = 1./sqrt(1+w.^2);
 subplot(2,1,1); %subplot(2,1,1) produces a picture with
   %two rows 1 column and the first row;
   stem(w,abs(dftx1),'k');
 hold on; %will produce a second plot on the same figure;
   stem(w,FT1,'k','filled'); %produces black ('k') lines
   %which are filled;
 xlabel('\omega'); ylabel('Magnitude of FT1 and DFT, T = 1');
 title('(a)'); axis([0 2*pi 0 2]); legend('DFT','FT');
 nt = 0:0.5:4;
 x1 = exp(-t);
 dftx2 = fft(x1,32);
```
\[ w_1 = 0:4\pi/32:4\pi-(4\pi/32); \]
\[ \text{FT2} = 1./\sqrt{1+w.^2}; \]
\[ \text{subplot}(2,1,2); \text{stem}(w1,\text{abs(dftx2)},'k'); \]
\[ \text{hold on}; \]
\[ \text{stem}(w1,\text{FT2},'k','filled'); \]
\[ \text{xlabel}('\omega'); \text{ylabel}('\text{Magnitude of DFT and FT, } T = 0.5'); \]
\[ \text{title}('(b)'); \text{axis}([0 4\pi 0 2]); \text{legend('DFT','FT')}; \]

**Note:**

1. From Figure 3.3a, we observe the following: (a) The amplitude of the FT of the continuous signal decreases for ever as it should. (b) The DFT of \( x(t) \) has folding frequency at \( \pi/1 \) (\( T = 1 \) in this case). (c) The approximation of the DFT to the exact DFT is up to fold-over frequency. (d) For the present assumed values, the two spectra are considerable different. (e) If the DFT command was \( \text{fft}(x,64) \), the lines would be twice as many but the accuracy would remain the same. If, however, we had padded with zeros the vector \( x \), the accuracy would stay the same. (f) The accuracy would have increased if we extended the time from \( 0 \leq t \leq 4 \) to \( 0 \leq t \leq 10 \), for example.

2. From Figure 3.3b, we observe that by decreasing the sampling time, in this case to 0.5, the accuracy increases. If, in addition, we had increased the sampling part of the time function, then the accuracy would have increased even further. We also observe that the time function has a spectrum with an infinite range. Therefore, it is practically impossible to produce an infinitely small sampling time, thus producing an extremely large number of vector samples. However, if the signal was band limited with a maximum frequency \( \omega_N \) (Nyquist frequency), then we would have been able to sample the signal at twice the Nyquist frequency and recapture the exact spectrum. Observe also that by reducing the sampling time by 1/2, the fold-over frequency doubles to \( 2\pi \).

3. Note also that we have plotted the values of the continuous spectrum, FT, at the corresponding frequencies of those found using the DFT. This gives us a better visual understanding of the accuracy of the sampled signal to the exact one.

### 3.3 THE \( z \)-TRANSFORM

The \( z \)-transform of a sequence is a transformation of a discrete-time sequence from the time domain to the frequency domain (\( z \)-domain). The \( z \)-transform is defined by

\[
X(z) = \mathcal{Z}\{x(n)\} = \sum_{n=-\infty}^{\infty} x(n)z^{-n}
\]  

(3.6)

For the inverse transform, we shall use Table 3.3 that is given below.
Example 3.3.1

Obtain the z-transform of the function \( x(n) = 0.8^n u(n) \).

**Solution:** Applying (3.6), we find

\[
X(z) = \sum_{n=0}^{\infty} 0.8^n z^{-n} = \sum_{n=0}^{\infty} (0.8 z^{-1})^n = \frac{1}{1 - 0.8 z^{-1}}
\]

For the last step, we used the geometric series property \((1 + x + x^2 + \cdots) = 1/(1-x)\) only if \( |x| < 1 \). This guarantees the convergence of the series in the z-domain. To obtain the region of convergence (ROC) in the z-domain, we must have \(|0.8 z^{-1}| = (0.8/|z|) < 1 \) or \(|z| > 0.8 \). Hence, the ROC for this problem is all the complex space outside the circle with radius 0.8. The ROC is important if we are to find the inverse transform using integration in the complex plane. If we set \( z = \exp(j \omega t) \) in (3.6), we obtain the DTFT for the time function \( x(n) \).

Example 3.3.2

Find the inverse z-transform of the function:

\[
F(z) = \frac{1}{(1 - 0.2 z^{-1})(1 + 0.2 z^{-1})} = \frac{z^2}{z^2 - 0.04}
\]

**Solution:** The equation can be written in the form:

\[
F(z) = \frac{z}{(z-0.2)(z+0.2)} = \frac{A}{z-0.2} + \frac{B}{z+0.2}
\]

Table 3.2 and 3.3 show the properties and the transform pairs of the z-transform.

**TABLE 3.2**

Properties of the z-Transform

<table>
<thead>
<tr>
<th>Property</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( \mathbb{Z}{ax(n) + by(n)} )</td>
<td>( aX(z) + bY(z) )</td>
</tr>
<tr>
<td>2. ( \mathbb{Z}{y(n-m)} )</td>
<td>( z^{-m}Y(z) + \sum_{i=0}^{m-1} y(-i)z^{-i} )</td>
</tr>
<tr>
<td>3. ( \mathbb{Z}{y(n+m)} )</td>
<td>( z^{m}Y(z) - \sum_{n=0}^{m-1} y(n)z^{-n-m} )</td>
</tr>
<tr>
<td>4. ( \mathbb{Z}{a^n y(n)} )</td>
<td>( Y(z/a) )</td>
</tr>
<tr>
<td>5. ( \mathbb{Z}{y(n)} )</td>
<td>( (z^n/z^n - 1)Y(z) ); ( y(0) ) is the first period of a periodic sequence ( y(n) = y(n+N) )</td>
</tr>
<tr>
<td>6. ( \mathbb{Z}{y(n) \ast h(n)} )</td>
<td>( Y(z)H(z) )</td>
</tr>
<tr>
<td>7. ( \mathbb{Z}{ny(n)} )</td>
<td>( -z(d/dz)Y(z) )</td>
</tr>
<tr>
<td>8. ( \mathbb{Z}{a^n y(n)} )</td>
<td>( [-(d/dz)]^n Y(z) )</td>
</tr>
<tr>
<td>9. ( \mathbb{Z}{y(n_0) } )</td>
<td>( z^{n_0}Y(z)</td>
</tr>
<tr>
<td>10. ( \lim_{z \to \infty} y(n) )</td>
<td>( \lim_{z \to \infty} (1-z^{-1})Y(z) )</td>
</tr>
<tr>
<td>Entry Number</td>
<td>$f(n)$, $f(nT)$ for $n \geq 0$</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>$\delta(n)$</td>
</tr>
<tr>
<td>2</td>
<td>$\delta(n-m)$</td>
</tr>
<tr>
<td>3</td>
<td>$1$</td>
</tr>
<tr>
<td>4</td>
<td>$n$</td>
</tr>
<tr>
<td>5</td>
<td>$n^2$</td>
</tr>
<tr>
<td>6</td>
<td>$n^3$</td>
</tr>
<tr>
<td>7</td>
<td>$a^n$</td>
</tr>
<tr>
<td>8</td>
<td>$na^n$</td>
</tr>
<tr>
<td>9</td>
<td>$n^2a^n$</td>
</tr>
<tr>
<td>10</td>
<td>$a^n/n!$</td>
</tr>
<tr>
<td>11</td>
<td>$(n+1)a^n/n!$</td>
</tr>
<tr>
<td>12</td>
<td>$[(n+1)(n+2)a^n]/2!$</td>
</tr>
<tr>
<td>13</td>
<td>$[(n+1)(n+2)\cdots(n+m)a^n]/m!$</td>
</tr>
<tr>
<td>14</td>
<td>$\sin(n\omega T)$</td>
</tr>
<tr>
<td>Line</td>
<td>Expression</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>15</td>
<td>( \cos(n\omega T) )</td>
</tr>
<tr>
<td>16</td>
<td>( a^n \sin(n\omega T) )</td>
</tr>
<tr>
<td>17</td>
<td>( a^n \cos(n\omega T) )</td>
</tr>
<tr>
<td>18</td>
<td>( e^{-anT} \sin(n\omega T) )</td>
</tr>
<tr>
<td>19</td>
<td>( e^{-anT} \cos(n\omega T) )</td>
</tr>
<tr>
<td>20</td>
<td>( n(n-1)/2! )</td>
</tr>
<tr>
<td>21</td>
<td>( n(n-1)(n-2)/3! )</td>
</tr>
<tr>
<td>22</td>
<td>( [n(n-1)(n-2)\cdots(n-m+1)/m!]a^{n-m} )</td>
</tr>
<tr>
<td>23</td>
<td>( e^{-anT} )</td>
</tr>
<tr>
<td>24</td>
<td>( ne^{-anT} )</td>
</tr>
</tbody>
</table>
The unknown constants are found as follows:

\[ A = \frac{z}{z + 0.2} \bigg|_{z\to0.2} = 0.5, \quad B = \frac{z}{z - 0.2} \bigg|_{z\to0.2} = 0.5 \Rightarrow F(z) = \frac{1}{2} \left[ \frac{z}{z + 0.2} + \frac{z}{z - 0.2} \right] \]

Using Table 3.3, we obtain

\[
\begin{align*}
  f(n) &= \begin{cases} 
    \frac{1}{2} \left[ 0.2^n + (-0.2)^n \right] & n \geq 0 \\
    0 & n < 0 
  \end{cases}
\end{align*}
\]

For a small number of values for the function \( f(n) \) in the above example, we can use the following Book MATLAB function example3_3_2.

```matlab
%example3_3_2
num = [1 0 0];
den = [1 0 -0.04];
f = dimpulse(num,den,5); %the number 5 requests 5 values
%of the time function f(n)
```

The 5 values that the above Book MATLAB function gives are \( f = [1 \ 0 \ 0.04 \ 0 \ 0.0016] \).

3.4 DISCRETE-TIME SYSTEMS

3.4.1 LINEARITY AND SHIFT INVARINATE

Two of the most important properties of the systems are linearity and time invariance or shift invariance (LTI). The system is linear if two inputs are applied to the system and its output is the sum of the outputs corresponding to the inputs. The system is time independent (or shift invariant) if the input is shifted in time and the output is shifted by the same amount.

3.4.2 CAUSALITY

A system is causal if its output at time \( t_0 \) is due only to the inputs for \( t \leq t_0 \). In other words, for a causal system, it is not possible for changes in the output to precede the excitations to the input.

3.4.3 STABILITY

If the output of a system is bounded for any bounded input is called a stable system. A system with such a property is said to be stable bounded-input–bounded-output (BIBO) sense.

A discrete time-invariant system is a physical device or an algorithm that transforms an input (or excitation) signal, \( \{x(n)\} \), into another is called the output signal, \( \{y(n)\} \).
Every system, including the discrete system, is defined by their output, known as the impulse response $h(n)$ whose $z$-transform $H(z)$ is known as the transfer function, when the input signal is the delta function. For the discrete system, the delta function is defined as follows:

$$
\delta(n) = \begin{cases} 
1 & n = 0 \\
0 & n \neq 0 
\end{cases}
$$

(3.7)

The following example presents the way we can transform a continuous system to a discrete and find its impulse response.

**Example 3.4.1**

Deduce an expression for the impulse response for the circuit shown in Figure 3.4a with the voltage source as input and the current as output. Use the $z$-transform technique.

**Solution:** The controlling equation is

$$
L \frac{di(t)}{dt} + Ri(t) = v(t)
$$

Figure 3.4b shows the approximation of a first-order derivative at point $A$. In the figure, $AD$ is the tangent at point $A$. The approximate tangent is $AB$. It is apparent that as $T$ approaches 0, the line $AB$ approaches $AD$ as it should be. Therefore, the above equation is written in discrete form as follows:

$$
L \frac{i(nT) - i(nT - T)}{T} + Ri(nT) = v(nT)
$$

or

$$
i(nT) - a_i i(nT - T) = b_v v(nT)
$$

$$
i(n) - a_i i(n - 1) = b_v v(n)
$$

Taking the $z$-transform (Tables 3.2 and 3.3) of the third equation, we obtain

$$
I(z) = b_v V(z) + a_i z^{-1} I(z) \quad \text{or} \quad I(z) = \frac{b_v}{1 - a_i z^{-1}} V(z) = H(z) V(z) \quad \text{or}
$$

$$
\text{Transfer function} = \frac{\text{Output}}{\text{Input}} = \frac{I(z)}{V(z)} = H(z) = \frac{b_v}{1 - a_i z^{-1}}
$$

From Table 3.3, we find that the inverse transform of the last equation, that is, the impulse response of the system, is

$$
h(n) = b_v (a_i)^n \quad n \geq 0
$$
Adaptive Filtering

\[ \nu(t) \]

\[ i(t) \]

(a)

(b)

\[ i(\nu T) \]

\[ i(\nu T - T) \]

\[ t \]

(c)

(d)

FIGURE 3.4
Let us assume the following values for the system components: \( L = 2 \), \( R = 1 \), and \( T = 1 \). Hence, \( a_1 = 2/3 \) and \( b_0 = 1/3 \). Therefore, the first 10 values of the impulse response are as follows:

\[
\begin{align*}
0.3333 & \quad 0.2222 & \quad 0.1481 & \quad 0.0988 & \quad 0.0658 & \quad 0.0439 & \quad 0.0293 & \quad 0.0195 & \quad 0.0130 & \quad 0.0087 \\
\end{align*}
\]

Next, we solve by iteration the difference equation \( i(n) = (2/3)i(n-1) + (1/3)v(n) \), with \( i(-1) = 0 \).

\[
i(0) = \frac{2}{3}i(-1) + \frac{1}{3} = \frac{1}{3} = 0.3333 \\
i(1) = \frac{2}{3}i(0) = \frac{2}{3} \frac{1}{3} = 0.2222 \quad \delta(\neq 0) = 0 \\
i(2) = \frac{2}{3}i(1) = \frac{2}{3} \frac{2}{3} \frac{1}{3} = 0.1481 \\
\vdots
\]

Let us try to find the current through and the voltage across the capacitor of an RC in a series circuit. Applying Kirchhoff’s law, we obtain

\[
Ri(t) + \frac{1}{C} \int_0^t i(t')dt' = v(t) \Rightarrow \frac{dq(t)}{dt} + \frac{1}{RC}q(t) = \frac{1}{R}v(t) \quad \left[ i(t) = \frac{dq(t)}{dt} \right]
\]

With \( R = 0.5 \), \( C = 1 \), and \( v(t) = \exp(-t) \), we obtain

\[
\frac{dq(t)}{dt} + 2q(t) = 2e^{-t}
\]

The homogeneous solution, \( v(t)=0 \), is \( q_h(t) = Ae^{-2t} \). The particular solution is assumed to be equal to \( q_p(t) = Be^{-t} \). Inserting this solution into a complete equation, we obtain \( B = 2 \). Hence, the total solution is \( q(t) = Ae^{-2t} + 2 \Rightarrow q(0) = 0 = A + 2 \Rightarrow A = 2 \).

Therefore, \( i(t) = \left[ dq(t)/dt \right] = 4e^{-2t} \). The discretized charge of the capacitor is \( q(nT) = 2 - 2e^{-2nT} \). Hence, the discrete form of the current is

\[
i(nT) = \frac{q(nT) - q(nT - T)}{T} = \frac{2 - 2e^{-2nT} - 2 + 2e^{-2(n-1)T}}{T} = \frac{2}{T} \left[ e^{-2(n-1)} - e^{-2nT} \right]
\]

Figure 3.4c shows the exact and discretized values of the current for two different \( T \)s.

We also observe that the voltage across the capacitor \( [q(t)/C] \) approaches the value of two volts as the time goes to infinity.

In Example 3.4.1, we found that the output in the \( z \)-domain is the product of the input and the transfer function. Based on Table 3.3, we observe that the inverse \( z \)-transform of a product of functions is equal to their convolution in the time domain. Hence, the inverse \( z \)-transform of the product \( Y(z) = H(z)X(z) \) is
\[ y(nT) = T \sum_{m=-\infty}^{\infty} x(mT)h(nT-mT) \quad n = 0, \pm 1, \pm 2, \ldots \quad m = 0, \pm 1, \pm 2, \ldots \]  
(3.8)

\[ y(n) \triangleq x(n) * h(n) = \sum_{m=-\infty}^{\infty} x(m)h(n-m) = \sum_{m=-\infty}^{\infty} x(n-m)h(m) \]

The above equation indicates what procedure must be executed to find \( y(n) \).

1. We select another domain, in this case \( m \)-domain, for both the signals \( x(n) \) and \( h(n) \).
2. For one of the signals, we substitute \( m \) in place of \( n \). This means that the signal appears identical to both domains.
3. We shift the other function by \( n \) and flip it (see the minus sign in front of \( m \)) in the \( m \)-domain.
4. Next, we multiply the two sequences term by term and then add the result.
5. The result is the output at time \( n \).
6. We repeat the same procedure for all \( n \)’s and thus obtain the output for \(-\infty \) to \( \infty \).

**Example 3.4.2**

Obtain the convolution of the two functions \( x(n) = u(n) \) and \( h(n) = 0.9^n u(n) \).

**Solution:** We first flip and shift the unit function. If the shift is done toward to the left, by setting \( n < 0 \), the two functions do not overlap and the product is 0 for all the negative values of \( n \). Therefore, we obtain the output if \( n \) is greater or equal to 0. Then, for this case (3.8) takes the form:

\[ y(n) = \sum_{m=0}^{\infty} u(n-m)0.9^m = \sum_{m=0}^{n} 0.9^m = 1 + 0.9 + 0.9^2 + \cdots + 0.9^n = \frac{1 - 0.9^{n+1}}{1 - 0.9} \]

where we used the formula for finite geometric series. The step function \( u(n-m) \) is 0 for \( m > n \) and it is equal to 1 for \( m < n \).

Table 3.4 gives the most important properties of the convolution operation of discrete signals.

---

**Table 3.4**

<table>
<thead>
<tr>
<th>Useful Convolution Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commutative</strong></td>
</tr>
<tr>
<td><strong>Associative</strong></td>
</tr>
<tr>
<td><strong>Distributive</strong></td>
</tr>
<tr>
<td><strong>Identity sequence</strong></td>
</tr>
<tr>
<td><strong>Shifted delta function</strong></td>
</tr>
</tbody>
</table>
3.4.3 Transform-Domain Representation

The output, without initial conditions, in the \( z \)-domain of a discrete system is equal to the product of its transfer function and its input. Hence,

\[
Y(z) = X(z)H(z) = \text{System function or transfer function} \quad (3.9)
\]

If the poles of the transfer function are inside the unit circle, the system is stable and \( H(e^{\omega j}) \) provides its frequency response.

A general realizable discrete LTI and causal system (the impulse response of the system is 0 for \( t < 0 \)) can be represented by the following difference equation:

\[
y(n) + \sum_{m=1}^{p} a_m y(n - m) = \sum_{m=0}^{q} b_m x(n - m) \quad (3.10)
\]

where:
- \( y(n) \) is the output
- \( x(n) \) is the input
- \( a_m \) and \( b_m \) are the constants specifying each particular system

To obtain the output of a discrete system using MATLAB, we write

\[
\texttt{y = \text{filter}(b,a,x);} \quad \% \text{of the b’s (see (3.10)); x = row input vector \{x(n)\};}
\]

\[
\texttt{y = \text{output vector \{y(n)\};}
\]

Taking the \( z \)-transform of (3.10), and remembering that the transform is with respect to the time \( n \) and referring to Table 3.2 for shifting functions, we obtain the transfer function for the general system:

\[
Y(z) + \sum_{m=1}^{p} a_m Y(z) z^{-m} = \sum_{m=0}^{q} b_m X(z) z^{-m}
\]

or

\[
Y(z) = \frac{\sum_{m=0}^{q} b_m z^{-m}}{1 + \sum_{m=1}^{p} a_m z^{-m}} = \frac{B(z)}{A(z)} \quad (3.11)
\]

The above system is known as the autoregressive moving average (ARMA) system. It is made up of two systems (filters): a finite impulse response (FIR) and an infinite response system (IIR). The example below presents a first-order ARMA system.
Example 3.4.3

Using (3.10), define a first-order ARMA system and plot its block-format diagram.

**Solution**: In Equation 3.11, we set \( m = 0, 1 \) and, thus, we find the equation:

\[
y(n) + a_1 y(n-1) = b_0 x(n) + b_1 x(n-1)
\]

Taking the \( z \)-transform of the above equation, we obtain the transfer function:

\[
H(z) \triangleq \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}
\]

Set \( z = \exp(-j\omega) \) in the above equation and use the following Book MATLAB program to obtain the frequency spectrum of the transfer function (Figure 3.5). Using \( a_1 = 2 \), \( b_0 = 3 \), and \( b_1 = 2 \), we obtain

```matlab
>> w = 0:0.1:2*pi;
>> H = (3+2*exp(j*w))./(1+2*exp(j*w));
>> subplot(2,1,1);
>> plot(w,abs(H),’k’); % abs() is a MATLAB function that gives
>> ylabel(‘abs(H)’);
>> subplot(2,1,2);
>> plot(w,angle(H),’k’); % angle() is a MATLAB function;
>> xlabelf(‘\omega’); ylabel(‘Angle in rad.’);
```

![Figure 3.5](image-url)
If we had continued the frequency beyond $2\pi$, we would reproduce the diagram every $2\pi$; in other words, the spectrum is periodic every $2\pi$. We also observe that the curve falls off from 0 to $\pi$. This indicates that the given system is a low-pass filter.

To be able to create block diagrams, we must use the fundamental building blocks that are shown in Figure 3.6. The block diagram representation is shown in Figure 3.7.

If in Equation 3.11 we set the $a$’s equal to zero, then the system in time and $z$-domain is given, respectively, by

$$y(n) = \sum_{m=0}^{q} b_m x(n-m)$$

$$H(z) = B(z)$$

(3.12)

The above system is known as the **FIR** system or **nonrecursive**.

If in the same equation we set $b$’s equal to 0 besides $b_0$, we obtain a system called the **infinite impulse response** (IIR) system or **recursive**. The system in time and $z$-domain is given by

\[ y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + a_1 y(n-1) \]

**FIGURE 3.6**

**FIGURE 3.7**
Figure 3.8 shows the above three systems in their block-format realization. In this chapter, we are mostly concerned with only FIR filters because they are inherently stable. Figure 3.8a shows a FIR system, Figure 3.8b shows an IIR system, and Figure 3.8c shows an ARMA filter.

**PROBLEMS**

3.1.1 Use MATLAB to plot $\exp(-0.1n)$, $\exp(-0.25n)$, and $f(n) = \exp(-0.1n) - \exp(-0.25n)$.

3.2.1 Show that sequences $\{x(n)\}$ and $\{X(k)\}$ of the DFT are periodic with a period $N$. 

$$y(n) + \sum_{m=1}^{p} a_m y(n-m) = b_0 x(n)$$

$$H(z) = \frac{b_0}{1 + \sum_{m=1}^{p} a_m z^{-m}} = \frac{b_0}{A(z)}$$

(3.13)
3.2.2 Use the Book MATLAB m-file, example3_2_3, and verify all the observations given in the Note section following Example 3.2.3. The reader should change the appropriate constants to verify the results.

3.2.3 Find the fold-over frequency for the function \( f(t) = \exp(-0.2t)u(t) \) for \( 0 \leq n \leq 255 \), \( T = 0.1 \), and \( T = 0.4 \).

3.3.1 Find the \( z \)-transform for the function \( f(t) = A\exp(-at)u(t) \) sampled every \( T \) seconds.

3.4.1 Find the convolution of the two discrete functions: \( x(n) = 0.9^n u(n) \) and the pulse having 7 elements \( h(n) = u(n) - u(n-6) \).

**HINTS–SUGGESTIONS–SOLUTIONS**

3.2.1 If we replace \( k \) in (3.5) by \( k + N \), we find that \( X(k) \) is periodic with period \( N \). Similarly, if we introduce \( n + N \) instead of \( n \) in the second equation, we also find that \( n \) is periodic with period \( N \).

3.3.1

\[
\{f(nT)\} = A, Ae^{-aT}, Ae^{-2aT}, \ldots; F(z) = A \left[ 1 + \frac{e^{-at}}{z} + \left( \frac{e^{-at}}{z} \right)^2 + \cdots \right]
\]

\[
= \frac{A}{1 - \left( e^{-aT}/z \right)} = \frac{Az}{z - e^{-aT}}
\]

The ROC is defined as

\[
\left| e^{-aT}/z \right| = e^{-aT} |z^{-1}| < 1 \quad \text{or} \quad |z| > e^{-aT}
\]

3.4.1 Use the following Book MATLAB program to produce Figure P3.4.1

---

**FIGURE P3.4.1**
% prob3_4_1
x = [1 1 1 1 1 1];
n = 0:20;
h = 0.8.^n;
y = conv([x zeros(1,15)],h); % conv(x,h) is a MATLAB function
% that gives the convolution between two vectors
% of equal number of elements, if they are not
% equal we pad the smaller with zeros;
m = 0:40;
stem(m,y,'k');xlabel('Time units');ylabel('Output y');

We can also use the following Book MATLAB program:

% prob3_4_1a
for n = 0:6
    for m = 0:n
        y(m+1) = 0.8^(n-m);
    end;
    o(n+1) = sum(y);
end;
for n = 8:20
    for m = 0:6
        y1(m+1) = 0.8^(n+m-6);
    end;
    o1(n) = sum(y1);
end;
o11 = o1(8:20);k = 0:19;stem(k,[o o11],'k');
xlabel('Time units');ylabel('Conv. x*h');
4 Discrete-Time Random Processes

4.1 DISCRETE RANDOM SIGNALS, PROBABILITY DISTRIBUTIONS, AND AVERAGES OF RANDOM VARIABLES

Most signals in practice are not deterministic but random. However, they can be described by precise mathematical analysis whose tools are contained in the theory of statistical analysis. In this chapter, we are dealing only with discrete random signals. This can always be accomplished by sampling the continuous signals at sampling rates at least twice their highest frequency, thus avoiding aliasing. Remember that the signals we receive are band limited because all signals must be detected by a physical (not ideal) transducer, such as a voltmeter or a receiver, which measures the electrocardiogram, the movement during an earthquake, and so on. All these physical transducers cannot respond to a delta function excitation, and hence, they pass only frequencies up to a specific value.

It is customary to use uppercase letters for random variables. However, in this chapter, we use for both cases, deterministic and random, lowercase letters, and their identity will be understood by the context. A discrete random signal \( \{ x(n) \} \) is a sequence of \textbf{indexed random variables} (rv’s) assuming the values:

\[
\{ x(0), x(1), x(2), \ldots \}
\] (4.1)

The random sequence with values \( \{ x(n) \} \) is discrete with respect to sampling index \( n \). In our case, we assume that the rv at any time \( n \) takes continuous values, and hence, it is a continuous rv at any time \( n \). What we really say is that we can associate at each \( n \) an infinite set of values (continuous) of the rv \( x(n) \). This type of sequence is also known as \textbf{time series}. In case we study a continuous random signal, we will assume that we have sampled it at high enough rate so that we construct a time series that is free of aliasing in the frequency domain.

A particular rv \( x(n) \) is characterized by its \textbf{probability density function} (pdf), \( f(x(n)) \), at the specific time \( n \) by

\[
f(x(n)) = \frac{\partial F(x(n))}{\partial x(n)}
\] (4.2)

and its \textbf{cumulative density function} (cdf), \( F(x(n)) \), is given by

\[
F(x(n)) = \text{pr}\{ x(n) \leq x(n) \} = \int_{-\infty}^{x(n)} f(y(n))dy(n)
\] (4.3)
In the above equation, the first $x(n)$ from left is an rv and the second is the value it takes. $f(x(n))$ is the pdf such as the normal and the exponential. $\Pr\{x(n) \leq x(n)\}$ specifies the probability of the rv $x(n)$ at time $n$ to be less than or equal to the value $x(n)$ at time $n$. As the value $x(n)$ at $n$ approaches infinity, the cdf approaches 1.

Similarly, for multivariate distributions, we have

$$F(x(n_1), \ldots, x(n_k)) = \Pr\{x(n_1) \leq x(n_1), \ldots, x(n_k) \leq x(n_k)\}$$

$$f(x(n_1), \ldots, x(n_k)) = \frac{\partial^k F(x(n_1), \ldots, x(n_k))}{\partial x(n_1) \cdots \partial x(n_k)}$$

(4.4)

If, for example, we want to check the accuracy of reading a dial by a person, we will have two readings: one due to the person and another due to the instruments. A simultaneous plot of these two readings, each one associated with a different orthogonal axis, will produce a scattering diagram but with a linear dependence. The closer the points fall on a straight line, the more reliable the person’s readings are. This example presents a case of a bivariate distribution.

To obtain a formal definition of a discrete-time stochastic process, we consider an experiment with a finite or infinite number of unpredictable outcomes from a sample space, $S(z_1, z_2, \ldots)$, each one occurring with a probability $\Pr\{z_i\}$. Next, by some rule we assign a deterministic sequence $\{x(n, z)\}$, $-\infty < n < \infty$, to each element $z_i$ of the sample space. The sample space, the probabilities of each outcome, and the sequences constitute a discrete-time stochastic process or a random sequence. From this definition, we obtain the following four interpretations:

- $x(n, z)$ is an rv if $n$ is fixed and $z$ is variable.
- $x(n, z)$ is a sample sequence called realization if $z$ is fixed and $n$ is variable.
- $x(n, z)$ is a number if both $n$ and $z$ are fixed.
- $x(n, z)$ is a stochastic process if both $n$ and $z$ are variable.

Each time we run an experiment under identical conditions, we create a sequence of rv’s $\{x(n)\}$, which is known as a realization and constitutes an event. A realization is one member of a set called the ensemble of all possible results from the repetition of an experiment. It is obvious that the more realizations we create, the better we can find some of the characteristics of the rv. The following Book MATLAB m-file produces 5 realizations shown in Figure 4.1.

Book m-File: ch4_realizations

```matlab
%Book MATLAB m-file: ch4_realizations
for n = 1:5
    x(n,:) = randn(1,50); %x = 5x50 matrix with each row having
    %zero mean normal (Gaussian) rv signals;
end;
m = 0:49;
for i = 1:5
```
4.1.1 Stationary and Ergodic Processes

It is seldom in practice that we will be able to create an ensemble of a random process with numerous realizations so that we can find some of its statistical characteristics, for example, mean value, variance, and so on. To find these statistical quantities, we need to know the pdf of the process, which, most of the times, is not possible to know. Therefore, we will restrict our studies to processes that are easy to study and handle mathematically.

The process that produces an ensemble of realizations and whose statistical characteristics do not change with time is called stationary. For example, the pdf of the rv’s $x(n)$ and $x(n+k)$ of the process $\{x(n)\}$ are the same independently of the values of $n$ and $k$.

Since we will be unable to produce ensemble averages in practice, we are left with only one realization of the stochastic process. To overcome this difficulty, we assume that the process is ergodic. This characterization permits us to find the

```matlab
subplot(5,1,i);stem(m,x(i,:),'k'); % plots five rows of matrix x;
xlabel('n');ylabel('x(n)');
```

**FIGURE 4.1**
desired statistical characteristics of the process from only one realization at hand. We refer to those statistical values as **sample mean**, **sample variance**, and so on. We must have in mind that these values are the approximate to the ensemble values. One of the main objectives of the statisticians is to find the ways and give to these sample values some confidence limits around the sample values within which, with a high degree of probability, the ensemble value exists.

### 4.1.2 Averages of RV

#### 4.1.2.1 Mean Value

The ensemble **mean value** or **expectation value** \( m(n) \) at time \( n \) of a \( \text{rv} \ \text{x}(n) \) having a pdf \( f(x(n)) \) is given by

\[
m(n) = E\{x(n)\} = \int_{-\infty}^{\infty} x(n)f(x(n))dx(n)
\]  

(4.5)

where:

\( E[.] \) stands for expectation operator that is defined by the integration above

We can use the frequency formula to approximate the ensemble average by using the realizations. It is obvious that the larger the number of the realizations we use, the better approximation of the mean value we obtain to the ensemble average. We can find the mean value using the approximation formula:

\[
m(n) = \lim_{N \to \infty} \left\{ \frac{1}{N} \sum_{i=1}^{N} x_i(n) \right\}
\]  

(4.6)

where:

\( N \) is the number of realizations

\( x_i(n) \) is the outcome at time \( n \) and of the \( i \)th realization

The above equation indicates that at a particular time \( n \), we add all the values \( x(n) \) belonging to different realizations and divide by the number of the realizations used (in this case, \( N \) realizations).

For an ergotic process, the sample mean approximation (estimator) is found using the formula:

\[
\hat{m} = \frac{1}{N} \sum_{n=0}^{N-1} x(n)
\]  

(4.7)

It turns out that (see Problem 4.1.1) the sample average (mean) that is found using (4.7) is equal to the ensemble average of the mean. We call the estimate an **unbiased** estimator.
Example 4.1.1

Using the results of the Book m-file: ch4_realizations, find the mean value using the frequency formula and for \( n = 12, n = 17, \) and \( n = 21. \)

Solution: Running the program, we obtain \( x(12) = \left\{ -1.3335 \ 1.8482 \ -0.6885 \right\}, \ x(17) = \left\{ -0.2494 \ -1.6805 \ 1.6197 \ -1.9171 \ -0.1435 \right\}, \ x(21)= \left\{ -1.0290 \ 0.8369 \ 1.7357 \ 1.3808 \ -0.6614 \right\}. \) The corresponding means are \( m(12) = 0.0468, \ m(17) = -0.4742, \) and \( m(21) = 0.4526. \) We must note that if we rerun the program, we will find different vectors for the same times. Since the program gives a matrix \( x \) with dimensions \( 5 \times 50 \) (5 rows by 50 columns), we use the MATLAB expression \( x(:,n) \) to get the vectors. This expression says: Produce the vector that is its \( n \) column. We can also think of all the rows of the column \( n. \)

\[ r_{xy}(m,n) = E\{x(m),y(n)\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x(m)y(n)f(x(m),y(n))dx(m)dy(n) \] (4.8)

The limits of the integrals are from \(-\infty\) to \(\infty.\) If \( x(n) = y(n), \) the correlation is called autocorrelation.

4.1.2.2 Frequency Interpretation of the Correlation

Having an ensemble of realizations, the frequency interpretation of the autocorrelation function is found using the formula:

\[ r_{xx}(m,n) = \lim_{N \to \infty} \left\{ \frac{1}{N} \sum_{i=1}^{N} x_i(m)x_i(n) \right\} \] (4.9)

The Book m-function that produces the mean and autocorrelation using the frequency interpretation formulas is as follows:

**Book m-Function**

```matlab
function [mx,rx] = fr_mean_autocor(M,N)
    % to store the function we write only the name: fr_mean_autocor;
    % at the MATLAB command window we write: [a,b] = fr_mean_autocor;
    % other pdf's; M is the rows (realizations) of the matrix;
    x = randn(M,N);  \% randn = MATLAB function producing zero mean
    % Gaussian distributed white noise; x = MxN matrix;
    % sum(x,1) = MATLAB function that sums all the rows;
    % sum(x,2) = MATLAB function that sums all the columns;
    mx = sum(x,1)/M;
    for i = 1:N
        rx(i) = sum(x(:,1).*x(:,i))/M;
    end;
```
4.1.2.2 Sample Autocorrelation Function

The sample autocorrelation function is found from only one realization at hand. The formula is

\[ \hat{r}_{xx}(m) = \frac{1}{N-|m|} \sum_{n=0}^{N-|m|-1} x(n)x\left(n + |m|\right) \quad m = 0, 1, \ldots, N - 1 \]  

(4.10)

The absolute value of \( m \) ensures the symmetry of the sample autocorrelation sequence at \( n = 0 \). Although the formula gives an unbiased autocorrelation sequence, it sometimes produces autocorrelation matrices (discussed below) that do not have inverses. Therefore, it is customary in practice to use anyone of the biased formulas:

\[ \hat{r}_{xx}(m) = \frac{1}{N} \sum_{n=0}^{N-1} x(n)x(n-m) \quad 0 \leq m \leq N - 1 \]

or

(4.11)

Book m-Function for Biased Autocorrelation Function

```matlab
function [r] = lms_sample_biased_autoc(x, lg)
% this function finds the biased autocorrelation function
% with lag from 0 to lg; it is recommended that lg is 20-30% of
% N; N is the total number of elements of the observed vector;
% x = data; lg stands for lag number;
N = length(x);%x = data;lg stands for lag number;
for m = 1:lg
    for n = 1:N+1-m
        x1(n) = x(n-1+m)*x(n);
    end;
    r(m) = sum(x1)/N;
end;
```

Book m-Function for Unbiased Autocorrelation Function

```matlab
function [r] = lms_sample_unbiased_autoc(x, lg)
% this function finds the biased autocorrelation function
% with lag from 0 to lg; it is recommended that lg is 20-30% of
% N; N is the total number of elements of the observed vector;
% x = data; lg stands for lag number;
for m = 1:lg
    for n = 1:N+1-m
        x1(n) = x(n-1+m)*x(n);
    end;
    r(m) = sum(x1)/N;
end;
```
for n = 1:N+1-m
    x1(n) = x(n-1+m)*x(n);
end;
    r(m) = sum(x1)/(N-m);
end;

Figure 4.2 shows the plots of an unbiased autocorrelation function (shown by circles) and a biased autocorrelation function (shown by dots).

We can also use MATLAB functions to find the biased and unbiased autocorrelation of a signal. The function is

```
>> r = xcorr(x,y,'biased'); % for biased case; xcorr() is a MATLAB function;
>> r = xcorr(x,y,'unbiased'); % for unbiased case;
>> % x, y are N length vectors; r is a 2N-1 symmetric cross-correlation vector; for unequal vectors we pad
>> % with zeros the shorter;
```

If we do not use any of the words, biased or unbiased, the correlation will be unbiased and the results are not divided by $N$. The reader will learn more about the xcorr function by entering in the MATLAB Command window help xcorr.

4.1.2.3 Covariance

A **covariance** (autocovariance) of a random sequence is defined by

![Figure 4.2](image-url)
\[ c_{xx}(m,n) = E \{ [x(m) - m_m] \{x(n) - m_n \} \} \]
\[ = E \{ x(m)x(n) \} - m_m E \{ x(n) \} - m_n E \{ x(m) \} + m_m m_n \]  
\[ = r_{xx}(m,n) - m_m m_n \]  
(4.12)

If we set the second \( x \) with \( y \), we produce the cross-covariance of the two different random sequences. If \( c_{xx}(m,n) = 0 \) for all \( m \)'s and \( n \)'s, then
\[ r_{xy}(m,n) = m_m m_n \]  
(4.13)

Two random sequences are orthogonal if their cross-correlation function is 0. Hence,
\[ r_{xy}(m,n) = 0 \]

**Example 4.1.2**

The output \( y(n) \) of the linear time-invariant (LTI) system is given by the convolution of the input \( x(n) \) and its impulse response \( h(n) \) (deterministic function), or
\[ y(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m) \]

The cross-correlation between the input and the output is given by
\[ r_{xy}(k,l) = E \{ x(k)y(l) \} = E \left\{ x(k) \sum_{m=-\infty}^{\infty} h(m)x(l-m) \right\} \]
\[ = \sum_{m=-\infty}^{\infty} h(m)E\{x(k)x(l-m)\} \]
\[ = \sum_{m=-\infty}^{\infty} h(m)r_{xx}(k,l-m) \]

The variance is found by setting \( m = n \) in (4.12). Therefore, we obtain
\[ c_{xx}(n,n) \triangleq \sigma_n^2 = E \left\{ [x(n) - \bar{m}_n]^2 \right\} = E \left\{ x^2(n) \right\} - \bar{m}_n^2 = r_{xx}(n,n) - \bar{m}_n^2 \]  
(4.14)

4.1.2.3.1 Sample Variance

The estimator of the variance, when we use one sample from the ensemble, is given by
\[ \hat{\sigma}_n^2 = \frac{1}{N} \sum_{n=1}^{N} [x(n) - \hat{m}]^2; \quad \hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^{N} x(n)^2 \quad \hat{m} = 0 \]  
(4.15)
To obtain the variance, we can also use the MATLAB function `var(data_vector)`. The standard deviation that is equal to the square root of the variance can be found using the MATLAB function `std(data_vector)`.

### 4.1.2.3.2 Independent and Uncorrelated rv’s

If the joint pdf of two rv’s can be separated into two pdf’s, $f_{xy}(m,n) = f_x(m)f_y(n)$, then the rv’s are statistically independent. Therefore,

$$E\{x(m)x(n)\} = E\{x(m)\}E\{x(n)\} = m_mm_n$$  \hspace{1cm} (4.16)

The above equation is necessary and sufficient condition for the two rv’s $x(m)$, $x(n)$ to be uncorrelated. Note that independent rv’s are always uncorrelated. However, the converse is not necessarily true. If the mean value of any two uncorrelated rv’s is 0, then the rv’s are called orthogonal. In general, two rv’s are called orthogonal if their correlation is 0.

### 4.2 Stationary Processes

For a wide-sense (or weakly) stationary (WSS) process, the joint pdf satisfies the relation

$$f(x(n),x(m)) = f(x(n+k)x(m+k))$$  \hspace{1cm} (4.17)

for any $k$, $m$, and $n$ (Table 4.1).

#### 4.2.1 Autocorrelation Matrix

In this case, we use a $3 \times 3$ autocorrelation matrix for elucidating the procedure. This is easily extended to any matrix dimensions. Let $x = [x(0) \ x(1) \ x(2)]^T$ is a vector representing a finite random sequence (in this case three elements); then the autocorrelation matrix is given by

---

**Table 4.1**

Properties of WSS Processes

1. $m_m = m_{n+k} = m =$ constant.
2. The autocorrection $r_x(m,n)$ depends on the difference $m-n$.
3. The variance is less than infinity.
4. The autocorrelation is symmetric: $r_x(n) = r_x(-n)$.
5. For lag $l = 0$, $r_x(0) = E\{|x(n)|^2\} \geq 0$.
6. $r_x(0) \geq |r_x(l)|$.

**Note:** See Problem 4.2.1.
The covariance matrix is given by (see Problem 4.2.4)

\[ C = E \left\{ (x - \mathbf{m}_x)(x - \mathbf{m}_x)^T \right\} = \mathbf{R}_x - \mathbf{m}_x \mathbf{m}_x^T \]  \hspace{1cm} (4.19)

**Example 4.2.1**

Find (1) the unbiased autocorrelation with lag 15 of a 50-term sequence, (2) the biased autocorrelation with the same settings, and (3) a $3 \times 3$ autocorrelation matrix. Use a sequence of independent rv’s having Gaussian (normal) distribution and zero mean value.

Solution: The sequence $\{x(n)\}$ of 50 terms, a vector, is obtained by using the MATLAB function `randn(1,50)`. To find the autocorrelation, we use either the Book m-files or the MATLAB function `xcorr()`. Figure 4.3 shows one realization of an unbiased autocorrelation function. To produce the $3 \times 3$ matrix, we first created the vector $\hat{x} = [0.7444 \ -0.0745 \ -0.1010]$ from the first three elements of the correlation function and then we used the MATLAB function `toeplitz(x',x)` to produce the matrix:

\[
\hat{\mathbf{R}}_x = \begin{bmatrix}
0.7444 & -0.0745 & -0.1010 \\
-0.0745 & 0.7444 & -0.0745 \\
-0.1010 & -0.0745 & 0.7444
\end{bmatrix}
\]

Note the symmetry of the matrix along the diagonals from left to right. This type of a matrix is called **Toeplitz**.
Note: If we have a row vector \( x \) and we need to create another row vector \( y \) with the elements of \( x \) from \( m \) to \( n \) only, we write

\[ y = x(1:m:n); \]

Note: If we have a column vector \( x \) and we need to create another column vector \( y \) with the elements of \( x \) from \( m \) to \( n \) only, we write

\[ y = x(m:n,1); \]

Note: If we have produced the autocorrelation vector \( r \) and we want to create the autocorrelation matrix \( R \) with the first 5 elements of \( r \), at the Command window we write:

\[ R = \text{Toeplitz}(r(1,1:5)); \]

Note: To create a subscript to xlabel() and ylabel() MATLAB functions, we write \( \text{xlabel('r \_ \{xx\}') \} \), which will produce the following expression on the x-axis of the figure: \( x_{xx} \). For a superscript, we write \( \text{xlabel('r^{12}') \} \), : \( x^{12} \).

Example 4.2.2

Let \( \{v(n)\} \) be a zero-mean, uncorrelated Gaussian random sequence with variance \( \sigma_v^2(n) = \sigma^2 = \text{constant} \). (1) Characterize the random sequence \( \{v(n)\} \) and (2) determine the mean and the autocorrelation of the sequence \( \{x(n)\} \) if \( x(n) = v(n) + av(n-1) \), in the range \( -\infty < n < \infty \); \( a \) is a constant.
Solution: (1) The variance of \( \{v(n)\} \) is constant and, hence, is independent of the time, \( n \). Since \( \{v(n)\} \) is an uncorrelated sequence, it is also independent due to the fact that it is Gaussian sequence. Therefore, we obtain
\[
c_r(l,n) = r_c(l,n) - m_x m_x = r_c(l,n)
\]
or \( \sigma^2 = r_c(n,n) = \text{constant} \). Hence, \( r_c(l,n) = \sigma^2 \delta(l-n) \), which implies that \( \{v(n)\} \) is a WSS process. (2) \( E\{x(n)\} = 0 \) since \( E\{v(n)\} = E\{v(n-1)\} = 0 \). Hence,
\[
r_c(l,n) = E\{[v(l) + av(l-1)]\{v(n) + av(n-1)\}\}
\]
\[
= E\{v(l)v(n)\} + aE\{v(l-1)v(n)\} + aE\{v(l)v(n-1)\} + a^2E\{v(l-1)v(n-1)\}
\]
\[
= r_c(l,n) + ar_c(l-1,n) + ar_c(l,n-1) + a^2r_c(l-1,n-1)
\]
\[
= \sigma^2 \delta(l-n) + a\sigma^2 \delta(l-n+1) + a^2 \sigma^2 \delta(l-n)
\]
\[
= (1 + a^2) \sigma^2 \delta(r) + a \sigma^2 \delta(r+1) + a^2 \sigma^2 \delta(r-1), \ l-n = r
\]
Since the mean of \( \{x(n)\} \) is 0, a constant, and its autocorrelation is a function of the lag factor \( r = l - n \), it is a WSS process.

\[
4.2.2 \quad \text{Purely Random Process (White Noise)}
\]
A discrete process is a purely random process if the rv’s \( \{x(n)\} \) of a sequence are mutually independent and identically distributed (iid) variables. Since the mean and \( \text{cov}\{x(m),x(m+k)\} \) do not depend on time, the process is said to be WSS. This process is also known as \text{white noise} (WN) and is given by
\[
c_{xx}(k) = \begin{cases} E\{[x(m)-m_x][x(m+k)-m_x]\} & k = 0 \\ 0 & k = \pm 1, \pm 2, \pm 3, \ldots \end{cases}
\]

\[
4.2.3 \quad \text{Random Walk}
\]
Let \( \{x(n)\} \) be a purely random process with mean \( m_x \) and variance \( \sigma_x^2 \). A process \( \{y(n)\} \) is a random walk if
\[
y(n) = y(n-1) + x(n) \quad y(0) = 0
\]
For \( n = 1 \), \( y(1) = x(1) \); for \( n = 2 \), \( y(2) = x(2) + x(1) \), and therefore, the solution is
\[
y(n) = \sum_{i=1}^{n} x(i)
\]
The mean is easily found to be \( E\{y(n)\} = nm_x \), and the covariance is
\[
\text{cov}\{y(n)y(n)\} = E\{[y(n)-m_y][y(n)-m_y]\} = n\left[E\{x^2\} - m_x^2\right] = n\sigma_x^2
\]
It is important to note that the difference \( x(n) = y(n) - y(n-1) \) is purely random and hence stationary.
4.3 SPECIAL RANDOM SIGNALS AND PDF’S

In this section, we present the most important types of r.v. distributions which, it is recommended, are used as additive noise in later chapters dealing with the adaptive filtering.

4.3.1 WHITE NOISE

If the pdf of a WSS discrete random sequence satisfies the relation

\[ f(x(0), x(1), \cdots) = f(x(0)) f(x(1)) f(x(2)) \cdots \]  

(4.24)

is a purely random sequence whose elements \( x(n) \) are **statistically independent and identically distributed**. Therefore, the zero-mean iid sequence has the following correlation function (see Problem 4.3.1):

\[ r_{xx}(m-n) = E \{ x(m) x(n) \} = \sigma_x^2 \delta(m-n) \]  

(4.25)

where:

- \( \sigma_x^2 \) is the variance of the r.v.’s at each \( n \)
- \( \delta(m-n) \) is the discrete-delta function

For \( m \neq n \), \( \delta(m-n) = 0 \), and hence, (4.25) becomes

\[ r_{xx}(k) = \begin{cases} \sigma_x^2 \delta(k) & k = 0 \\ 0 & k \neq 0 \end{cases} \]  

(4.26)

For example, a random process consisting of a sequence of uncorrelated Gaussian r.v’s is a WN process referred to as **white Gaussian noise** (WGN). The MATLAB has a special function that will produce WGN. For example, if we write \( x = \text{randn}(1,500) \) will produce a sequence (vector \( x \)) that its elements are normally distributed and variance 1. We can then use the MATLAB function \( \text{hist}(x,20) \) to produce the pdf of the time series \( \{ x(n) \} \). The function divides the range between the maximum and minimum values of the sequence in 20 sections. Then plot the number of \( x \)'s, whose values fall within each section, versus the range of values of \( x \)'s.

It is instructive to be able to create different types of r.v. distributions to be able to check their influence on filtering.

4.3.2 GAUSSIAN DISTRIBUTION (NORMAL DISTRIBUTION)

The pdf of a Gaussian r.v \( x(n) \) at time \( n \) is given by

\[ f(x(n)) \triangleq \mathcal{N}(m_{xn}, \sigma_{xn}^2) = \frac{1}{\sqrt{2\pi \sigma_{xn}^2}} e^{-\frac{(x(n) - m_{xn})^2}{2\sigma_{xn}^2}} \]  

(4.27)
The algorithm to produce normalized Gaussian distribution is as follows:

1. Generate two rv’s $u_1$ and $u_2$ from uniform distribution $(0,1)$.
2. $x_1 = [-2\ln(u_1)]^{1/2} \cos(2\pi u_2)$ [or $x_2 = [-2\ln(u_1)]^{1/2} \sin(2\pi u_2)$].
3. Keep $x_1$ or $x_2$.

**Book m-Function for Producing a Normal Distribution**

```matlab
function [x] = lmsnormalpdf(m,s,N)
%in the command window we write: [x] = lmsnormalpdf(m,s,N);
% m = mean value; s = standard deviation; N = number of % realizations;
for i = 1:N
    u1 = rand;%uniform distributed rv 0<u1<1;
    u2 = rand;
    z(i) = sqrt(-2*log(u1))*cos(2*pi*u2);%MATLAB function log()
    %gives the natural logarithm; log10()
    %gives the logarithm with base 10;
end;
x = s*z+m;
```

A discrete-time random process $\{x(n)\}$ is said to be Gaussian if every finite collection of samples of $x(n)$ are jointly Gaussian. A Gaussian random process has the following properties: (1) It is completely defined by its mean vector and covariance matrix, (2) any linear operation on the time variables produces another Gaussian random process, (3) all higher moments can be expressed by the first and second moments of the distribution (mean, covariance), and (4) WN is necessarily generated by iid samples (independence implies uncorrelated rv’s and vice versa).

For example, the distribution of the rv’s $x = \text{randn}(1,2000) + 2 * \text{randn}(1,2000) - 0.5 * \text{randn}(1,2000) + 0.8 * \text{randn}(1,2000) + \text{randn}(1,2000)$ is shown in Figure 4.4 using the Book m-file MATLAB program:

**Book m-File: normal_hist**

```matlab
%Book m-file: normal_hist;
x = randn(1,2000)+2*randn(1,2000)-0.5*randn(1,2000)...
+0.8*randn(1,2000)+randn(1,2000);
[m,z] = hist(x,20);%calculates counts in bins and %bin coordinates for 20 bins;
w = max(x)/length(z);%calculates bin length;
pb = m/(2000*w);%probability in each bin;
colormap([1 1 1]);%creates white bars, for other %colors see help colormap;
bar(z,pb);%plots the histogram;
```

Observe that the summation of normal rv’s, linear operation, is also normally distributed.
Example 4.3.1

If the pdf is $N(2.5,4)$ (mean 2.4 and variance 4), find $\text{pr}\{x \leq 6.5\}$.

Solution:

$$\text{pr}\{x \leq 6.5\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{6.5} e^{-\frac{(x-2.5)^2}{8}} \, dx \quad (1)$$

Set $y = (x-2.5)/2$ in (1) to obtain: for $x = -\infty \Rightarrow y = -\infty$, for $x = 6.5 \Rightarrow y = 2$ and $dx = 2dy$. Hence, (1) becomes

$$\text{pr}\{x \leq 6.5\} = \text{pr}\{y \leq 2\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{y^2}{2}} \, dy + \frac{1}{\sqrt{2\pi}} \int_{0}^{2} e^{-\frac{y^2}{2}} \, dy = 0.5 + \text{erf}(2)$$

The error function is tabulated, but MATLAB can give very accurate values by introducing sampling values of <0.0001. For this sampling value, MATLAB gives the result: $\text{erf}(2) = (1/\sqrt{2\pi})*0.0001*\text{sum}(-y.^2/2)=0.47727$, where the vector $y$ is given by $y = 0:0.0001:2$. Tables give $\text{erf}(2) = 0.47724$. Therefore, $\text{pr}\{x \leq 6.5\} = 0.97727$.

\[\square\]

**Book m-Function for Normalized Normal pdf**

```matlab
function [x] = lms_normalized_normal_pdf(m,s,N)
    z = sqrt(-2*log(rand(1,N))).*cos(2*pi*rand(1,N));
    x = s*z+m;
```

![FIGURE 4.4](image-url)
We can also use Monte Carlo approach to obtain the normal distribution of rv’s. The following Book m-function produces the desired results.

**Book m-Function for Monte Carlo Normal pdf**

```matlab
function [x,y] = lms_monte_carlo_normal_pdf(m,s,N,M)
    %m = mean value; s = standard deviations;
    %N = number of variables; M = number of summable
    %normal variables;
    for n = 1:N
        x(n) = sum(randn(1,M))/sqrt(M);
    end;
    y = m+s*x;
end;
```

The above Monte Carlo processing was based on the **central limit theorem**, which states as follows:

For $N$ independent rv’s $X_1$, $X_2$, ..., $X_N$ with mean $m_i$ and variance $\sigma_i^2$, respectively,

$$y_N = \frac{\sum_{i=1}^{N} (x_i - m_i)}{\sqrt{\sum_{i=1}^{N} \sigma_i^2}}, \quad \lim_{N \to \infty} N \rightarrow \infty \Rightarrow y \equiv N(0,1) \quad (4.28)$$

The central limit theorem has the following interpretation:

The properly normalized sum of many uniformly small and negligibly independent rv’s tends to be a standard normal (Gaussian) rv. If a random phenomenon is the cumulative effect of many uniformly small sources of uncertainty, it can be reasonably modeled as normal rv.

### 4.3.3 Exponential Distribution

The pdf of the exponential distribution is given by

$$f(x(n)) = \begin{cases} 
\frac{1}{a} e^{-x(n)/a} & 0 \leq x < \infty, \ a > 0 \\
0 & \text{otherwise} 
\end{cases} \quad (4.29)$$

The exponential distribution is generally denoted by exp($a$) and its standard form by exp(1). The mean value of the distribution is $a$ and its variance is $a^2$.

The algorithm to produce exponential distribution is as follows:

1. Generate $u$ from the uniform distribution (0,1).
2. $x = -\ln(u)$.
3. Keep $x$. 
Book m-Function for Producing an Exponential Distribution

function [x,m,sd] = lms_exp_pdf(a,N)
    %N = number of rv's needed to produce
    %the pdf;a = mean value of x;
    %variance of x is a^2;mean of x = a;
    %standard deviation = a;
    for i = 1:N
        u = rand;
        x(i) = -a*log(u);
    end;
    m = mean(x);sd = std(x);

4.3.4 Lognormal Distribution

Let \( X(n) \) be a normal distributed rv with mean \( m \) and variance \( \sigma^2 \), \( N(m,\sigma^2) \). Then, \( Y(n) = e^{X(n)} \) has the lognormal distribution with pdf:

\[
f(y(n)) = \begin{cases} \frac{1}{\sqrt{2\pi \sigma y(n)}} \exp \left\{ -\frac{[\ln y(n) - m]^2}{2\sigma^2} \right\} & 0 \leq y(n) < \infty \\ 0 & \text{otherwise} \end{cases}
\]

The algorithm to produce lognormal pdf is as follows:

1. Generate \( Z \) from a normal distribution with zero mean and unit variance, \( N(0,1) \).
2. \( X = m + \sigma Z \).
3. \( Y = e^X \).
4. Keep \( Y \).

Book m-Function to Produce Lognormal PDF

function [y,x] = lms_lognormal_pdf(m,s,N)
    %[y] = lognormal pdf; m = mean;s = standard deviation;
    %N = number of samples;
    for i = 1:N
        r1 = rand;r2 = rand;
        z(i) = sqrt(-2*log(r1))*cos(2*pi*r2);
    end;
    x = m+s*z;%x normaly distributed;
    y = exp(x);

Other statistical characteristics of the lognormal distribution are as follows:

Range: \( 0 \leq x < \infty \)
Mean: \( \exp(m)\exp(s^2/2) \)
Variance: \( \exp(2m)\exp(s^2)[\exp(s^2) - 1] \)
4.3.5 CHI-SQUARE DISTRIBUTION

If \( z_1, z_2, z_3, \ldots, z_n \) are \( N(0,1) \), then

\[
y = \sum_{i=1}^{n} z_i^2
\]

(4.31)

has chi-square distribution with \( n - 1 \) degrees of freedom, and it is denoted by \( \chi^2(n) \).

If \( z_1, z_2, z_3, \ldots, z_n \) are normally distributed with \( N(m,\sigma) \), then

\[
y = \frac{1}{\sigma^2} \sum_{i=1}^{n} (z_i - m)^2
\]

(4.32)

is chi-square distributed with \( n - 1 \) degrees of freedom and it is denoted by \( \chi^2(n) \).

**Book m-Function for Chi-Squared Distributed rv**

```matlab
function [y] = lms_monte_carlo_chi_squared_pdf(r,N)
    %y is chi-squared distributed rv; r = degrees
    %of freedom, MUST BE EVEN NUMBER; number
    %of chi squared rvs y;
    for n = 1:N
        y(n) = sum(randn(1,r).^2);
    end;
```

Other statistical characteristics of the chi-squared distribution are as follows:

- **Range**: \( 0 \leq y < \infty \)
- **Mean**: \( r \)
- **Variance**: \( 2r \)

4.4 WIENER–KHINCHIN RELATIONS

For a WSS process \( \{x(n)\} \), the correlation function asymptotically goes to 0, and therefore, we can find its spectrum using the discrete-time Fourier transform (DTFT). Hence, the power spectral density (PSD) is given by

\[
S_x(e^{j\omega}) = \sum_{k=-\infty}^{\infty} r_{xx}(k)e^{-j\omega k}
\]

(4.33)

This function is periodic with period \( 2\pi \) since \( \exp(j\omega) = \exp[j(\omega + 2\pi)] \). Therefore, the autocorrelation sequence is given by (see Problem 4.4.1)

\[
r_{xx}(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_x(e^{j\omega})e^{j\omega k} d\omega
\]

(4.34)
For a real symmetric correlation function, the power spectrum is even function. In addition, the power spectrum of a WSS process is also nonnegative. The mathematical expressions of the above assertions are as follows:

\[ S_x(e^{j\omega}) = S_x(e^{-j\omega}) = S_x^*(e^{j\omega}) \]

\[ S_x(e^{j\omega}) \geq 0 \]  

(4.35)

**Example 4.4.1**

Find the PSD of the sequence \( x(n) = \cos(0.2 \times 2\pi n) + \text{randn}(1,64) \) for \( n = 0, 1, 2, \ldots, 63 \).

**Solution:** The following Book m-file produces Figure 4.5.

**Book m-File: ex4_4_1**

```matlab
%Book m-file: ex4_4_1;
n = 0:63; s = cos(0.1*2*pi*n); v = randn(1,64);
x = cos(0.1*2*pi*n)+rand(1,64);
r = xcorr(x,'biased');% the biased autocorrelation 
% is divided by N, in this case by 64;
fs = fft(s); fr = fft(r,64);
w = 0:2*pi/64:2*pi-(2*pi/64);
subplot(3,2,1); stem(n,s,'k'); xlabel('n'); ylabel('s(n)');
subplot(3,2,2); stem(n,v,'k'); xlabel('n'); ylabel('v(n)');
subplot(3,2,3); stem(n,x,'k'); xlabel('n'); ylabel('x(n)');
subplot(3,2,4); stem(n,r(1,64:127), 'k');
```

![Figure 4.5](image-url)
A comparison between the PSD’s of the noisy signal \( x \) and its autocorrelation is shown in Figure 4.6.

The summary of correlation and spectral properties of stationary random sequences is given in Table 4.2.

**Figure 4.6**

**TABLE 4.2**

Summary of Correlation and Spectral Properties of Stationary Sequences

<table>
<thead>
<tr>
<th>Definitions</th>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean value</td>
<td>( m_x = E{x(n)} )</td>
</tr>
<tr>
<td>Autocorrelation</td>
<td>( r_{xx}(m) = E{x(n)x^*(n-m)} )</td>
</tr>
<tr>
<td>Autocovariance</td>
<td>( c_{xx}(m) = E{[x(n) - m_x][x(n-m) - m_x]^*} )</td>
</tr>
<tr>
<td>Cross-correlation</td>
<td>( r_{xy}(m) = E{x(n)y^*(n-m)} )</td>
</tr>
<tr>
<td>Cross-covariance</td>
<td>( c_{xy}(m) = E{[x(n) - m_x][y(n-m) - m_y]^*} )</td>
</tr>
<tr>
<td>Power spectral density</td>
<td>( S_x(e^{j\omega}) = \sum_{m=-\infty}^{\infty} r_{xx}(m)e^{-j\omega m} )</td>
</tr>
<tr>
<td>Cross-power spectral density</td>
<td>( S_{xy}(e^{j\omega}) = \sum_{m=-\infty}^{\infty} r_{xy}(m)e^{-j\omega m} )</td>
</tr>
</tbody>
</table>

*(Continued)*
TABLE 4.2  
(Continued) Summary of Correlation and Spectral Properties of Stationary Sequences

<table>
<thead>
<tr>
<th>Interrelations</th>
<th>Auto-PSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{xx}(m) = r_{xx}(m) - m_x m_x^\ast = r_{xx}(m) - m_x^2$</td>
<td>$S_x(e^{j\omega}) \geq 0$ and real</td>
</tr>
<tr>
<td>$r_{yy}(m) = r_{yy}(-m)$</td>
<td>$S_y(e^{j\omega}) = S_y(e^{-j\omega})$ [real $x(n)$]</td>
</tr>
<tr>
<td>$</td>
<td>r_{xx}(m)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cross-Correlation</th>
<th>Auto-PSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{yy}(m) = r_{yy}(-m)$</td>
<td>$S_y(z) = S_y^\ast(1 / z^\ast)$</td>
</tr>
<tr>
<td>$</td>
<td>r_{xy}(m)</td>
</tr>
</tbody>
</table>

4.5  FILTERING RANDOM PROCESSES

LTI filters are used in many signal processing applications. Since the input signals of these filters are usually random processes, we need to determine how the statistics of these signals are modified as a result of filtering.

Let $x(n)$, $y(n)$, and $h(n)$ be the filter input, the filter output, and the filter impulse response, respectively. It can be shown that if $x(n)$ is a WSS process, then the filter output autocorrelation $r_{yy}(k)$ is related to the filter input autocorrelation $r_{xx}(k)$ as follows (see Problem 4.5.1):

$$r_{yy}(k) = \sum_{l=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(l)r_{xx}(m-l+k)h(m)$$

$$= r_{xx}(k) * h(k) * h(-k) = r_{xx}(k) * r_{hh}(k)$$

$$h(k) * h(-k) = \text{Convolution between } h(k) \text{ and its reflected form } h(-k)$$

$$= \text{Autocorrelation of } h(k)$$

The right-hand side of the above expression shows convolution of three functions. We can take the convolution of two of the functions and the resulting function is then convolved with the third function. Based on the convolution property discussed in Chapter 3, the results are independent of the order we operate on the functions.

From the $z$-transform table, we know that the $z$-transform of the convolution of two functions is equal to the product of their $z$-transform. Remembering the definition of the $z$-transform, we find the relationship (the order of summation does not change the results)
\[ \mathcal{F}\{h(-k)\} = \sum_{k=-\infty}^{\infty} h(-k)z^{-k} = \sum_{m=0}^{\infty} h(m)(z^{-1})^{-m} = H(z^{-1}) \]  

(4.37)

since the summation is the same regardless of the direction in which we sum the series.

Therefore, the z-transform of (4.36) becomes

\[ R_y(z) = \mathcal{F}\{r_y(k) * h(k)\} = \mathcal{F}\{h(-k)\} = R_x(z)H(z) \]  

(4.38)

If we set \( z = e^{j\omega} \) in the definition of the z-transform of a function, we obtain the spectrum of the function. Therefore, the Wiener–Khinchin theorem (4.38) becomes

\[ S_y(e^{j\omega}) = S_x(e^{j\omega})|H(e^{j\omega})|^2 \]  

(4.39)

Note: The above equation indicates that the power spectrum of the output random sequence is equal to the power spectrum of the input sequence modified by the square of the absolute value of the spectrum of the filter transfer function.

Example 4.5.1

A first-order finite impulse response (FIR) filter is defined in the time domain by the difference equation \( y(n) = x(n) + 0.8x(n-1) \). If the input signal is \( N(0,1) \), find the PSD of the filter output.

Solution: Taking into consideration the z-transform properties of linearity and time shifting, the z-transform of the difference equation is \( Y(z) = (1 + 0.8z^{-1})X(z) \). Therefore, the transfer function is given by \( H(z) = \frac{Y(z)}{X(z)} = 1 + 0.8z^{-1} \). The absolute value square of the spectrum of the transfer function is

\[ |H(e^{j\omega})|^2 = (1 + 0.8e^{j\omega})(1 + 0.8e^{-j\omega}) = 1.64 + 0.8(e^{j\omega} + e^{-j\omega}) = 1.64 + 0.8\cos(\omega) + j\sin(\omega) = 1.64 + 1.6\cos(\omega) \]

Therefore, the PSD of the output is given by

\[ S_y(e^{j\omega}) = S_x(e^{j\omega})|H(e^{j\omega})|^2 = S_x(e^{j\omega})(1.64 + 1.6\cos(\omega)) \]

Figure 4.7 shows the requested results. Remember that the spectrum is valid in the range \( 0 \leq \omega \leq \pi \). We remind the reader that if the sequence was coming from a continuous function sampled at times \( T \), the spectrum range would have been \( 0 \leq \omega (\text{rad/s}) \leq \pi / T \). The following Book.m file produces Figure 4.7.

**Book.m File: ex4_5_1**

```matlab
%Book m-file: ex4_5_1;
x = randn(1,128); %x is N(0,1);
rx = xcorr(x,'biased');
sx = fft(rx(1,128:255),128);
w = 0:2*pi/128:2*pi-(2*pi/128);
```
The reader should observe that the filter is a low-pass one since it attenuates the high frequencies (frequencies close to $\pi$).

\[ H(z) = \frac{b(0)}{1 + \sum_{k=1}^{p} a(k)z^{-k}} = \frac{b(0)}{A(z)} \]  

\[ (4.40) \]
Using (4.39), the PSD of the output of this type of a filter, when the input \( \{v(n)\} \) is a
WN with zero mean and variance \( \sigma_v^2 \), is
\[
S_y(e^{j\omega}) = \sigma_v^2 \left| H(z)H(z^{-1}) \right|_{z=e^{j\omega}} = \sigma_v^2 \frac{b(0)^2}{A_y(e^{j\omega})^2} \tag{4.41}
\]

It can be shown that the following correlation relationship exists for the AR process
with a white input noise having zero mean value zero and variance \( \sigma_v^2 \):
\[
r_{yy}(k) + \sum_{m=1}^{n} a(m)r_{yy}(k-m) = \sigma_v^2 b(0)^2 \delta(k) \tag{4.42}
\]

The above equation can be written in the form:
\[
\begin{bmatrix}
    r_{yy}(0) & r_{yy}(-1) & \cdots & r_{yy}(-p) \\
    r_{yy}(1) & r_{yy}(0) & \cdots & r_{yy}(-p+1) \\
    \vdots & \vdots & \ddots & \vdots \\
    r_{yy}(p) & r_{yy}(p-1) & \cdots & r_{yy}(0)
\end{bmatrix}
\begin{bmatrix}
    a(1) \\
    \vdots \\
    \vdots \\
    a(p)
\end{bmatrix}
= \sigma_v^2 b(0)^2 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.43}
\]

For real sequences, the autocorrelation is symmetric: \( r_{yy}(k) = r_{yy}(-k) \).

**Example 4.6.1**

Find the AR coefficients \( a(1) \) to \( a(5) \) if the autocorrelation of the observed signal
\( \{y(n)\} \) is given. Let \( \{v(n)\} \) be a WN with mean 0 and variance 1, which is the input
to the filter \( y(n) = 0.8y(n-1) + 0.4y(n-2) + v(n) \).

**Solution:** The results are shown in Figure 4.8 and the Book m-file is given below.
Two of the figures were plotted in a continuous way to show the results better. First
we find the \( a \)'s using the system:
\[
\begin{bmatrix}
    r_{yy}(1) & r_{yy}(0) & \cdots & r_{yy}(-p+1) \\
    \vdots & \vdots & \ddots & \vdots \\
    r_{yy}(p) & r_{yy}(p-1) & \cdots & r_{yy}(0)
\end{bmatrix}
\begin{bmatrix}
    1 \\
    a(1) \\
    \vdots \\
    a(p)
\end{bmatrix}
= \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.44}
\]

Then, we find \( b(0)^2 \) from the following equation [see (4.43) first row], assuming
that the variance of the input to the system has variance 1:
\[
[r_{yy}(0) \ r_{yy}(-1) \ \cdots \ r_{yy}(-p)][1 \ a(1) \ \cdots \ a(p)]^T = b(0)^2 \tag{4.45}
\]

**Book m-File: ex4_6_1**

```matlab
%Book m-file: ex4_6_1;
x = (rand(1,1000)-0.5)*3.5;
```
% distributed WGN of zero mean
% and unit variance;
\[ y = \text{filter}(1, [1 -0.8 0.4], x); \] % y = observed signal;
\[ [\text{ry1}, \text{lags}] = \text{xcorr}(y, y, 15, 'biased'); \] % \text{ry1} = autocorrelation of y;
\[ \text{ry} = \text{ry1}(1, 16:31); \] % one sided;
\[ \text{Ry} = \text{toeplitz}(\text{ry}(1, 1:6)); \] % \text{Ry} = autocorrelation matrix of the output;
\[ \text{Rya} = \text{Ry}(2:6, 2:6); \] % creates a 5x5 matrix from \text{Ry} matrix;
\[ a = \text{inv}(\text{Rya}) \ast (\text{Ry}(2:6, 1) \ast (-1)); \] % solves for a's from the lower part of the system which is given in the example;
\[ \text{b02} = [\text{Ry}(1, 1:6)] \ast [1 \ a']'; \]
\[ \text{H} = \text{b02} ./ (\text{fft}([1 \ a'], 512)); \] % spectrum of the system function;
\[ \text{SH} = \text{H} \ast \text{conj(H)}/512; \] % PSD of the transfer function;
\[ \text{nn} = 0:511; w = 2*\pi*\text{nn}/512; \]
\[ \text{subplot}(2, 2, 1); \text{stem}(\text{x}(1, 1:50), 'k'); \text{xlabel}('n'); \text{ylabel}('x(n)'); \]
\[ \text{subplot}(2, 2, 2); \text{plot}(w, \text{abs(fft(y(1, 1:512), 512))), 'k'); \text{xlabel}('\text{Spectrum of } y(n)'); \text{ylabel}('r_{yy}(k)'); \]
\[ \text{subplot}(2, 2, 3); \text{plot}(w, \text{abs(fft(y(1, 1:512), 512))), 'k'); \text{xlabel}('\omega'); \text{ylabel}('PSD of \text{H}(\omega)'); \]
\[ \text{subplot}(2, 2, 4); \text{plot}(w, \text{abs(fft(y(1, 1:512), 512))), 'k'); \text{xlabel}('\omega'); \text{ylabel}('PSD of \text{H}(\omega)'); \]
### 4.7 NONPARAMETRIC SPECTRA ESTIMATION

The spectra estimation problem in practice is based on a finite-length record \( \{x(1), \ldots, x(N)\} \) of a second-order stationary random process. However, harmonic processes have line spectra and appear in applications either alone or combined with noise.

#### 4.7.1 Periodogram

The **periodogram spectral estimator** of the data segment \( \{x(0), x(1), \ldots, x(N-1)\} \), finite number of data, is based on the following formula:

\[
\hat{S}_x(e^{j\omega}) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x_w(n)e^{-j\omega n} \right|^2 = \frac{1}{N} |X(e^{-j\omega})|^2
\]  

(4.46)

where:

- \( X(e^{-j\omega}) \) is the DTFT of the windowed sequence \( x_w(n) = x(n)w(n) \), \( 0 \leq n \leq N-1 \)  

(4.47)

In (4.46), the signal can be multiplied by a unit value square window \( \{1, 1, 1, \ldots, 1\} \). The PSD of the signal, \( \hat{S}_x(e^{j\omega}) \), is periodic with period \( 2\pi, -\pi \leq \omega \leq \pi \). The spectrum \( X(e^{j\omega}) \) is the DTFT of the windowed signal \( x_w(n) \). The periodicity is easily shown by introducing \( \omega + 2\pi \) in place of \( \omega \) in (4.46) and remembering that \( \exp(j2\pi) = 1 \).

The values of the periodogram at discrete points in the frequency domain are located at \( \left\{ \frac{\omega_k}{N} \right\}_{k=0}^{N-1} \) and are found by

\[
\hat{S}_x(k) \triangleq \hat{S}_x(e^{j2\pi k/N}) = \frac{1}{N} |X(\omega_k)|^2 \quad k = 0, 1, \ldots, N-1 \quad \omega_k = \frac{2\pi k}{N}
\]  

(4.48)

where:

- \( X(k) \) is the \( N \)-point DFT of the windowed signal \( x_w(n) \)

It is recommended that we multiply the random signal with an appropriate window for a smoother PSD. The following example and the Book m-file explain the use of the windows. Some of the MATLAB window functions are given below. See also Appendix 1 at the end of the book for more types of windows.

MATLAB functions for several types of windows are as follows:

- @blackmanharris—Minimum 4-term Blackman–Harris window
- @gausswin—Gaussian window
- @hamming—Hamming window
- @rectwin—Rectangular window

An example of MATLAB program using window functions is given as follows:

```matlab
N = 64;
w = window(@blackmanharris,N);
w1 = window(@hamming,N);
w2 = window(@gausswin,N,2.5);
plot(1:N,[w,w1,w2]); axis([1 N 0 1]);
legend('Blackman-Harris','Hamming','Gaussian');
```
Example 4.7.1

Plot the PSD of the signal \(y(n)\) that is produced by the filter \(y(n) - 0.9y(n-1) + 0.2y(n-2) = v(n)\), where \(v(n) = 0.8 \cdot \text{randn}(n)\). Use a rectangular and a Hamming window with \(N = 128\).

Solution: The results are shown in Figure 4.9 using the following Book m-file.

Book m-File: ex4_7_1

```matlab
%Book m-file: ex4_7_1;
y(1) = 0;y(2) = 0;
for n = 3:1024
    y(n) = 0.9*y(n-1)-0.2*y(n-2)+0.8*randn;
end;
N = 128;
xw = y(1,50:50+N-1).*window(@rectwin,N)';%the window
    %is given as a column vector, here is
    %a rectangular window;
psd = (abs(fft(xw,N))).^2/N;
xw1 = y(1,1:N).*window(@hamming,N)';
psd1 = (abs(fft(xw1,N))).^2/N;
sx128 = psd(1,1:N);
sx1128 = psd1(1,1:N);
om = 0:2*pi/N:2*pi-(2*pi/N);
subplot(2,1,1);stem(om,sx128,'k');
xlabel('k');ylabel('PSD');
subplot(2,1,2);stem(om,sx1128,'k');
xlabel('k');ylabel('PSD');
```

![Figure 4.9](image-url)
The reader will observe that the Hamming window (and any other one) produces a smoother PSD than the rectangular one. We must point out that the results showing in the figure is only for one realization. This means that any time we run the program we will observe different PSDs. We can also proceed to create a number of these realizations and then average them.

4.7.2 Correlogram

The correlogram spectral estimator is given by the formula:

$$\hat{S}_r(e^{j\omega}) = \sum_{m=-(N-1)}^{N-1} \hat{r}_xx(m)e^{-j\omega m}$$

where:

$\hat{r}_xx(m)$ is the estimate of the biased autocorrelation (assumed mean value of $\{x(n)\}$ to be 0) given by (4.11)

It can be shown that the correlogram spectral estimator, evaluated using the standard biased autocorrelation estimates, coincides with that of the periodogram spectral estimator. As in (4.46), the correlogram is a periodic function of $\omega$ with period $2\pi$.

4.7.3 Computation of Periodogram and Correlogram Using FFT

Since both functions are continuous with respect to frequency (DTFT), we can sample the frequency as follows:

$$\omega_k = \frac{2\pi}{N} k \quad k = 0, 1, 2, \ldots, N-1$$

Introducing the above equation in (4.46) and (4.49), we obtain

$$\hat{S}_r(e^{j\omega}) = \frac{1}{N}\left|X(e^{j\omega})\right|^2; \quad \hat{S}_r(e^{j\omega}) = \sum_{m=-(N-1)}^{N-1} \hat{r}_xx(m)e^{-\frac{2\pi}{N} km} \quad 0 \leq k \leq N-1$$

$$X(e^{j\omega}) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N} nk} = \sum_{n=0}^{N-1} x(n)W^{nk} \quad W = e^{-j2\pi/N} \quad 0 \leq k \leq N-1$$

The most efficient way to find the DFT using fast Fourier transform (FFT) is to set $N = 2^r$ for some integer $r$. The following two Book m-functions give the windowed periodogram and correlogram.

Book m-Function for Windowed Periodogram: \[s,as,phs\] = lmsperiodogram \(x,w,L\)

```matlab
function \[s,as,phs\] = lmsperiodogram(x,w,L)
%w = window(@name,length(x)),w is in column form
%see also Appendix 1
%name = hamming,kaiser,hann,rectwin,
```
Discrete-Time Random Processes

%bartlett,tukeywin,blackman,gausswin,nattallwin,
%triang,blackmanharris);
%L = desired number of points (bins) of the spectrum;
%x = data in row form;s = complex form of its DFT;

```
xw = x.*w';
for m = 1:L
    n = 1:length(x);
    s(m) = sum(xw.*exp(-j*(m-1)*(2*pi/L)*n));
end; %the for loop finds the set of complex numbers
%of the summation for each m;
as = ((abs(s)).^2/length(x))/norm(w);
%as = amplitude value of the periodogram;
phs = (atan(imag(s)./real(s))/length(x))/norm(w);
%phase of the periodogram;
wb = 0:2*pi/L:2*pi-(2*pi/L); stem(wb,as,'k');
xlabel('Frequency bins, 2\pi/L'); ylabel('Magnitude');
```

**Book m-Function for Windowed Correlogram: [s,as,ps] = lms_correlogram(x,w,L)**

```
x = data with mean zero;w = window(@name,length(2*lg)),
%L = desired number of spectral points;
%lg = lag number<<N;rc = symmetric autocorrelation
%function;
r = lms_sample_biased_autoc(x,lg);
rcw = rc.*w';
for m = 1:L
    n = -lg+1:lg;
    s(m) = sum(rcw.*exp(-j*(m-1)*(2*pi/L)*n));
end;
as = (abs(s).^2)/norm(w)/L;%amplitude spectrum;
ps = (atan(imag(s))/real(s))/norm(w);%phase spectrum;
stem(0:2*pi/L:2*pi-(2*pi/L),as,'k');xlabel('\omega(k)');
ylabel('Correlogram');
```

To plot, for example, as or ps, we can use the command: plot (0:2*pi/L:2* pi-(2*pi/L),as).

**Note:** In case the reader does not want to apply window, we write `w = window(@rectwin,length(2*lg)')` or `w = rectwin(2*lg)'`.

### 4.7.4 General Remarks on the Periodogram

The general remarks on the periodogram are as follows:

1. The variance of the periodogram does not tend to 0 as \( N \to \infty \). This indicates that the periodogram is an **inconsistent** estimator; that is, its distribution does not tend to cluster more closely around the true spectrum as \( N \) increases.
2. To reduce the variance and, thus, produce smoother spectral estimator, we must obtain
   a. The average contiguous values of the periodogram.
   b. The average periodogram obtained from multiple data segments.

3. The effect of the side lobes of the windows on the estimated spectrum consists of transferring power from strong bands to less strong bands or bands with no power. This process is known as the **leakage** problem.

Figure 4.10 shows a plot of both the periodogram and the correlogram.

**Example 4.7.2**

It is instructive to obtain the average of a number of realizations of a periodogram.

**Solution:** The signal is the sum of two sinusoids and a WN of unit variance. Using the Book m-file given below, the results are shown in Figure 4.11. The top plot shows 50 realizations of the signal, the second shows 50 realizations of the periodograms, the third shows the average PSD of all 50 PSDs, and the last one shows just the PSD of one of the realizations.

**Book m-File: ex4_7_2**

```matlab
%Book m-file: ex4_7_2;
for m = 1:60
    n = 0:127;
    x(m,:) = sin(0.35*pi*n)+sin(0.37*pi*n)...+3.5*(rand(1,128)-0.5);
    subplot(4,1,1);plot(x(m,1:64),'k');xlabel('n');
end
```

![Figure 4.10](image-url)
Discrete-Time Random Processes

4.7.4.1 Windowed Periodogram

We can define a window periodogram as the DTFT of the time function \( \{x(n)w(n)\} \) multiplied by a window, where \( w(n) \) is any desired window function. Hence, the periodogram with temporal window is given by

\[
\hat{S}_x(e^{j\omega}) = \frac{1}{NP_w} \left| \sum_{n=0}^{N-1} x(n)w(n)e^{-j\omega n} \right|^2
\]

where:

\( P_w \) is the average power of the window
Example 4.7.3

Apply the rectangular and Hamming windows to the signal given below and compare the resulting spectra:

\[ x(n) = \{0.06 \sin(0.2\pi n) + \sin(0.3\pi n) + 0.05[\text{rand}(1,128) - 0.5]\}w(n) \quad 0 \leq n \leq 127 \]

**Solution:** The window provides a trade-off between spectra resolution (main lobe) and spectral masking (side lobe amplitude). In this case, the Hamming window has a side lobe at about \(-45\) dB compared with the rectangular window having a side lobe at about \(-13\) dB. Figure 4.12 shows that although the side lobe of the rectangular window just about obscures the \(0.2\pi\) frequency, the Hamming window clearly resolves it.

It can be shown that that the amount of smoothing in the periodogram is determined by the type of the window that is applied to the data. Selecting the type of window, we must weigh our intention between resolution and smoothing.

The following Book m-function calculates the temporal windowed data with the ability to introduce any one of the several windows that are given in the function without the obligation to introduce it in the Command window. This helps if one wants to repeat the simulation using different windows.

**Book m-Function That Produces Windowed Periodograms**

```matlab
%Book m-function: [s,as,ps] = lmsperiodogram(x,win,L);
function [s,as,ps] = lmsperiodogramwin(x,win,L)
%window names = hamming,kaiser,hann,rectwin,
%bartlett,tukeywin,blackman,nattallwin,
%triang,blackmanharris;
```
Discrete-Time Random Processes

%L = desired number of points (bins) of the spectrum;
%x = data in row form; s = complex form of the DFT;

if (win = =2) w = rectwin(length(x));
elseif (win = =3) w = hamming(length(x));
elseif (win = =4) w = bartlett(length(x));
elseif (win = =5) w = tukeywin(length(x));
elseif (win = =6) w = blackman(length(x));
elseif (win = =7) w = triang(length(x));
elseif (win = =8) w = blackmanharris(length(x));
end;
xw = x.*w';

for m = 1:L
    n = 1:length(x);
    s(m) = sum(xw.*exp(-j*(m-1)*(2*pi/L)*n));
end;
as = ((abs(s)).^2/length(x))/norm(w);%as = amplitude spectral density;

ps = (atan(imag(s)/real(s))/length(x))/norm(w);%ps = phase spectrum;

%To plot as or ps we can use the command:
%plot(0:2*pi/L:2*pi-(2*pi/L),as);

4.7.5 Proposed Book Modified Method for Better Frequency Resolution

4.7.5.1 Using Transformation of the rv’s

One of the difficulties is that if the sequence is short (N small number), we may not be able to sufficiently resolve frequencies being close together. Since we have at hand one realization and we cannot extract another one from the population in the probability space, we propose to create a pseudosequence from the data at hand by linear transformation of the rv’s. Next, we overlap (about 25%) these two series. These series can also be multiplied by a window and then processed.

Note: The proposed Book Modified method is based on the linear transformation of rv’s.

Example 4.7.4

Compare the spectrum based on the periodogram and the proposed modified one.

Solution: Figure 4.13 shows the resolution capabilities of the sequence that is made up from the original sequence as follows: y = 0.2* [xzeros(1,48)] + 0.2*[zeros(1,48)x]. The original signal, a 64-term series, was x = sin(0.3*pi*n) + sin(0.32*pi*n) + 0.2 * randn (1,64). The following Book m-function produces the figure.

Book m-Function Linearly Modified Signal: [ax,ay,w,y] = lmslinear_modified_periodogram(x)

function [ax,ay,w,y] = lmslinear_modified_periodogram(x)
    %Book MATLAB function:
Adaptive Filtering

% [ax, ay, w]: lmslinear_modified_periodogram(x);
[sx, ax, px] = lmsperiodogramwin(x, 2, 512);
y1 = [x zeros(1, 48)] + 0.1*rand(1, 112);
y2 = [zeros(1, 48) x] + 0.1*rand(1, 112);
y = 0.2*y1 + y2*0.2;
[sy, ay, py] = lmsperiodogramwin(y, 2, 512);
w = 0:2*pi/512:2*pi-(2*pi/512);
% 2 implies rectangular window, see Book
% MATLAB function lmsperiodogramwin
% for other windows; to plot we must
% write in the command window:
% plot(w, 20*log10(ax/max(ax)),'k') and
% similar for the ay;

4.7.5.2 Blackman–Tukey Method
Because the correlation function at its extreme lag values is not reliable due to the small overlapping of the correlation process, it is recommended to use lag values of about 30%–40% of the total length of the data. The Blackman–Tukey (BT) estimator is a windowed correlogram and is given by

$$\hat{S}_{BT}(e^{j\omega}) = \sum_{m=-(L-1)}^{L-1} \hat{r}_x(m)w(m)e^{-j\omega m}$$  (4.53)
where:  
\( w(m) \) is the window with 0 value for \(|m| > L - 1\) and \( L << N \)

The above equation can also be written in the form:

\[
\hat{S}_{BT}(e^{j\omega}) = \sum_{m=-\infty}^{\infty} \tilde{\rho}_{xx}(m)w(m)e^{-j\omega m} 
= \hat{S}_e(e^{j\omega}) \ast W(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{S}_e(e^{j\tau})W(e^{j(\omega-\tau)})d\tau
\]

(4.54)

where we applied the DTFT frequency convolution property (the DTFT of the multiplication of two functions is equal to the convolution of their Fourier transforms). Since windows have a dominant and relatively strong main lob, the BT estimator corresponds to a “locally” weighting average of the periodogram. Although the convolution smooths the periodogram, it reduces the resolution at the same time. It is expected that the smaller the \( L \), the larger the reduction in variance and the lower the resolution. It turns out that the resolution of this spectral estimator is on the order of \( 1/L \), whereas its variance is on the order of \( L/N \).

For convenience, we give again some of the most common windows below. For the Kaiser window, the parameter \( \beta \) trades the main lobe width for the side-lobe leakage; \( \beta = 0 \) corresponds to a rectangular window, and \( \beta > 0 \) produces lower side lobe at the expense of a broader main lobe.

1. **Rectangular window**
   
   \( w(n) = 1 \quad n = 0, 1, 2, \cdots, L - 1 \)

2. **Bartlett (triangle) window**

   \[
   w(n) = \begin{cases} 
   \frac{n}{L/2} & n = 0, 1, \cdots, L/2 \\
   \frac{L-n}{L/2} & n = L/2 + 1, \cdots, L-1 
   \end{cases}
   \]

3. **Hann window**

   \[
   w(n) = 0.5 \left[ 1 - \cos \left( \frac{2n}{L} \pi \right) \right] \quad n = 0, 1, 2, \cdots, L - 1
   \]

4. **Hamming window**

   \[
   w(n) = 0.54 - 0.46 \cos \left( \frac{2\pi}{L} n \right) \quad n = 0, 1, 2, \cdots, L - 1
   \]

5. **Blackman window**

   \[
   w(n) = 0.42 + 0.5 \cos \left( \frac{2\pi}{L} \left( n - \frac{L}{2} \right) \right) \\
   + 0.08 \cos \left( \frac{2\pi}{2} \left( n - \frac{L}{2} \right) \right) \quad n = 1, 2, \cdots, L - 1
   \]
6. Kaiser window

\[
I_0 \left( \sqrt{1.0 - \left( \frac{n}{L/2} \right)^2} \right) - (L - 1) \leq n \leq L - 1
\]

\[
I_0(x) = \sum_{k=0}^{\infty} \left( \frac{x^2}{k!} \right)^{\beta}
\]

= Zero-order modified Bessel function

The windows \( w(k) = 0 \) for \( |k| \geq L \) and \( w(k) = w(-k) \), and the equations are valid for \( 0 \leq k \leq L - 1 \).

The following Book m-function produces the spectrum under the BT method. Figure 4.14 shows the effect of averaging of different lengths of the autocorrelation function that was proposed in this chapter. The signal was produced as follows: \( n = 0:127 \), \( x = \sin(0.3\pi n) + \sin(0.33\pi n) + \text{randn}(1,128) \).

**Book m-Function:** \([rx,asn,ps] = \text{lmsBTperiodogram}(x,\text{win},L,lg)\)

```matlab
function [rx,asn,ps] = lmsBTperiodogram(x,win,L,lg)
% [rx,asn,ps] = lmsBTperiodogram(x,win,L,lg);
%window names = hamming,kaiser,hann,rectwin,
```

**FIGURE 4.14**
Discrete-Time Random Processes

% bartlett, tukeywin, blackman, gausswin, nattallwin, triang,
% blackmanharris; lg = lag number;
% L = desired number of points (bins) of the spectrum;
% x = data in row form; s = complex DFT data;

rx = lms_sample_biased_autoc(x, lg);
if (win = =2) w = rectwin(2*length(rx));
elseif (win = =3) w = hamming(2*length(rx));
elseif (win = =4) w = bartlett(2*length(rx));
elseif (win = =5) w = tukeywin(2*length(rx));
elseif (win = =6) w = blackman(2*length(rx));
elseif (win = =7) w = triang(2*length(rx));
elseif (win = =8) w = blackmanharris(2*length(rx));
end;
wh = w(length(rx):2*length(rx)-1,1)';
rxw = (rx.*wh);
for m = 1:L
    n = 1:length(rx);
    s(m) = sum(rxw.*exp(-j*(m-1)*(2*pi/L)*n));
end;
as = (abs(s)).^2/norm(w);% as = amplitude spectral density;
asn = as/max(abs(as));
ps = atan((imag(s)./(real(s)+eps))/((length(x))/...
    (norm(w)+eps)));
% ps = phase spectrum;
To plot as or ps we can use the command:
plot(0:2*pi/L:2*pi-(2*pi/L),as);

Another Book proposed method is based on linearly extending the short windowed signal. The following Book m-file is used to produce Figure 4.15.

Book m-File: fig4_15

% Book m-file: fig4_15;
n = 0:63;
x = sin(0.3*pi*n)+sin(0.32*pi*n)+0.5*randn(1,64);
y = [0.4*x 0.2*x+0.05].*hamming(128)';
[wh,rx,rxw,asn,psx] = lmsBTperiodogram(x, 2, 512, 64);
[why,ry,ryw,asny,psy] = lmsBTperiodogram(y, 2, 512, 64);
om = 0:2*pi/512:2*pi-(2*pi/512);
subplot(2,1,1);plot(om, asnx,'k');xlabel('omega per unit');
ylabel('BT PSD');
subplot(2,1,2);plot(om, asny,'k');xlabel('omega per unit');
ylabel('BT Modif. PSD');

Further modification can be achieved by the proposed BT method using the combination of the rv’s transformation and averaging. The Book m-function is given as follows:

Book m-Function for Modified BT method Using rv’s Transformation and Averaging: [rz,az,pz] = lmsav_modifiedBTpsd(x,win,avn,L,lg)

function [sz,az,pz] = lmsav_modifiedBTpsd(x, win, avn, L, lg)
% win = 2 implies rectangular, see the function
for m = 1:avn
    z1(m,:) = [x*rand*0.5 x*rand*0.2+
    0.05*(rand-0.5)].*hamming(2*length(x))';
end;
z = sum(z1,1);
[rz,az,pz] = lmsBTperiodogram(z,win,L,lg);

The top of Figure 4.16 is the BT periodogram with Hamming window. The data
were as follows: x = sin(0.3πn) + sin(0.33πn) + 0.8randn(1,128), lg = 64, L = 256,
Hamming window, and number of averaging 40. The bottom figure is the proposed
modified average BT periodogram. The reader will observe that by increasing the
resolution the variance also increases.

4.7.6 BARTLETT PERIODOGRAM

The variance of the periodogram presupposes that an ensemble averaging for vari-
ance decreases. However, for a single realization, the variance does not decrease as
N → ∞ (inconsistent estimator), and hence, there is a need for other approaches to
reduce the variance. We can improve the statistical properties of the periodogram by replacing the expectation operator with averaging a set of periodograms. Figure 4.17 shows the ensemble averaging effect on the variance using 10 realizations of the PSD only. The variance of one realization is about 55 and the variance of the ensemble is about 2.5. The number of realizations in this case was 30. The following Book m-file produces Figure 4.17.

**Book m-File: fig4_17**

```matlab
%Book m-file: fig4_17;
N = 256;
for m = 1:30
    x(m,1:N) = randn(1,N);
end;
w = 0:2*pi/N:2*pi-(2*pi/N);
subplot(2,1,1);plot(w,20*log10(abs(fft(...
    (x(10,:),N))),'k');
xlabel('\omega per unit');ylabel('PSD');
subplot(2,1,2);plot(w,20*log10(abs(fft...
    (sum(x,1),N)))/30,'k');
xlabel('\omega per unit');ylabel('Ave. PSD');
```

**FIGURE 4.16**

reduce the variance. We can improve the statistical properties of the periodogram by replacing the expectation operator with averaging a set of periodograms. Figure 4.17 shows the ensemble averaging effect on the variance using 10 realizations of the PSD only. The variance of one realization is about 55 and the variance of the ensemble is about 2.5. The number of realizations in this case was 30. The following Book m-file produces Figure 4.17.

**Book m-File: fig4_17**

```matlab
%Book m-file: fig4_17;
N = 256;
for m = 1:30
    x(m,1:N) = randn(1,N);
end;
w = 0:2*pi/N:2*pi-(2*pi/N);
subplot(2,1,1);plot(w,20*log10(abs(fft(...
    (x(10,:),N))),'k');
xlabel('\omega per unit');ylabel('PSD');
subplot(2,1,2);plot(w,20*log10(abs(fft...
    (sum(x,1),N)))/30,'k');
xlabel('\omega per unit');ylabel('Ave. PSD');
```
Bartlett proposed to split the data \( \{ x(n) \} \) into \( K \) nonoverlapping segments, which is assumed to be statistically independent. This condition presupposes that the autocorrelation of each segment decays much faster than the length of the segments. The \( N \) samples of the data signal are divided into \( K \) nonoverlapping segments of \( M \) samples each such that \( KM \leq N \). The \( i \)th segment will then consist of the samples:

\[
x^{(i)}(n) = x(iM + n) \quad n = 0, 1, \ldots, M - 1
\]

\( i = \text{segment number} = 0, 1, \ldots, K - 1 \)  

(4.55)

Each of the segments has the spectrum

\[
\hat{S}_{x}^{(i)}(e^{j\omega}) = \frac{1}{M} \sum_{n=0}^{M-1} x(iM + n)e^{-j\omega n}
\]

(4.56)

and hence, the Bartlett average periodogram is given by

\[
\hat{S}_{xb} = \frac{1}{K} \sum_{i=0}^{K-1} \hat{S}_{x}^{(i)}(e^{j\omega})
\]

(4.57)

The diagrammatic presentation of the above Bartlett equation is shown in Figure 4.18. Since the power spectrum is due to the reduced number of terms, \( M \ll N \), compared to the total available data, the resolution reduces from the order of \( 1/N \) to the order...
of $1/M$. This implies that the resolution is reduced by $N/M = K$ compared to the resolution of the original data set. It can be shown that the variance is reduced by the same amount.

The following Book m-function produces the Bartlett periodogram with any desired window:

**Book m-Function:**\[ s, as, ps \] = lmsbartlettpsd(\( x,k,win,L \))

```matlab
function[s,as,ps] = lmsbartlettpsd(x,k,win,L);
    %x = data;k = number of sections;
    %L = number of points desired in the FT domain;
    %M = number of points in each section; kM< = N = length(x);
    %the number win identifies the type of %window;
    M = floor(length(x)/k);
    s = 0;
    ns = 1;
    for m = 1:k
        s = s+lmsperiodogramwin(x(1,ns:ns+M-1),win,L);
        ns = ns+M;
    end;
    as = ((abs(s)/k).^2)/length(x);
    ps = (atan(imag(s/k)./real(s/k)))/length(x);
end;
```

Figure 4.19 shows the Bartlett PSD for two windows: the rectangular and the Hamming. We observe that the PSD using the Hamming window is smoother (smaller variance) than that using the rectangular window. The data used were as follows: $n = 0:127$, $\text{win} = 2$ and $3$, $L = 256$ (number of bins in the frequency domain), $k = 2$ (the length 128 of the signal was split into two equal sections with 64 points each) and the signal was $x = \sin(0.3\pi n) + \sin(0.315\pi n) + 0.5\text{randn}(1,128)$.
We propose three different Book m-functions that increase the resolution of the desired Bartlett-type spectrum as follows:

**Book m-Function for Modified Bartlett Spectrum-Modified No. 1**

```matlab
function [s,as,ps,apsd] = lms_aver_mod_bartlettpsd1(x,k,win,L,R)
    %x = data; k = number of sections; L = number of points
    %desired in the FT domain; M = number of points in each
    %section; kM< = N = length(x); R = number of realizations;
    %win = 2 for rectwin, 3 for hamming etc.,
    %see sspperiodogramwin function;
    %s, as and ps are RxL amtrices;apsd is
    %the average spectrum of the R realizations;
    %if desired we can plot the amplitude
    %spectrum of one realization:
    %plot(as(5,1:512));
    for r = 1:R
        xr(r,:) = (x + [zeros(1,floor(0.8*length(x))) + 0.1*rand(1,length(x))]).*2;
        [s(r,:),as(r,:),ps(r,:)] = lmsbartlettpsd(xr,k,win,L);
    end;
    apsd = sum(as,1)/R; %as is matrix and 1 means add the rows;
end;
```

**FIGURE 4.19**

![Rectangular window](image1.png)

![Hamming window](image2.png)
Figure 4.20 shows the result of the proposed Book m-file No. 1 given above. It shows that the proposed method has higher resolution capabilities than the Bartlett proposed approach.

**Book m-Function for Modified Bartlett Spectrum-Modified No. 2**

```matlab
function [s, as, ps, apsd] = ...
    lms_aver_mod_bartlettpsd2(x, k, win, L, R)
    %x = data; k = number of sections; L = number of points
    %desired in the FT domain; M = number of points in each
    %section; kM< = N = length(x); R = number of realizations;
    for r = 1:R
        x1(r,:) = [x*0.5+0.05*randn(1,length(x))...
                   x*0.5+0.05*randn(1,length(x))];
        [s(r,:), as(r,:), ps(r,:)] = lmsbartlettpsd(x1,k,win,L);
    end;
    apsd = sum(as,1)/R;
end;
```

Figure 4.21 shows the results of the No. 2 proposed modification. The input data were identical to the above two figures. We find similar results using the modified No. 3 given below.

**Book m-Function for Modified Bartlett Spectrum-Modified No. 3**

```matlab
function [s, as, ps, apsd] = ...
    lms_aver_mod_bartlettpsd3(x, k, win, L, R)
```
Adaptive Filtering

for m = 1:R
    x1(m,:) = [x 0.2*x+0.1*randn(1,length(x))];
end;

Welch proposed the modifications to Bartlett method as follows: Data segments are allowed to overlap and each segment is windowed prior to computing the periodogram. In most practical applications, we create smaller sections as follows:

\[ x_i(n) = x(iD + n)w(n) \quad 0 \leq n \leq M - 1 \quad 0 \leq i \leq K - 1 \]  

where:

- \( w(n) \) is the window of length \( M \)
- \( D \) is an offset distance
- \( K \) is the number of sections that the sequence \( \{x(n)\} \) is divided into

4.7.7 The Welch Method

![Rectangular window](image1)

![Hamming window](image2)
Discrete-Time Random Processes

Pictorially, the Welch method is shown in Figure 4.22. The $i$th periodogram is given by

$$\hat{S}_x(e^{j\omega}) = \frac{1}{M} \sum_{n=0}^{M-1} x_i(n)e^{-j\omega n}$$

Therefore, the average Welch periodogram is given by

$$\hat{S}_x(e^{j\omega}) = \frac{1}{K} \sum_{i=0}^{K-1} \hat{S}_x(e^{j\omega})$$

If $D = M$, then the segments do not overlap and the result is equivalent to the Bartlett method with the exception that the segments are windowed.

The MATLAB program for the Welch PSD is given as follows:

```matlab
ps = spectrum.welch(window,segmentlength,overlappercent) %window = 'hamming', we could have put rectangular etc;
%segmentlength is a number power of two, e.g. if length(x) = 128
%we put for segmentlength = 64 or 32 etc;
%we write overlappercent = 50 for 50%, 30 for 30% etc;
hpsd = psd(ps,x,'nfft',1024); %the DFT with 1024 bins, we can %use 2048 etc; 'nfft' is written as is given;
plot(hpsd); %spectrum.welch() and psd() are MATLAB functions
%and must be used as shown in the example below;
```
Example 4.7.5

Find the Welch spectrum for two lengths of the sections: 64 and 128. Create a signal with 256 elements, hamming window, and overlap 30%. Figure 4.23 shows the results.

Solution: In the Command window, we write

```matlab
>> n = 0:255;
>> x = sin(0.30*pi*n)+sin(0.30*pi*n)+0.5*randn(1,256);
>> ps = spectrum.welch('hamming',64,30);
>> hpsd = psd(ps,x,'nfft',1024);
>> subplot(2,1,1);
>> plot(hpsd);
```

By changing segmentlength = 128, we proceed the same way and find the PSD given in the bottom of Figure 4.23. The top figure corresponds to a length of 64.

Book m-Function for Welch PSD:

```matlab
function [s, ast, ps, K] = lmswelchpsd(x, win, frac, overlap, L)
    % x = data; M = section length; L = number of samples
    % desired in the frequency domain; win = 2 means rectwin,
```

![Welch PSD estimate](image_url)
if (win == 2) w = rectwin(floor(frac*length(x))); 
elseif (win == 3) w = hamming(frac*length(x)); 
end;

N = length(x);
M = floor(frac*length(x));
K = floor(floor((N-M+floor(overlap*M)))/floor(overlap*M));
%K = number of processings;
s = 0; as = 0;
for i = 1:K
s = s+fft(x(1,(i-1)*(floor(overlap*M))+1:(i-1)*floor(overlap*M)+M).*w',L);
as = as+abs(s);
end;
ast = as/(M*K); %as = amplitude spectral density;
ps = atan(imag(s)./real(s))/(M*K); %phase spectral density;

The data used for Figure 4.24 were as follows: rectangular window, 2; FFT length 512; $x = \sin(0.3\pi n) + \sin(0.32\pi n) + 0.5\text{randn}(1,256)$; $n = 1,2,...,256$. The following additional inputs were used for Figure 4.24a: length of sections is 0.25 or equivalently 64 elements and overlap is 0.25 or equivalently 16 elements overlapping. The following additional inputs were used for Figure 4.24b: length of sections is 0.50 or equivalently 128 elements and overlap is 0.25 or equivalently 32 elements. The following additional inputs were used for Figure 4.24c: length of sections is 0.25 or equivalently 64 elements and overlap is 0.50 or equivalently 32 elements. The following additional inputs were used for Figure 4.24d: length of sections is 0.50 or equivalently 128 elements and overlap is 0.50 or equivalently 64 elements.

4.7.8 **Proposed Modified Welch Methods**

4.7.8.1 Modified Method Using Different Types of Overlapping

It is evident from Figure 4.24 that if the lengths of the sections are not long enough and the overlaps are not enough, frequencies close together cannot be differentiated and the variance is large. Therefore, we propose a procedure defined as the **symmetric modified Welch method** (SMWM), and its implementation is shown in Figure 4.25. Windowing of the segments can also be incorporated. This approach and the rest of the proposed schemes have the advantage of progressively incorporating longer and longer segments of the data and thus introducing better and better resolution. In addition, due to the averaging process, the variance decreases and smoother periodograms are obtained but not as smooth as the Welch method. It is
We refer to Figure 4.24 for an illustration of the Welch PSD (Power Spectral Density) estimates for different data segments. Figure 4.24 shows four plots (a), (b), (c), and (d) illustrating the Welch PSD for different segments.

**Figure 4.24**

$$0 \rightarrow N - 1$$

Data

- Segment 1
- Segment 2

;  

Segment K

Peroidogram 1

+  

Peroidogram 2

+  

;  

Peroidogram K

=  

Total/K  

Averaging

PSD estimate

**Figure 4.25**
up to the reader to decide between smoothness of the periodogram and resolution of frequencies. Figure 4.26 shows another proposed method defined as the asymmetric modified Welch method (AMWM). Figure 4.27 shows another suggested approach for better resolution and reduced variance. The procedure is based on the method of prediction and averaging. This proposed method is defined as the symmetric prediction modified Welch method (SPMWM). This procedure can be used in all the other forms, for example, nonsymmetric. The above methods can also be used for spectral estimation if we substitute the word periodogram for the word correlogram.

4.7.8.2 Modified Welch Method Using Transformation of rv’s

Figure 4.28a shows the result of the Welch method for the signal \( x = \sin(0.3\pi n) + \sin(0.33\pi n) + 0.2\text{randn}(1,128) \), \( n = 0, 1, \ldots, 127 \). The signal was split into two sections with 50\% overlap. For Figure 4.28b, the following transformed signal was used by incorporating a linear transformation of the rv’s: \( z = [x \quad 0.1 \times x + 0.05 \times \text{randn}(1,128)] \)

For this signal, the Welch method was used splitting the series into two parts and using 50\% overlapping. The figure was obtained using the following Book MATLAB program:

```matlab
>> n = 0:127;
>> x = sin(0.30*pi*n)+sin(0.33*pi*n)+0.2*randn(1,128);
>> [sx,as,ps,K] = lmswelchpsd(x,2,0.5,0.5,512);
```
FIGURE 4.27

FIGURE 4.28
%rectangular
%window;
>>om = 0:2*pi/512:2*pi-(2*pi/512);
>>z = [x 0.1*x+0.05*randn(1,128)];
>>[sz,asz,psz,Kz] = lmswelchpsd(z,2,0.5,0.5,512);
%rectangular
%window;
>>subplot(2,1,1);plot(om,as/max(as),'k');
>>subplot(2,1,2);plot(om,asz/max(asz),'k');
>>xlabel('\omega per unit');ylabel('Magnitude');

**PROBLEMS**

**4.1.1** Verify that the sample mean is equal to the ensemble mean.

**4.1.2** If two random processes $x(n)$ and $y(n)$ with zero mean value are uncorrelated, find the autocorrelation of their sum.

**4.1.3** Find the mean and autocorrelation function using the frequency interpretation approach. Use 10 and 1000 realizations.

**4.2.1** Verify the properties 1, 4, 2, and 6 of a WSS process (see Table 4.1).

**4.2.2** If $R_x$ is an autocorrelation matrix, then show that $a^T R_x a \geq 0$ for any vector $a$.

**4.2.3** Assume a discrete sine wave (vector) $s(n) = \sin(0.2\pi n)$ with $n = 0 : 39$, a vector $v(n) = \text{randn}(1,40)$ with statistically independent elements and normally distributed, and the sum of the above is $x(n) = s(n) + v(n)$. (a) Find the biased autocorrelation function up to lag 20, (b) find the FFT of the sine wave, and (c) find the FFT of the autocorrelation function. Plot the functions $s(n)$, $v(n)$, $s(n) + v(n)$, and $r_{xx}(k)$, and their FFTs (their spectra).

**4.2.4** Verify (4.19).

**4.3.1** Verify (4.25).

**4.3.2** Find the joint pdf of a sequence of WGN with $n$ elements, each one having a zero mean value and the same variance.

**4.3.3** If an rv $x$ is $N(m_x, \sigma^2)$ with variance greater than 0, then the rv $w = (x - m_x) / \sigma$ is $N(0,1)$.

**4.3.4** Show that the sample mean is equal to the population mean.

**4.4.1** Verify (4.33).

**4.4.2** Determine the PSD of a zero-mean WSS process $\{x(n)\}$ with $r_{xx}(k) = a^{|k|}, -1 < a < 1$.

**4.5.1** Verify (4.36).

**4.6.1** Verify (4.42).

**4.7.1** Find the periodogram for a white Gaussian (WG) signal with $N = 128$ and $N = 1024$. Verify that the variance (variability) does not decrease with increasing the number of the sequence.
HINTS–SOLUTIONS–SUGGESTIONS

4.1.1

\[ E[\hat{m}] = E\left( \frac{1}{N} \sum_{n=1}^{N} x(n) \right) = \frac{1}{N} \sum_{n=1}^{N} E[x(n)] = \frac{1}{N} \sum_{n=1}^{N} m = \frac{Nm}{N} = m \]

4.1.2

\[ r_{x_l}(k,l) = E\{[x(k) + y(k)][x(l) + y(l)]\} = E\{x(k)x(l)\} + E\{y(k)y(l)\} + E\{x(k)E\{y(l)\} + E\{y(k)x(l)\} \]

\[ = r_{xx}(k,l) + r_{yy}(k,l) \]

4.1.3 The Book MATLAB program is given as follows:

```matlab
function [mx, rx] = lms_mean_autoc_ensemble(M, N)
    % N = number of data whichever distribution is % desired; M = number of realizations;
    x = randn(M,N); % randn = is a MATLAB function producing % zero mean normal distribution; x = MxN
    % matrix; sum(x,1) = MATLAB function that % sums all the rows of the matrix x;
    % sum(x,2) sums all the columns;
    mx = sum(x,1)/M;
    for i = 1:N
        rx(i) = sum(x(:,1).*x(:,i))/M;
    end;
end;
```

```matlab
>> [mx, rx] = lms_mean_autoc_ensemble(10,50);
>> [mx1, rx1] = lms_mean_autoc_ensemble(1000,50);
>> subplot(2,2,1); plot(mx,'k');
>> xlabel('Mean of 10 realizations');
>> subplot(2,2,2); plot(mx1,'k');
>> xlabel('Mean of 1000 realizations');
>> subplot(2,2,3); plot(rx,'k');
>> xlabel('Autoc. of 10 realizations');
>> subplot(2,2,4); plot(rx1,'k');
>> xlabel('Autoc. of 1000 realizations');
```

Figure of P4.1.3 shows the results.

4.2.1

a. \( E\{x(n+l)\} = E\{x(m+l)\} \Rightarrow \text{Mean must be constant} \)

b. \( r_{xx}(l) = E\{x(n+l)x(n)\} = E\{x(n)x(n+l)\} = r_{xx}(n-n-l) = r_{xx}(-l) \)

c. \( r_{xx}(m,n) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x(m)x(n)f(x(m+l)x(n+l))dx(m)dx(n) = r_{xx}(m+l,n+l) \)

\[ = r_{xx}(m+l-n-l) = r_{xx}(m-n) \]
d. \[ E[(x(n + l) - x(n))^2] = E[x^2(n + l) + x^2(n) - 2x(n + l)x(n)] \]
\[ = r_{xx}(0) + r_{xx}(0) - 2r_{xx}(l) \]
\[ = 2r_{xx}(0) - 2r_{xx}(l) \geq 0 \text{ or } r_{xx}(0) \geq r_{xx}(l) \]

\[ \alpha^T R \alpha = a^T E[x x^T] a = E \{ (\alpha^T x)(x^T \alpha) \} = E \{ (\alpha^T x)^2 \} \geq 0, \]

\[ a^T x = x^T a = \text{a number}; \quad \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \alpha = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}, \]

\[ a^T R a = [0.1 \quad -0.2] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} = 0.1^2 + (-0.2)^2 > 0 \]

4.2.3 The Book m-file is given as follows:

```matlab
%prob4_2_3 Book m-file;
n = 0:39; s = sin(0.2*pi*n); v = randn(1,40);
```
\begin{verbatim}
116
 Adaptive Filtering

x = s+v;
 rxx = lms_sample_biased_autoc(x,20);
 fts = fft(s,40);ftrxx = fft(rxx,40);
 subplot(3,2,1);stem(s,'k');xlabel('n');ylabel('s(n)');
 subplot(3,2,2);stem(v,'k');xlabel('n');ylabel('v(n)');
 subplot(3,2,3);stem(x,'k');xlabel('n');ylabel('x(n) = s(n)+v(n)');
 subplot(3,2,4);stem(rxx,'k');xlabel('k');ylabel('\text{r}_{xx}');
 subplot(3,2,5);stem(abs(fts),'k');xlabel('Freq. bins');
 subplot(3,2,6);stem(abs(ftrxx),'k');xlabel('Freq. bins');

Figure of P4.2.3 shows the results.

4.2.4

\[ C_x = E\{ (x - m_x)(x - m_x)^T \} = E\{ (x - m_x)(x^T - m_x^T) \} \]

\[ = E\{ xx^T - m_x x^T - x m_x^T + m_x x^T \} \]

\[ = E\{ xx^T \} - m_x E\{ x^T \} = R_x - m_x m_x^T \]
\end{verbatim}

\text{FIGURE P4.2.3}
4.3.1
\[ r_{xx}(m-n) = \frac{1}{2\pi \sigma_x(m) \sigma_x(n)} \int \int x(m)x(n) \exp \left[ -\frac{x^2(m)}{2\sigma_x^2(m)} - \frac{x^2(n)}{2\sigma_x^2(n)} \right] \]
\[ = \frac{1}{2\pi \sigma_x(m) \sigma_x(n)} \left\{ \int x(m) \exp \left[ -\frac{x^2(m)}{2\sigma_x^2(m)} \right] \int x(n) \exp \left[ -\frac{x^2(n)}{2\sigma_x^2(n)} \right] \right\} \]
\[ = \frac{1}{2\pi \sigma_x(m) \sigma_x(n)} 0 \times 0 = 0 \]

4.3.2 The joint pdf is
\[ f(x(1), x(2), \cdots x(n)) = f_1(x(1))f_2(x(2)) \cdots f_n(x(n)) \]
\[ = \frac{1}{(2\pi)^{n/2} \sigma^2} \exp \left[ -\frac{1}{2\sigma^2} \sum_{k=1}^{n} x^2(k) \right] \]

4.3.3
\[ F(w) = \text{cdf} = \Pr \left\{ \frac{x-m_x}{\sigma} \leq w \right\} = \Pr \{x \leq w\sigma + m_x \} \]
\[ = \int_{-\infty}^{w\sigma + m_x} \frac{1}{\sigma \sqrt{2\pi}} \exp \left[ -\frac{(x-m_x)^2}{2\sigma^2} \right] dx \]

If we change the variable of integration by setting \( y = (x-m_x)/\sigma_x \), then \( F(w) = \int_{-\infty}^{\\infty} (1/\sqrt{2\pi}) \exp[-(y^2/2)]dy \). But \( f(w) = dF(w)/dw \), and hence, \( f(w) \) is equal to the indegrand which is the desired solution.

4.3.4
\[ E[\hat{m}] = E \left\{ \frac{1}{N} \sum_{k=1}^{N} x(k) \right\} = \frac{1}{N} \sum_{k=1}^{N} E[x(k)] = \frac{1}{N} \sum_{k=1}^{N} m = \frac{Nm}{m} = m \]

4.4.1
\[ \int_{-\infty}^{\infty} S_x(e^{j\omega}) e^{j\omega k} d\omega = \sum_{k=-\infty}^{\infty} \int_{-\infty}^{\infty} r_{xx}(k) e^{j\omega(k-m)} d\omega = r_{xx}(m)2\pi \]

4.4.2
\[ S_x(e^{j\omega}) = \sum_{k=-\infty}^{\infty} a^k e^{-j\omega k} = \sum_{k=0}^{\infty} a^k e^{-j\omega k} - 1 + \sum_{k=0}^{\infty} a^k e^{-j\omega k} = \sum_{k=0}^{\infty} a^k e^{-j\omega k} \]
\[ -1 = \frac{1}{1-ae^{j\omega}} + \frac{1}{1-ae^{-j\omega}} - 1 = \frac{1-a^2}{1+a^2-2a\cos \omega} \quad -1 < a < 1 \]
The PSD is a real-valued, even, and nonnegative function of \( \omega \).
4.5.1

\[ y(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m) \quad (1) \]

\[ E \{ x(n+l)y(n) \} \triangleq r_{yx}(l) = \sum_{m=-\infty}^{\infty} h(m)E \{ x(n+l)x(n-m) \} \]

\[ = \sum_{m=-\infty}^{\infty} h(m)r_{xx}(l+m) = \sum_{k=-\infty}^{\infty} h(-k)r_{xx}(l-k) = h(-l) \ast r_{xx}(l) \quad (2) \]

Similarly, we get \( r_{yx}(l) = h(l) \ast r_{xx}(l) \) from (1)

\[ E \{ y(n)y(n-l) \} \triangleq r_{yy}(l) = \sum_{m=-\infty}^{\infty} h(m)E \{ x(n-m)y(n-l) \} \]

\[ = \sum_{m=-\infty}^{\infty} h(m)r_{yx}(l-m) = h(l) \ast r_{yx}(l) \quad (3) \]

We get \( r_{yx}(l) = h(l) \ast h(-l) \ast r_{xx}(l) \) or \( r_{yy}(k) \)

\[ = \sum_{l=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(l)r_{xx}(m-l+k)h(m) \quad \text{from (2) and (3)} \]

4.6.1

\[ H(z) = \frac{Y(z)}{V(z)} = \frac{b(0)}{1 + \sum_{l=0}^{p} a(l)z^{-l}} \Rightarrow y(n) + \sum_{l=1}^{p} a(l)y(n-l) = \sum_{l=0}^{q} b(l)v(n-l) \quad \text{or} \]

\[ E \{ y(n)y(n-k) \} + \sum_{l=1}^{p} a(l)E \{ y(n-l)y(n-k) \} = \sum_{l=0}^{q} b(l)E \{ v(n-l)y(n-k) \} \]

\[ \Rightarrow r_{yy}(k) + \sum_{l=1}^{p} a(l)r_{yy}(k-l) = \sum_{l=0}^{q} b(l)r_{yx}(k-l) \quad (1) \]

\[ y(n) = h(n) \ast v(n) = \sum_{m=-\infty}^{\infty} v(m)h(n-m) \quad (2) \]
From (2) we get

\[
E \{ y(n-l)y(n-k) \} = \sum_{m=-\infty}^{\infty} E \{ y(n-l)y(m) \} h(n-k-m) = \sigma_v^2 \delta(n-l-m)h(n-k-m) = \sigma_v^2 h(l-k)
\]  

(3)

Setting (3) in (1) we get

\[
r_{yy}(k) + \sum_{l=1}^{p} a(l)r_{yy}(k-l) = \sum_{l=0}^{q} b(l)\sigma_v^2 h(l-k)
\]

(4)

\[
H(z) = \frac{Y(z)}{V(z)} = b(0) \frac{1}{1 + \sum_{l=0}^{p} a(l)z^{-l}} \Rightarrow H(z) = b(0)[1 - a(1)z^{-1} - \cdots]
\]

(5)

\[
\Rightarrow h(n) = b(0)[\delta(k) - a(1)\delta(k-1) - \cdots] \Rightarrow h(0) = b(0)\delta(k)k \geq 0
\]

(4) \[ r_{yy}(k) + \sum_{l=1}^{p} a(l)r_{yy}(k-l) = b(0)^2 \sigma_v^2 \delta(k) \quad k \geq 0 \]
This page intentionally left blank
5 The Wiener Filter

5.1 INTRODUCTION

In this chapter, we shall develop two basic important principles used extensively in signal processing, communications, system identification, denoising signals, and so on. The least-squares (LS) principle, proposed first by Gauss when he was 18 years old, is widely applicable to the design of digital signal processing systems. We shall first describe the use of the LS approach to modeling, interference canceling as well as the cases involving prediction. Next, we shall study the celebrated Wiener filter, which was developed during World War II.

5.2 THE LS TECHNIQUE

The principle of LS is used in this chapter for signals of one variable, although the concept applies equally well for signals with more than one variable. Furthermore, we study discrete signals, which can always be derived from continuous signals if the sampling frequency is sufficient high enough so that aliasing does not take place.

Let us, for example, have a discrete signal that is represented in its vector form as follows: \( x = [x(1) \ x(2) \ \cdots \ x(N)] \). We would like, next, to approximate the function with a polynomial of the form:

\[
\hat{x}(nT) = c_1 + c_2 \times (nT) + c_3 \times (nT)^2 + \cdots + c_M \times (nT)^{M-1} \quad T = \text{sampling time} \quad (5.1)
\]

Since the approximating polynomial is a linear function of the unknown \( c_i \)'s, we are dealing with linear LS approximation. The difference between the exact and the approximate function is the error. We can define a cost function \( J \), which is equal to the sum of the difference of errors squared, the total square error. Hence, we write

\[
J = \sum_{n=1}^{N} [x(n) - \hat{x}(n)]^2 \quad (5.2)
\]

Example 5.2.1

Let \( x(n) = 2 \sin(0.05 \pi n) + \text{randn}(1,20) \), with \( T = 1 \), be the signal to be approximated. Let the signal \( \hat{x}(n) = c_1 + c_2 n + c_3 n^2 \), the sampling time \( T = 1 \), approximate the original signal. Find the unknown constants \( c_i \)'s and plot the results.

Solution: The cost function for this case is

\[
J = \sum_{n=1}^{20} [x(n) - \hat{x}(n)]^2 = \sum_{n=1}^{20} [x(n) - c_1 - c_2 n - c_3 n^2]^2
\]

121
The partial derivatives with respect to \( c_i \)'s are

\[
\frac{\partial J}{\partial c_1} = -2 \sum_{n=1}^{20} [x(n) - c_1 - c_2 n - c_3 n^2] = 0 \quad \text{or}
\]
\[
\sum_{n=1}^{20} x(n) - c_1 \sum_{n=1}^{20} 1 - c_2 \sum_{n=1}^{20} n - c_3 \sum_{n=1}^{20} n^2 = 0
\]

\[
\frac{\partial J}{\partial c_2} = -2 \sum_{n=1}^{20} [x(n) - c_1 - c_2 n - c_3 n^2] n = 0 \quad \text{or}
\]
\[
\sum_{n=1}^{20} x(n)n - c_1 \sum_{n=1}^{20} n - c_2 \sum_{n=1}^{20} n^2 - c_3 \sum_{n=1}^{20} n^3 = 0
\]

\[
\frac{\partial J}{\partial c_3} = -2 \sum_{n=1}^{20} [x(n) - c_1 - c_2 n - c_3 n^2] n^2 = 0 \quad \text{or}
\]
\[
\sum_{n=1}^{20} x(n)n^2 - c_1 \sum_{n=1}^{20} n^2 - c_2 \sum_{n=1}^{20} n^3 - c_3 \sum_{n=1}^{20} n^4 = 0
\]

Using MATLAB, we obtain the following system:

\[
\begin{align*}
20c_1 + 210c_2 + 2870c_3 &= 27.7347 \\
210c_1 + 2870c_2 + 44100c_3 &= 296.1023 \\
2870c_1 + 44100c_2 + 722666c_3 &= 3499.9838
\end{align*}
\]

The values of the unknown \( c_i \)'s are found by using the expression:

\[
\begin{bmatrix}
  c_1 \\
  c_2 \\
  c_3
\end{bmatrix} = \begin{bmatrix}
  20 & 210 & 2370 \\
  210 & 2870 & 44100 \\
  2870 & 44100 & 722666
\end{bmatrix}^{-1} \begin{bmatrix}
  27.7347 \\
  296.1023 \\
  3499.9838
\end{bmatrix} = \begin{bmatrix}
  -1.2457 \\
  0.7042 \\
  -0.0332
\end{bmatrix}
\]

Therefore, the estimate curve is given by

\[
\hat{x}(n) = -1.2457 + 0.7042n - 0.0332n^2
\]

Figure 5.1 shows the original signal \( \{x(n)\} \), the approximate one \( \{\hat{x}(n)\} \), based on the LS approach, and the undisturbed signal \( \{2\sin(0.05\pi n)\} \).

\[\Box\]

### 5.2.1 Linear LS

We proceed to generalize the linear LS technique by a set of known vectors \( \{h(n)\}, n = 0, 1, \ldots, M \) with \( M < N \). \( H \) is a matrix having inverse. Therefore, we write
The Wiener Filter

Next, to verify (5.3), we proceed as follows:

\[
J(c) = \sum_{n=0}^{N-1} [x(n) - \hat{x}(n)]^2 = \sum_{n=0}^{N-1} [x(n) - c^T h(n)]^2
\]

\[
= \sum_{n=0}^{N-1} [x(n) - [c_1 h_1(n) + \cdots + c_M h_M(n)]]^2 = (x - Hc)^T (x - Hc)
\]

\[\text{(5.3)}\]

Next, to verify (5.3), we proceed as follows:

\[
J(c) = \begin{bmatrix} x(0) \\ x(1) \end{bmatrix} - \begin{bmatrix} h_1(0) & h_2(0) \\ h_1(1) & h_2(1) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} h_1(0) & h_2(0) \\ h_1(1) & h_2(1) \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \end{bmatrix} - \begin{bmatrix} h_1(0) & h_2(0) \\ h_1(1) & h_2(1) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}
\]

\[
= [x(0) - c_1 h_1(0) - c_2 h_2(0) \quad x(1) - c_1 h_1(1) - c_2 h_2(1)] \begin{bmatrix} x(0) - h_1(0)c_1 - h_2(0)c_2 \\ x(1) - h_1(1)c_1 - h_2(1)c_2 \end{bmatrix}
\]
The last two expressions are identical. The reader should observe that the following matrix relationship was used:

\[(x - Hc)^T = (x^T - c^T H^T)\]  

(5.4)

where:

The exponent \( T \) stands for transpose of a matrix.

To find the unknowns \( c_i \), differentiate the cost function of (5.3) with respect to each \( c_i \) and then set the developed equations equal to zero. Therefore, we have a system with \( c_i \), the unknowns that are determined by solving the system. The following example will elucidate the procedure.

**Example 5.2.2**

Let a signal be a constant, \( s(n) = A \), and let the received signal be given by \( x(n) = 5 + \text{randn}(1, 10) \). Find \( A \).

**Solution:** According to the LS approach, we can estimate the constant 5 by minimizing the cost function. Taking the derivative \( J \) with respect to \( A \) and setting the results equal to zero, we obtain

$$J(A) = \sum_{n=0}^{N-1} [A - x(n)]^2; \quad \frac{\partial J(A)}{\partial A} = \sum_{n=0}^{N-1} [2A - 2x(n)] = 0 \quad \text{or} \quad A \sum_{n=0}^{N-1} 1 = \sum_{n=0}^{N-1} x(n)$$

Using MATLAB, we obtain \( \hat{A} = \text{sum}(x)/10 = 5.1794 \). The minimum \( J(A) \) is given by

$$J_{\text{min}} = \sum_{n=1}^{10} [x(n) - 5.1794]^2 = 1.3803$$

and for this case

$$J_{\text{min}} = \sum_{n=1}^{10} [x(n) - 5.1794]^2 = 1.3803$$

◼

**Example 5.2.3**

Find the amplitude constants of the signal

\[s(n) = A \sin(0.1\pi n) + B \sin(0.4\pi n) \quad n = 0, 1, \ldots, N - 1\]

if the received signal is \( x(n) = s(n) + \text{randn}(1, N) \) and their exact values are \( A = 0.2 \) and \( B = 5.2 \).
Solution: For this case, and for \( n = 0, 1, 2, 3 \) and \( N = 4 \), we obtain

\[
J(A,B) = \sum_{n=0}^{3} (x(n) - h_1(n)A - h_2(n)B) \]

\[
h_1(n) = \sin(0.1\pi n) \quad h_2(n) = \sin(0.4\pi n)
\]

If we differentiate \( J(A,B) \) first with respect to \( A \) and next with respect to \( B \), and then equate the results to zero, we obtain

\[
\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix}
\sum_{n=0}^{3} h_1^2(n) & \sum_{n=0}^{3} h_1(n)h_2(n) \\
\sum_{n=0}^{3} h_1(n)h_2(n) & \sum_{n=0}^{3} h_2^2(n)
\end{bmatrix}^{-1}
\begin{bmatrix}
\sum_{n=0}^{3} x(n)h_1(n) \\
\sum_{n=0}^{3} x(n)h_2(n)
\end{bmatrix}
\]

The following MATLAB program was used to find \( \hat{A} = 0.5372, \hat{B} = 4.7868 \) from the vector \( ab \):

\[
\begin{align*}
>> n = 0:3; x &= 0.2\sin(0.1\pi n) + 5.2\sin(0.4\pi n) \\
>> h1 &= \sin(0.1\pi n); h2 &= \sin(0.4\pi n); \\
>> ab &= \text{inv}([\text{sum}(h1.^2) \text{sum}(h1.*h2); \text{sum}(h1.*h2); \text{sum}(h2.^2)]) * [\text{sum}(x.*h1); \text{sum}(x.*h2)];
\end{align*}
\]

\[
\begin{figure}
\end{figure}
\]

5.2.2 LS Formulation

We consider a linear adaptive filter with coefficients at time \( n \):

\[
w(n) = [w_1(n) \quad w_2(n) \quad \ldots \quad w_M(n)]^T \quad \text{a measured real-valued input vector}
\]

\[
x(n) = [x_1(n) \quad x_2(n) \quad \ldots \quad x_M(n)]^T \quad \text{and a measured desired response } d(n)
\]

Note that no structure has been specified for the input vector \( x(n) \), and therefore, it can be considered as the successive samples of a particular process or as a snapshot of \( M \) detectors as shown in Figure 5.2. Hence, the problem is to estimate the desired response \( d(n) \) using the linear combination:

\[
y(n) = w^T(n)x(n) = \sum_{k=1}^{M} w_k(n)x_k(n) \quad n = 1, 2, \ldots, N
\]

The above equation can be represented by a linear combiner as shown in Figure 5.3. The combiner error is defined by the relation:

\[
e(n) = d(n) - y(n) = d(n) - w^T(n)x(n)
\]
The coefficients of the adaptive filter are found by minimizing the sum of the squares of the error (LS):

\[ J = \sum_{n=1}^{N} g(n)e^2(n) \]  

(5.7)

where:

- \( g(n) \) is a weighting function

Therefore, in the LS method, the filter coefficients are optimized by using all the observations from the time the filter begins until the present time and minimizing
the sum of the squared values of the error samples, which are equal to the measured desired signal and the output signal of the filter. The minimization is valid when the filter coefficient vector $w(n)$ is kept constant, $w$, over the measurement time interval $1 \leq n \leq N$. In statistics, the LS estimation is known as **regression**, $e(n)$ are known as **signals**, and $w$ is the **regression vector**.

We next define the matrix of the observed input samples as follows:

$$X^T = \begin{bmatrix} x_1(1) & x_1(2) & \cdots & x_1(N) \\ x_2(1) & x_2(2) & \cdots & x_2(N) \\ \vdots & \vdots & \ddots & \vdots \\ x_M(1) & x_M(2) & \cdots & x_M(N) \end{bmatrix} \rightarrow \text{Data records (} M \times N \text{)}$$  \hfill (5.8)

where we assume that $N > M$. This defines an overdetermined LS problem.

For the case in which we have one-dimensional input signal, as shown in Figure 5.2b, the data matrix takes the form:

$$X^T = \begin{bmatrix} x(M) & x(M+1) & \cdots & x(N) \\ x(M-1) & x(M) & \cdots & x(N-1) \\ \vdots & \vdots & \ddots & \vdots \\ x(1) & x(2) & \cdots & x(N-M+1) \end{bmatrix}$$  \hfill (5.9)

The output $y$, the error $e$, and the data $x_k$ are as follows:

$$y = Xw$$  \hfill (5.10)

$$e = d - y$$  \hfill (5.11)

where:

$$y = [y(1) \quad y(2) \quad \cdots \quad y(N)]^T \equiv \text{Filter output vector } (N \times 1)$$  \hfill (5.12)

$$d = [d(1) \quad d(2) \quad \cdots \quad d(N)]^T$$  \hfill (5.13)

$$e = [e(1) \quad e(2) \quad \cdots \quad e(N)]^T \equiv \text{Error vector } (N \times 1)$$  \hfill (5.14)

$$x = [x_1(n) \quad x_2(n) \quad \cdots \quad x_N(n)]^T \equiv \text{Snapshot } (N \times 1)$$  \hfill (5.15)

$$x_k = [x_k(1) \quad x_k(2) \quad \cdots \quad x_k(N)]^T \equiv \text{Data vector, } k = 1, 2, \ldots, M$$  \hfill (5.16)

$$w = [w_1 \quad w_2 \quad \cdots \quad w_M]^T \equiv \text{Filter coefficients } (M \times 1)$$  \hfill (5.17)

$$X = [x_1 \quad x_2 \quad \cdots \quad x_M]^T \equiv \text{Matrix } (N \times M)$$  \hfill (5.18)
In addition, with \( g(n) = 1 \) for all \( n \), (5.7) takes the form:

\[
J = e^T e = (d - y)^T (d - y) = (d - Xw)^T (d - Xw)
\]

\[
= d^T d - w^T X^T d - d^T Xw + w^T X^T Xw \tag{5.19}
\]

\[
= E_d - w^T p - p^T w + w^T R w = E_d - 2p^T w + w^T R w
\]

where:

\[
E_d = d^T d = \sum_{n=1}^{N} d(n)d(n) \tag{5.20}
\]

\[
R = X^T X = \sum_{n=1}^{N} x(n)x^T(n) \quad (M \times M) \tag{5.21}
\]

\[
p = X^T d = \sum_{n=1}^{N} x(n)d(n) \quad (M \times 1) \tag{5.22}
\]

\[
y = Xw = \sum_{k=1}^{M} w_k x_k \quad (N \times 1) \tag{5.23}
\]

The matrix \( R \) becomes time average if it is divided by \( N \). In statistics, the scaled form of \( R \) is known as the sample correlation matrix.

Setting the gradient of \( J \) with respect to the vector coefficients \( w \) equal to zero, we obtain

\[
R\hat{w} = p; \quad p^T = \hat{w}^T R^T = \hat{w}^T R \quad (R \text{ is symmetric}) \tag{5.24}
\]

or

\[
\hat{w} = R^{-1} p \tag{5.25}
\]

Therefore, the minimum sum of squared errors is given by

\[
J_{\min} = d^T d - 2p^T R^{-1} p + w^T R R^{-1} p = E_d - p^T R^{-1} p = E_d - p^T \hat{w} \tag{5.26}
\]

since \( R \) is symmetric.

**Example 5.2.4**

Let the desired response be \( d = [1 1 1 1] \), and the two measured \( x_1 = [0.7 \ 1.4 \ 0.4 \ 1.3]^T \) and \( x_2 = [1.2 \ 0.6 \ 0.5 \ 1.1]^T \). Then, we obtain
The LS technique is a mathematical procedure that enables us to achieve a best fit of a model to experimental data. In the sense of the $M$-parameter linear system, shown in Figure 5.4, (5.5) is written in the form:

$$y(n) = w_1 x_1(n) + w_2 x_2(n) + \cdots + w_M x_M(n) \quad n = 1, 2, \ldots, N$$

The above equation takes the following matrix form:

$$y = Xw$$

(5.28)

To estimate the $M$ parameters $w_i$, it is necessary that $N \geq M$. If $N = M$, then we can uniquely solve for $w$ to find

$$\hat{w} = X^{-1} y$$

(5.29)

provided that $X^{-1}$ exists. $\hat{w}$ is the estimate of $w$. Using the least error squares, we can determine $w$, provided that $N > M$.

Let us define an error vector $e = [e_1, e_1, \cdots, e_N]^T$ as follows:

$$e = y - Xw$$

(5.30)

Next, we choose $\hat{w}$ in such a way that the criterion

$$J = \sum_{i=1}^{N} e_i^2 = e^T e$$

(5.31)
Adaptive Filtering

is minimized. To proceed, we write

\[ J = (y - Xw)^T (y - Xw) \]
\[ = y^T y - w^T X^T y - yXw + w^T X^T Xw \]  \hspace{1cm} (5.32)

Differentiating \( J \) with respect to \( w \) and equating the results to zero, we determine the conditions on the estimate \( \hat{w} \) that minimizes \( J \). Hence,

\[ \frac{\partial J}{\partial w} \bigg|_{w=\hat{w}} = -2X^T y + 2X^T X \hat{w} = 0 \]
\[ X^T X \hat{w} = X^T y \] \hspace{1cm} (5.34)

from which we obtain

\[ \hat{w} = (X^T X)^{-1} X^T y \] \hspace{1cm} (5.35)

The above equation is known as the \textbf{LS estimator} (LSE) of \( w \). Equation 5.34 is known as the \textbf{normal equation}.

\[ \frac{\partial J}{\partial w} \bigg|_{w=\hat{w}} \]

5.2.3 \textbf{Statistical Properties of LSEs}

We rewrite (5.30) in the form \((X = \text{deterministic matrix}):\)

\[ y = Xw + e \] \hspace{1cm} (5.36)

and assume that \( e \) is a stationary random vector with zero mean value, \( E[e] = 0 \). Furthermore, \( e \) is assumed to be uncorrelated with \( y \) and \( X \). Therefore, on the given statistical properties of \( e \), we wish to know just how good or how accurate the estimates of the parameters are.

Substituting (5.36) in (5.35) and taking the ensemble average, we obtain

\[ E[\hat{w}] = E[w + (X^T X)^{-1} X^T e] = E[w] + E[(X^T X)^{-1} X] E[e] = w \ (E[e] = 0) \] \hspace{1cm} (5.37)

which indicates that \( \hat{w} \) is \textbf{unbiased}.

The covariance matrix corresponding to the estimate error \( \hat{w} - w \) is

\[ C_w = E[(\hat{w} - w)(\hat{w} - w)^T] \]
\[ = E[(X^T X)^{-1} X^T (Xw + e) - w](\hat{w} - w)^T] \]
\[ = E[(X^T X)^{-1} (X^T X)w + (X^T X)^{-1} e - w][\hat{w} - w]^T] \]
\[ = E[(X^T X)^{-1} X^T e][(X^T X)^{-1} X^T e]^T] \]
\[ = (X^T X)^{-1} X^T E[e e^T] X (X^T X)^{-1} \]
\[ = (X^T X)^{-1} X^T R_x X (X^T X)^{-1} \] \hspace{1cm} (5.39)
where:

\[ R_e \] is the error correlation matrix

If the noise sample \( e(i) \) for \( i = 1, 2, 3, \ldots \) is normal, identically distributed with zero mean and variance \( \sigma^2 [e = N(0, \sigma I)] \), then

\[
R_e = E\{ee^T\} = \sigma^2 I
\]

and hence,

\[
C_w = \sigma^2 (X^TX)^{-1}
\]

Using (5.36) and taking into consideration that \( e \) is a Gaussian random vector, then the natural logarithm of its probability density is given by

\[
\ln \frac{1}{(2\pi \sigma^2)^{N/2}} \mid C_e \mid \exp \left[ -\frac{1}{2} (y - Xw)^T C_e^{-1} (y - Xw) \right]
\]

(5.42)

\[
= -\ln(2\pi \sigma^2)^{N/2} - \frac{1}{2\sigma^2} (y - Xw)^T (y - Xw)
\]

since \( C_e = \sigma^2 I \) and \( |C_e| \) implies the determinant of \( C_e \). Next, we differentiate (5.42) with respect to parameter \( w \). Hence, we find

\[
\frac{\partial \ln(e;w)}{\partial w} = -\frac{1}{2\sigma^2} \frac{\partial}{\partial w} (y^T y - 2y^TXw + w^TX^TXw)
\]

(5.43)

since \( y^TXw = w^TX^Ty \) is scalar. Using the identity below (see Appendix 2)

\[
\frac{\partial b^T w}{\partial w} = b, \quad \frac{\partial w^T A w}{\partial w} = 2A w \quad (A \text{ is symmetric})
\]

(5.44)

Equation 5.43 becomes

\[
\frac{\partial \ln(e;w)}{\partial w} = \frac{1}{\sigma^2} (X^Ty - X^TXw)
\]

(5.45)

Assuming that \( X^TX \) is invertible, then

\[
\hat{w} = (X^TX)^{-1}X^Ty = g(w)
\]

(5.47)

and (5.46) becomes

\[
\frac{\partial \ln(e;w)}{\partial w} = \frac{X^TX}{\sigma^2} (\hat{w} - w)
\]

(5.48)
The matrix

\[ I(w) = \frac{X^T X}{\sigma^2} \]  

(5.49)
is known as the **Fisher information matrix**. In the CRLB theorem, the Fisher matrix is defined by the relation:

\[ (Iw)_{ij} = E \left[ \frac{\partial^2 \ln P(e;w)}{\partial w_i \partial w_j} \right] \]

(5.50)

and thus, the parameters are shown explicitly. Comparing (5.41) and (5.49), the MVU estimator of \( w \) is given by (5.47) and its covariance matrix is

\[ C_w = I^{-1}(w) = \sigma^2 (X^T X)^{-1} \]

(5.51)
The MVU estimator of the linear model (5.36) is efficient since it attains the CRLB or, in other words, the covariance matrix is equal to the inverse of the Fisher information matrix.

Let us rewrite the error covariance matrix in the form:

\[ C_w = I^{-1}(w) = \sigma^2 (X^T X)^{-1} = \frac{\sigma^2}{N} \left( \frac{1}{N} X^T X \right)^{-1} \]

(5.52)

where:

- \( N \) is the number of equations in the vector equation (5.36). Let \( \lim_{N \to \infty} [(1/N)X^T X]^{-1} = A \), where \( A \) is a rectangular constant matrix. Then

\[ \lim_{N \to \infty} A = \sigma^2 N^{-1} \]

(5.53)

Since the covariance becomes zero as \( N \) goes to infinity it implies that \( \hat{w} = w \). The above convergence property defines \( \hat{w} \) as a **consistent estimator**.

The above development shows that, if a system is modeled as linear in the presence of white Gaussian noise, the LSE approach provides estimators that are unbiased and consistent.

### 5.2.4 The LS Approach

Using the LS approach, we try to minimize the squared difference between the given data (or desired data) \( d(n) \) and the output signal of a linear time-invariant (LTI) system. The signal \( y(n) \) is generated by some system, which in turn depends on its unknown parameters \( w_i \)'s. The LSE of \( w_i \)'s chooses the values that make \( y \)'s closest to the given data. The measure of closeness is defined by the LSE [see also (5.19)]. For the one-coefficient system model, we have
The Wiener Filter

\[ J(w) = \sum_{n=1}^{N} [d(n) - y(n)]^2 \]  \hspace{1cm} (5.54)

and the dependence of \( J \) on \( w \) is via \( y(n) \). The value of \( w \) that minimizes the cost function \( J(w) \) is the LSE. It is apparent that the performance of LSE will depend on the statistical properties of the corrupting noise to the signal as well as any system modeling error.

**Example 5.2.5**

Let us assume that the signal is \( y(n) = a \cos(\omega_0 n) \), where \( \omega_0 \) is known and the amplitude \( a \) must be determined. Hence, the LSE minimizes the cost function:

\[ J(a) = \sum_{n=1}^{N} [d(n) - a \cos(\omega_0 n)]^2 \]  \hspace{1cm} (5.55)

Therefore, we obtain

\[ \frac{\partial J(a)}{\partial a} = \sum_{n=1}^{N} (-2) \cos(\omega_0 n)(d(n) - a \cos(\omega_0 n)) = 0 \]

\[ \hat{a} = \frac{\sum_{n=1}^{N} d(n) \cos(\omega_0 n)}{\sum_{n=1}^{N} \cos(\omega_0 n)} \]

\[ \Box \]

Let us assume that the output of a system is linear, and it is given by the relation \( y(n) = x(n)w \), where \( x(n) \) is a known sequence. Hence, the LSE criterion becomes

\[ J(w) = \sum_{n=1}^{N} [d(n) - x(n)w]^2 \]

The estimate value of \( w \) is

\[ \hat{w} = \frac{\sum_{n=1}^{N} d(n)x(n)}{\sum_{n=1}^{N} x^2(n)} \]

and the minimum LS error is given by (see Problem 5.2.4)

\[ J_{\text{min}} = J(\hat{w}) = \sum_{n=1}^{N} d^2(n) - \hat{w} \sum_{n=1}^{N} d(n)x(n) = \sum_{n=1}^{N} d^2(n) - \left[ \frac{\sum_{n=1}^{N} d(n)x(n)}{\sum_{n=1}^{N} x^2(n)} \right]^2 \]  \hspace{1cm} (5.56)
Example 5.2.6

Consider the experimental data shown in Figure 5.5. It is recommended that the linear model, \( y(n) = a + bn \), for the data be used. Using the LSE approach, we find the cost function:

\[
J(\mathbf{w}) = \sum_{n=1}^{N} [d(n) - a - bn]^2 = (d - X\mathbf{w})^T (d - X\mathbf{w})
\]

(5.57)

where:

\[
\mathbf{w} = \begin{bmatrix} a \\ b \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ \vdots \\ 1 & N \end{bmatrix}
\]

(5.58)

From (5.35), the estimate of \( \mathbf{w} \) is

\[
\hat{\mathbf{w}} = (X^T X)^{-1} X^T d
\]

(5.59)

and from the data shown in Figure 5.5

\[
\hat{\mathbf{w}} = \begin{bmatrix} \hat{a} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} 1.6147 \\ 0.0337 \end{bmatrix}
\]

FIGURE 5.5
The straight line was also plotted to verify the procedure of LSE. The data were produced using the equation \( d(n) = 1.5 + 0.035n + \text{randn}(n) \) for \( n = 1, 2, \ldots, 100 \).

### 5.2.5 Orthogonality Principle

To obtain the orthogonality principle for the LS problem, we follow the procedure developed for the Wiener filter. Therefore, using unweighted sum of the squares of the error, we obtain

\[
\frac{\partial J}{\partial w_k} = \frac{\partial}{\partial w_k} \left[ \sum_{m=1}^{N} e(m)e(m) \right] = 2 \sum_{m=1}^{N} e(m) \frac{\partial e(m)}{\partial w_k} \quad k = 1, 2, \ldots, M \quad (5.60)
\]

But (5.10) and (5.11) are equal to \((w)\) has \(M\) coefficients

\[
e(m) = d(m) - \sum_{k=1}^{M} w_k x_k(m) \quad (5.61)
\]

and, therefore, taking the derivative of \(e(m)\) with respect of \(w_k\) and introducing the results of (5.60), we obtain

\[
\frac{\partial J}{\partial w_k} = -2 \sum_{m=1}^{N} e(m)x_k(m) \quad (5.62)
\]

We note that when \(w = \hat{w}\) (the optimum value), we have the relationship \((\partial J/\partial w_k) = 0\) for \(k = 1, 2, \ldots, M\), and hence, (5.62) becomes

\[
\sum_{m=1}^{N} \hat{e}(m)x_k(m) = \hat{e}^T x_k = 0 \quad k = 1, 2, \ldots, M \quad (5.63)
\]

where:

\[
\hat{e} = [\hat{e}(1) \quad \hat{e}(2) \quad \hat{e}(3) \quad \ldots \quad \hat{e}(N)]^T = d - \hat{y} \quad (5.64)
\]

\[
x_k = [x_k(1) \quad x_k(2) \quad \ldots \quad x_k(N)]^T \quad k = 1, 2, \ldots, M \quad (5.65)
\]

The estimate error \(\hat{e}(m)\) is optimum in the LS sense. The above result is known as the **principle of orthogonality**.

### 5.2.6 Corollary

Equation 5.10 may be written as the sum of the columns of \(X\) as follows:

\[
\hat{y} = \sum_{k=1}^{M} x_k(n)\hat{w}_k \quad n = 1, 2, \ldots, N \quad (5.66)
\]
Multiplying (5.66) by $\hat{e}^T$ and taking into consideration (5.63), we obtain

$$\hat{e}^T \hat{y} = 0$$  \hspace{1cm} (5.67)

The above corollary indicates that when the coefficients of the filter are optimum in the LS sense, then the output of the filter and the error are orthogonal.

**Example 5.2.7**

Using the results of Example 5.2.4, we find

$$\hat{e} = d - \hat{y} = \begin{bmatrix} 1.05953819523825 \\ 0.91864528560697 \\ 0.4815639439564 \\ 1.21506254286554 \end{bmatrix}$$

$\hat{e}^T x_0 = 2.505 \times 10^{-15}$, $\hat{e}^T x_1 = 1.457 \times 10^{-15}$

5.2.7 **Projection Operator**

Projection operator is another form of interpretation to the solution of the LS problem. Let us, for clarity, assume that we have two vectors ($N$ vectors in the $N$-dimensional case) $x_k$ that form two-dimensional subspace (see Figure 5.6). The vectors $x_1$ and $x_2$ constitute the column space of the data matrix $X$.  

**FIGURE 5.6**
We note the following properties:

1. The vector $\hat{d}$ is obtained as a linear combination of the data column space $x_1, x_2, \ldots, x_M$ of $X$ that constitutes the subspace of $M$. 

2. From all the vectors in the subspace spanned by $x_1, x_2, \ldots, x_M$, the vector $\hat{d}$ has the minimum Euclidian distance from $d$.

3. The distance $\hat{e} = d - \hat{d}$ is a vector that is orthogonal to the subspace.

We also note that $\hat{y}$ satisfies the above three properties. From (5.66), we observe that $\hat{y}$ is a linear combination of the data column space, which spans the subspace. Next, minimizing $e^T \hat{e}$, where $\hat{e} = d - \hat{d}$, is equivalent to minimizing the Euclidian distance between $d$ and $\hat{y}$. The third property is satisfied by (5.67). Therefore, we can conclude that $\hat{y}$ is the projection of $d$ onto the subspace spanned by the vectors $x_1, x_2, \ldots, x_M$.

Equation 5.66 can also be written in the matrix form:

$$\hat{y} = X\hat{w} = X(X^TX)^{-1}X^Td$$  \hspace{1cm} (5.68)

In the above equation, we set $\hat{w} = R^{-1}p$ [see (5.25)], $R^{-1} = (X^TX)^{-1}$ [see (5.21)], and $p = X^Td$ [see (5.22)]. Since the matrix $P = X(X^TX)^{-1}X^T$ projects the desired vector in the $N$-dimensional space to $\hat{y}$ in the $M$-dimensional subspace ($N > M$), it is known as the projection matrix or projection operator. The name is due to the fact that the matrix $P$ projects the data vector $d$ onto the column space of $X$ to provide the LS estimate $\hat{y}$ of $d$.

The LS error can be expressed as

$$\hat{e} = d - \hat{y} = d - Pd = (I - P)d$$ \hspace{1cm} (5.70)

where:

$I$ is an $N \times N$ identity matrix

The projection matrix is equal to its transpose (Hermitian for complex matrices) and independent, that is,

$$P = P^T, \quad P^2 = P^TP = P$$ \hspace{1cm} (5.71)

The matrix $I - P$ is known as the orthogonal complement projection operator. The filter coefficients are given by

$$\hat{w} = R^{-1}p = (X^TX)^{-1}X^Td$$ \hspace{1cm} (5.72)

$$X^+ = (X^TX)^{-1}X^T$$ \hspace{1cm} (5.73)
is an $M \times N$ matrix known as the pseudoinverse or Moore–Penrose generalized inverse of matrix $X$ (see Appendix 2).

**Example 5.2.8**

Using the data given in Example 5.2.4, we obtain

$$P = PX(X^TX)^{-1}X^T = \begin{bmatrix} 0.7278 & -0.2156 & 0.2434 & 0.3038 \\ -0.2156 & 0.7762 & 0.0013 & 0.3566 \\ 0.2434 & 0.0013 & 0.0890 & 0.1477 \\ 0.3038 & 0.3566 & 0.1477 & 0.4068 \end{bmatrix}$$

$$\hat{y} = Pd = [1.0595 \ 0.9186 \ 0.4815 \ 1.2150]^T$$

$$\hat{e} = (I - P)d = [0.0595 \ 0.0813 \ 0.5184 \ -0.2150]$$

5.2.8 **LS Finite Impulse Response Filter**

The error of the filter is given by

$$e(n) = d(n) - \sum_{k=1}^{M} w_k x(n-k) = d(n) - w^T x(n) \quad (5.74)$$

where:

- $d(n)$ is the desired signal
- $x(n) = [x(n) \ x(n-1) \ \cdots \ x(n-M+1)]^T \quad (5.75)$
- $w = [w_1 \ w_2 \ \cdots \ w_M]^T \quad (5.76)$

is the input to the filter, and

is the filter coefficient vector.

It turns out that the exact form of $e$, $d$, and $X$ depends on the range $N_i \leq n \leq N_f$ of the data to be used. Therefore, the range of the squared-error summation then becomes

$$J \triangleq E = \sum_{n=N_i}^{n=N_f} e^2(n) = e^T e \quad (5.77)$$
The Wiener Filter

The LS finite impulse response (FIR) filter is found by solving the LS normal equations [(5.24) and (5.34)]:

$$(X^T X) \hat{w} = X^T d = p \quad \text{(or } R \hat{w} = p)$$

with the minimum LS error

$$J_{\text{min}} \triangleq E_{\text{min}} = E_d - p^T \hat{w}$$

where:

$E_d = d^T d$ is the energy of the desired signal

The elements of the time-averaged correlation matrix $R$ are given by (the real averaged correlation coefficients must be divided by $N_f - N_i$):

$$r_{ij} = x_i^T x_j = \sum_{n=N_i}^{N_f} x(n+1-i)x(n+1-j) \quad 1 \leq i, j \leq M$$

There are two important different ways to select the summation range $N_i \leq n \leq N_f$ which are exploited in Problem 9.6.1: the no-window case, where $N_i = M - 1$ and $N_f = N - 1$, and the full-window case, where the range of the summation is from $N_i = 0$ to $N_f = N + M - 2$. The no-window method is also known as the autocorrelation method and the full-window method is also known as the covariance method.

The covariance method, data matrix $D$, is written as follows:

$$D^T = \begin{bmatrix} x(M) & x(M+1) & \cdots & x(N) \\ x(M-1) & x(M) & \cdots & x(N-1) \\ \vdots & \vdots & \ddots & \vdots \\ x(1) & x(2) & \cdots & x(N-M+1) \end{bmatrix}$$

Then the $M \times M$ time-averaged correlation matrix is given by

$$R = \sum_{n=M}^{N} x(n)x^T(n) = D^T D$$

**Book m-Function for Covariance Data Matrix**

```matlab
function [dT] = lms_data_matrix_covar_meth(x,M)
%function [dT] = lms_data_matrix_covar_meth(x,M)
%M = number of filter coefficients;x = data vector;
%dT = transposed data matrix;
for m = 1:M
    for n = 1:length(x)-M+1
        dT(m,n) = x(M-m+n);
    end;
end;
```

The elements of the time-averaged correlation matrix $R$ are given by (the real averaged correlation coefficients must be divided by $N_f - N_i$):

$$r_{ij} = x_i^T x_j = \sum_{n=N_i}^{N_f} x(n+1-i)x(n+1-j) \quad 1 \leq i, j \leq M$$

There are two important different ways to select the summation range $N_i \leq n \leq N_f$ which are exploited in Problem 9.6.1: the no-window case, where $N_i = M - 1$ and $N_f = N - 1$, and the full-window case, where the range of the summation is from $N_i = 0$ to $N_f = N + M - 2$. The no-window method is also known as the autocorrelation method and the full-window method is also known as the covariance method.

The covariance method, data matrix $D$, is written as follows:

$$D^T = \begin{bmatrix} x(M) & x(M+1) & \cdots & x(N) \\ x(M-1) & x(M) & \cdots & x(N-1) \\ \vdots & \vdots & \ddots & \vdots \\ x(1) & x(2) & \cdots & x(N-M+1) \end{bmatrix}$$

Then the $M \times M$ time-averaged correlation matrix is given by

$$R = \sum_{n=M}^{N} x(n)x^T(n) = D^T D$$

**Book m-Function for Covariance Data Matrix**

```matlab
function [dT] = lms_data_matrix_covar_meth(x,M)
%function [dT] = lms_data_matrix_covar_meth(x,M)
%M = number of filter coefficients;x = data vector;
%dT = transposed data matrix;
for m = 1:M
    for n = 1:length(x)-M+1
        dT(m,n) = x(M-m+n);
    end;
end;
```
Example 5.2.9

If the data vector is \( x = [0.7 \quad 1.4 \quad 0.4 \quad 1.3 \quad 0.1]^T \), the data filter has three coefficients \((M = 3)\). Using the above Book m-function, we obtain

\[
D^T = \begin{bmatrix}
0.4 & 1.3 & 0.1 \\
1.4 & 0.4 & 1.3 \\
0.7 & 1.4 & 0.4
\end{bmatrix} \quad (D^T \text{ is Toeplitz matrix})
\]

\[
R = D^T D = \begin{bmatrix}
1.8600 & 1.2100 & 2.1400 \\
1.2100 & 3.8100 & 2.0600 \\
2.1400 & 2.0600 & 2.6100
\end{bmatrix}
\]

\[\square\]

The data matrix in (5.81) has the following properties:

**Property 1:** The correlation matrix \( R \) is equal to its transpose (for complex quantities \( R \) is Hermitian \( R = R^T \)). The proof is directly found from (5.82).

**Property 2:** The correlation matrix is nonnegative definite, \( a^T R a \geq 0 \) for any \( M \times 1 \) vector \( a \).

**Property 3:** The eigenvalues of the correlation matrix \( R \) are all real and non-negative (see Chapter 6).

**Property 4:** The correlation matrix is the product of two rectangular Toeplitz matrices that are the transpose of each other (see Example 5.2.9).

The following Book m-function produces the no-window FIR filter.

**Book m-Function No-Window LS Method**

```matlab
function [R,w,Jmin] = lms_no_window_least_square_fir(x,M,d)
    %x = data of length N;M = number of filter coefficients;
    %d = desired signal = [d(M) d(M+1) ... d(N)]';
    N = length(x);
    for i = 1:M
        for j = 1:N-M+1
            D(i,j) = x(M-i+j);
        end;
    end;
    Dt = D';
    R = D*Dt;
    p = D*d(1,1:N-M+1)';
    w = inv(R)*p;
    Jmin = d'*d-p'*w;
end;
```

5.3 THE MEAN-SQUARE ERROR

In this chapter, we develop a class of linear optimum discrete-time filters known as the **Wiener filters**. These filters are optimum in the sense of minimizing an appropriate function of the error known as the **cost function**. The cost function that is
commonly used in filter design optimization is the **mean-square error** (MSE). Minimizing the MSE involves only second-order statistics (correlations) and leads to a theory of linear filtering that is useful in many practical applications. This approach is common to all optimum filter designs. Figure 5.7 shows the block diagram presentation of the optimum filter problem. The MSE approach is mathematically tractable and the performance function has a global minimum.

Based on the MSE approach, we try to establish a linear-invariant filter, \( W(z) \), such that a signal \( \{ x(n) \} \) produces the minimum mean-square estimate, \( \hat{d}(n) \), with a desired one, \( d(n) \). The Wiener filter can consider the following problems:

1. **Filtering**: Given a noisy signal \( x(n) = d(n) + v(n) \), the filter will approximate the desired one, \( d(n) \), based on the present and past values of \( x(n) \).
2. **Smoothing**: The Wiener filter can be designed to estimate \( d(n) \) from \( x(n) \) using all the available data.
3. **Prediction**: If \( d(n) = x(n+1) \) and \( W(z) \) are causal filters, then the Wiener filter becomes a linear predictor.
4. **Deconvolution**: When \( x(n) = d(n) * g(n) + v(n) \), with \( g(n) \) being the unit sample response of a linear shift-invariant filter, the Wiener filter becomes a deconvolution filter.

The basic idea is to recover a desired signal \( d(n) \), given a noisy observation \( x(n) = d(n) + v(n) \), where both \( d(n) \) and \( v(n) \) are assumed to be wide-sense (or weakly) stationary (WSS) processes. Therefore, the problem can be stated as follows:

Design a filter that produces an estimate \( \hat{d}(n) \) using a linear combination of the data \( x(n) \) such that the mean square error (MSE) function (cost function)

\[
J = E\{(d(n) - \hat{d}(n))^2\} = E\{e^2(n)\}
\]

is minimized.

Depending on how the data \( \{ x(n) \} \) and the desired signal \( d(n) \) are related, there are four basic problems that need solution: filtering, smoothing, prediction, and deconvolution.

Observe that in (5.83) the LS formula appears again but with the difference that we take the expectation (ensemble average) value of the square value of the error. To proceed with our development and create useful formula, we will assume initially that we are able to repeat the experiment, produce the probability density function (pdf), and then find the MSE using the formula:
\[ E\{e^2(n)\} = \int_{-\infty}^{\infty} e^2(n)f(e(n))de(n) \]  

where:

\( f(.) \) is the pdf of the random variable (rv) \( e(n) \) of the error at time \( n \)

However, when we proceed to find the final formula, we will proceed to define the signals as ergotic and then find their means, variances, autocorrelation, and cross-correlation using the approximate formulas in time domain as were developed in Chapter 4.

### 5.3.1 The FIR Wiener Filter

Let the sample response (filter coefficients) of the desired FIR filter, shown in Figure 5.8, be denoted by \( w \). In this chapter, we use \( w \)'s for filter coefficients since it is customary to use this symbol in books involving adaptive and optimum filtering of random signals. This filter will process the real-valued stationary process \( \{x(n)\} \) to produce an estimate \( \hat{d}(n) \) of the desired real-valued signal \( d(n) \). Without loss of generality, we will assume, unless otherwise stated, that the processes \( \{x(n)\} \), \( d(n) \), and so on have zero mean values. Furthermore, assuming that the filter coefficients do not change with time, the output of the filter is equal to the convolution of the input and the filter coefficients. Hence, we obtain

\[
\hat{d}(n) = \sum_{m=0}^{M-1} w_m x(n-m) = w x^T(n)
\]  

![Figure 5.8](image-url)}
where:

\( M \) is the number of filter coefficients

\[
\mathbf{w} = [w_0 \ w_1 \ \cdots \ w_{M-1}]^T
\]

\( \mathbf{x}(n) = [x(n) \ x(n-1) \ x(n-2) \ \cdots \ x(n-M+1)]^T \) \hspace{1cm} (5.86)

The Wiener Filter

The MSE [see (5.83)] is given by

\[
J(\mathbf{w}) = E\{e^2(n)\} = E\{[d(n)-\mathbf{w}^T\mathbf{x}(n)]^2\}
\]

\[
= E\{[d(n)-\mathbf{w}^T\mathbf{x}(n)][d(n)-\mathbf{w}^T\mathbf{x}(n)]^T\}
\]

\[
= E[d^2(n)] - 2\mathbf{w}^T E[d(n)x(n)] + \mathbf{w}^T E[x(n)x^T(n)]\mathbf{w}
\]

\[
= \sigma_d^2 - 2\mathbf{w}^T \mathbf{p}_{dx} + \mathbf{w}^T \mathbf{R}_x \mathbf{w}
\]

where:

\( \mathbf{w}^T \mathbf{x}(n) = \mathbf{x}^T(n)\mathbf{w} = \text{number} \)

\( \sigma_d^2 = \text{variance of the desired signal, } d(n) \)

\( \mathbf{p}_{dx} = [p_{dx}(0) \ p_{dx}(1) \ \cdots \ p_{dx}(M-1)]^T = \text{cross-correlation vector} \)

\( \mathbf{R}_x = \text{correlation matrix of the input data, and it is symmetric because the random process is assumed to be stationary. Therefore, we have the} \)

\[
\mathbf{R}_x = \begin{bmatrix}
E[x(n)x(n)] & E[x(n)x(n-1)] & \cdots & E[x(n)x(n-M+1)] \\
E[x(n-1)x(n)] & E[x(n-1)x(n-1)] & \cdots & E[x(n-1)x(n-M+1)] \\
& \vdots & \ddots & \vdots \\
E[x(n-M+1)x(n)] & E[x(n-M+1)x(n-1)] & \cdots & E[x(n-M+1)x(n-M+1)]
end{bmatrix}
\]

\[
= \begin{bmatrix}
r_x(0) & r_x(1) & \cdots & r_x(M-1) \\
r_x(-1) & r_x(0) & \cdots & r_x(M-2) \\
& \vdots & \ddots & \vdots \\
r_x(-M+1) & r_x(-M+2) & \cdots & r_x(0)
end{bmatrix}
\]
Adaptive Filtering

relation \( r_{xx}(k) = r_{xx}(-k) \). Since in practice we generally have one realization, we assume that the signal is ergotic and we find the correlations using the time approach. Hence, the correlation matrix based on realization will be an approximate one and not the theoretical one based on ensemble averages.

Example 5.3.1

Let us assume that we have found the sample autocorrelation coefficients \((r_{xx}(0) = 1.0, r_{xx}(1) = 0)\) from a set of data \( \{x(n)\} \), which, in addition to noise, contains the desired signal. Furthermore, let the variance of the desired signal be \( \sigma_d^2 = 24.50 \) and the cross-correlation vector be \( \mathbf{p}_{dx} = [2 \quad 4.5]^T \). It is desired to find the surface defined by the mean-square function \( J(w) \) which is of second order with respect to the filter coefficients \( w \).

Solution: Substituting the values given above in (5.87), we obtain

\[
J(w) = 24.50 - 2[w_0 \quad w_1] \begin{bmatrix} 2 \\ 4.5 \end{bmatrix} + [w_0 \quad w_1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \\
= 24.50 - 4w_0 - 9w_1 + w_0^2 + w_1^2
\]

Note that the equation is quadratic with respect to filter coefficients and it is true for any number of filter coefficients. This is because we used the MSE approach for minimizing the error. Figure 5.9 shows the schematic representation of the Wiener filter. The data are the sum of the desired signal and the noise. From the data, we find the correlation matrix and the cross-correlation between the desired signal and the data. Note that to find the optimum Wiener filter coefficients the desired signal is needed. Figure 5.10 shows the **MSE surface**. This surface is found by inserting different values of \( w_0 \) and \( w_1 \) into the function \( J(w) \). The values of the coefficients that correspond to the bottom of the surface are the **optimum Wiener coefficients**. The vertical distance from the \( w_0 - w_1 \) plane to the bottom of the surface is known as the **minimum error**, \( J_{\text{min}} \), and corresponds to the optimum Wiener coefficients. We observe that the minimum height of the surface corresponds to about \( w_0 = 2 \) and \( w_1 = 4.5 \), which are the optimum coefficients as
The Wiener Filter

we will learn how to find them in Section 5.4. Figure 5.11 shows an adaptive FIR filter. The following Book MATLAB program is used to produce the MSE surface as shown in Figure 5.10.

**Book MATLAB Program**

```matlab
>> x = -2:0.2:10; y = -1:0.2:10;
>> [X,Y] = meshgrid(x,y);
```

![Figure 5.11](image)
5.4 THE WIENER SOLUTION

In Figure 5.10, we observe that there exists a plane touching the parabolic surface at its minimum point, and is parallel to the \( w \)-plane. Furthermore, we observe that the surface is concave upward, and therefore, calculus tell us that the first derivative of the MSE with respect to \( w_0 \) and \( w_1 \) must be zero at the minimum point and the second derivative must be positive. Hence, we write

\[
\frac{\partial J(w_0,w_1)}{\partial w_0} = 0 \quad \frac{\partial J(w_0,w_1)}{\partial w_1} = 0 \quad (5.90a)
\]

\[
\frac{\partial^2 J(w_0,w_1)}{\partial^2 w_0} > 0 \quad \frac{\partial^2 J(w_0,w_1)}{\partial^2 w_1} > 0 \quad (5.90b)
\]

For a two-coefficient filter, (5.87) becomes

\[
J(w_0,w_1) = w_0^2 r_x(0) + 2w_0w_1r_x(1) + w_1^2 r_x(0) - 2w_0r_{dx}(0) - 2w_1r_{dx}(1) + \sigma_d^2 \quad (5.91)
\]

Introducing next the above equation into (5.90a) produces the following set of equations:

\[
2w_0^2 r_x(0) + 2w_0w_1r_x(1) - 2r_{dx}(0) = 0 \quad (5.92a)
\]

\[
2w_1^2 r_x(0) + 2w_0w_1r_x(1) - 2r_{dx}(1) = 0 \quad (5.92b)
\]

The above system can be written in the following matrix form called the Wiener–Hopf equation:

\[
R_x w^o = p_{dx} \quad (5.93)
\]

where:

the superscript “\( o \)” indicates the optimum Wiener solution for the filter

Note that to find the correlation matrix we must know the second-order statistics. Furthermore, we must reiterate that mostly we have only one realization at hand and, as a consequence, the data matrix is an approximate one. If, in addition, the matrix is invertible, the optimum filter is given by

\[
w^o = R_x^{-1} p_{dx} \quad (5.94)
\]

For an \( M \)-order filter, \( R_x \) is an \( M \times M \) matrix, \( w^o \) is an \( M \times 1 \) vector, and \( p_{dx} \) is also an \( M \times 1 \) vector.
The Wiener Filter

If we differentiate again (5.92) with respect to $w_0^o$ and $w_1^o$, which is equivalent in differentiating $J(w)$ twice, we find that it is equal to $2r_x(0)$. But $r_x(0) = E\{x(m)x(m)\} = \sigma_x^2 > 0$, and then the surface is concave upward. Therefore, the extremum is the minimum point of the surface. If we, next, introduce (5.94) in (5.87), we obtain the minimum error:

$$J_{\text{min}} = \sigma_d^2 - p_{dx}^T w^o = \sigma_d^2 - p_{dx}^T R_x^{-1} p_{dx} \quad (5.95)$$

which indicates that the minimum point of the error surface is at a distance $J_{\text{min}}$ above the $w$-plane. The above equation shows that if no correlation exists between the desired signal and the data, the error is equal to the variance of the desired signal.

The problem we are facing is how to choose the length of the filter $M$. In the absence of a priori information, we compute the optimum coefficients, starting from a small reasonable number. As we increase the number, we check the minimum MSE (MMSE), and if its value is small enough, for example, $\text{MMSE} < 0.01$, we accept the corresponding number of the coefficients.

**Example 5.4.1**

It is desired to obtain the filter coefficients of an unknown filter (system), which is shown in Figure 5.12, together with the Wiener filter. The coefficients of the unknown filter are $b_0 = 1$ and $b_1 = 0.38$.

**Solution:** The following Book m-file is used:

**Figure 5.12**
Book m-File: ex5_4_1

%Book m-file: ex5_4_1
v = 0.5*(rand(1,20)-0.5); %v = noise vector
% (20 uniformly distributed rv's with mean zero);
x = randn(1,20); %x = data vector entering
% the system and the Wiener filter (20 normal
% distributed rv's with mean zero);
sysout = filter([1 0.38],1,x); %sysout = system
% output with x as input; filter(b,a,x) is a
% MATLAB function, where b is the
% vector of the coefficients of the ARMA numerator,
% a is the vector of the coefficients
% of the ARMA denominator (see (2.4.7));
dn = sysout+v;
rx = lms_sample_biased_autoc(x,2); %book MATLAB
% function with lag = 2;
Rx = toeplitz(rx); %toeplitz() is a MATLAB function
% that gives the symmetric autocorrelation matrix;
pdx = xcorr(x,dn,'biased'); %xcorr() is a MATLAB function
% that gives a symmetric biased cross-correlation;
p = pdx(1,19:20);
w = inv(Rx)*p';
dnc = lms_sample_biased_autoc(dn,1); %s2dn = variance
% of the desired signal;
jmin = dnc-p*w;

Some typical values obtained after running the program were w=[0.9205 0.3855],
R_x=[0.7447 -0.0486; -0.0486 0.7447], and J_{min}=0.0114.

5.4.1 Orthogonality Condition

In order for the set coefficients to minimize the cost function J(w), it is necessary
and sufficient that the derivatives of J(w) with respect to w_k be equal to zero for
k = 0, 2, 3, ..., M - 1. Hence,

$$\frac{\partial J}{\partial w_k} = \frac{\partial}{\partial w_k} E[e(n)e(n)] = 2E \left\{ e(n) \frac{\partial e(n)}{\partial w_k} \right\} = 0 \quad \text{and}$$

$$e(n) = d(n) - \sum_{m=0}^{M-1} w_m x(n-m) \quad (5.96)$$

Therefore, it follows that

$$\frac{\partial e(n)}{\partial w_k} = -x(n-k) \quad (5.97)$$

Therefore, (5.96) becomes

$$E[e^2(n)x(n-k)] = 0 \quad k = 0, 1, 2, ..., M - 1 \quad (5.98)$$
where the superscript “o” denotes that the corresponding $w_k$’s used to find the estimation error $e^o(n)$ are the optimal ones. Figure 5.13 illustrates the orthogonality principle where the error $e^o(n)$ is orthogonal (perpendicular) to the data set $\{x(n)\}$ when the estimator employs the optimum set of filter coefficients.

Next, let us continue and write

$$E \{e^o(n) \hat{d}^o(n)\} = E \left\{ e^o(n) \sum_{i=0}^{M-1} w_i^o(n)x(n-i) \right\} = \sum_{i=0}^{M-1} w_i^o(n)E \{e^o(n)x(n-i)\} \quad (5.99)$$

where:

$\hat{d}^o(n)$ is the Wiener filter output when its tap weights (coefficients) are set to their optimum values

Then, using (5.98) and (5.99), we obtain

$$E \{e^o(n) \hat{d}^o(n)\} = 0 \quad (5.100)$$

Note: The optimized Wiener filter output and the estimated error are orthogonal.

5.4.2 Normalized Performance Equation

From the error equation, we write

$$d(n) = e^o(n) + \hat{d}^o(n) \quad (5.101)$$

Squaring both sides of the above equation and taking the ensemble average (expectation), we get

$$E \{d^2(n)\} = E \{[e^o(n)]^2\} + E \{[\hat{d}^o(n)]^2\} + 2E \{e^o(n)\hat{d}^o(n)\} \quad (5.102)$$
But \( E \left[ \left( e^n(n) \right)^2 \right] = J_{\min} \) (see Problem 5.4.3) and the last term is zero due to (5.100). Thus, we obtain

\[
J_{\text{min}} = E \left\{ d^n(n) \right\} - E \left\{ \left( \hat{d}^n(n) \right) \right\}
\]  
(5.103)

**Note:** The MMSE at the Wiener filter output is the difference between the MSE of the desired output and the MSE of the best estimate of that at the filter output.

The normalized performance function is defined by

\[
\zeta = \frac{J}{E[d^2(n)]}, \quad \zeta_{\text{min}} = \frac{E[d^2(n)] - E[\hat{d}^2(n)]}{E[d^2(n)]} = 1 - \frac{E[\hat{d}^2(n)]}{E[d^2(n)]}
\]  
(5.104)

**Note:** The minimum performance is an indication of the ability of the filter to estimate the desired output. A value of \( \zeta_{\min} \) close to zero is an indication of good performance of the filter, and a value close to one indicates poor performance.

### 5.4.3 Canonical Form of the Error-Performance Surface

We have developed above (5.87) the MSE in the form:

\[
J(w) = \sigma_w^2 - w^T p - p^T w + w^T R w
\]  
(5.105)

The above equation can be written in the form:

\[
J(w) = \sigma_w^2 - p^T R^{-1} p + (w - R^{-1} p)^T R (w - R^{-1} p)
\]  
(5.106)

At the optimum condition, the above equation becomes

\[
J(w) = J_{\text{min}} + (w - w^o)^T R (w - w^o)
\]  
(5.107)

We observe that at the optimum condition \( J(w^o) = J_{\text{min}} \).

To simplify the error-performance surface, we proceed to change the basis on which is defined. To proceed, we use **eigendecomposition** and express the autocorrelation matrix in the form:

\[
R = Q \Lambda Q^T
\]  
(5.108)

The matrix \( \Lambda \) is a diagonal matrix consisting of the eigenvalues \( \lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{M-1} \) of the autocorrelation matrix and the matrix \( Q \) has for its columns \( q_0, q_1, q_2, \ldots, q_{M-1} \) associated with these eigenvalues, respectively. Substituting (5.108) in (5.107), we obtain

\[
J(w) = J_{\text{min}} + (w - w^o)^T Q \Lambda Q^T (w - w^o)
\]  
(5.109)

Next, we introduce the transformation

\[
v = Q^T (w - w^o)
\]  
(5.110)

into (5.109) to obtain
The Wiener Filter

\[ J(\mathbf{w}) = J_{\text{min}} + \mathbf{v}^T \Lambda \mathbf{v} = J_{\text{min}} + \sum_{k=0}^{M-1} \lambda_k v_k^2 \]  

(5.111)

where:
\( v_k \) is the \( k \)th component of the vector \( \mathbf{v} \). The components \( v_k \) constitute the principal axes of the surface.

5.5 WIENER FILTERING EXAMPLES

The examples in this section illustrate the use and utility of the Wiener filter.

Example 5.5.1 (Filtering)

Filtering of noisy signals (noise reduction) is extremely important, and the method has been used in many applications such as speech in noisy environment, reception of data across a noisy channel, and enhancement of images.

Let the received signal be \( x(n) = d(n) + v(n) \), where \( v(n) \) is a noise with zero mean and variance \( \sigma_v^2 \), and it is uncorrelated with the desired signal, \( d(n) \). Hence,

\[
p_{d,v}(m) = E\{d(n)x(n-m)\} = E\{d(n)d(n-m)\} + E\{d(n)v(n-m)\} \\
= E\{d^2(m)\} = r_{dd}(m)
\]

(5.112)

Similarly, we obtain

\[
r_{x,v}(m) = E\{x(n)x(n-m)\} = E\{d(n)+v(n)\}[d(n-m)+v(n-m)] \\
= E\{d(n)d(n-m)\} + E\{v(n)d(n-m)\} + E\{d(n)v(n-m)\} \\
+ E\{v(n)v(n-m)\} \\
= r_{dd}(m) + r_{dv}(m)
\]

(5.113)

We have assumed that \( d(n) \) and \( v(n) \) are uncorrelated, \( v(n) \) has zero mean value. Therefore, the Wiener equation becomes

\[
(R_d + R_v)w^\ast = p_{d,v}
\]

(5.114)

The following Book m-file is used to produce the results shown in Figure 5.14.

Book m-File: ex5_5_1

```
%ex5_5_1 m-file
n = 0:511;
d = sin(0.1*pi*n); %desired signal
v = 0.5*randn(1,512); %white Gaussian noise
% with zero mean value;
x = d+v; %input signal to Wiener filter;
rd = lms_sample_biased_autoc(d,20); %rdx = rd = biased autoc function of the
% desired signal (see (5.112));
rv = lms_sample_biased_autoc(v,20); %rv = biased autoc function of the noise;
R = toeplitz(rd(1,1:12))+toeplitz(rv(1,1:12)); %see (5.114);
```
pdx = rd(1,1:12);  
w = inv(R)*pdx';  
y = filter(w',1,x);  % output of the filter;

But

\[
\sigma_n^2 = \sigma_d^2 + \sigma_v^2 \quad \left[ \sigma_n^2 = r_{nn}(0), \sigma_d^2 = r_{dd}(0), \sigma_v^2 = r_{vv}(0) \right]
\]  

(5.115)

and hence, from MATLAB function \( \text{var()} \), we obtain for one run: \( \text{var}(d) = \text{var}(x) - \text{var}(v) = 0.5094 \) and \( J_{\min} = 0.5094 - p_{dt} \cdot w^* = 0.0432 \).

Example 5.5.2 (Filtering)

It is desired to find a two-coefficient Wiener filter for the communication channel that is shown in Figure 5.15. Let \( v_1(n) \) and \( v_2(n) \) be the white noises with zero mean, uncorrelated with each other, and with \( d(n) \), and have the following variances: \( \sigma_{v1}^2 = 0.31 \), \( \sigma_{v2}^2 = 0.12 \). The desired signal produced by the first filter shown in Figure 5.15 is

\[
d(n) = -0.796d(n-1) + v_1(n)
\]

(5.116)

Therefore, the autocorrelation function of the desired signal becomes
The Wiener Filter

\[ \begin{align*}
E\{d(n)d(n)\} &= 0.796^2 E\{d^2(n-1)\} - 2 \times 0.796E\{d(n-1)\}E\{v_1(n)\} + E\{v_1^2(n)\} \\
\sigma_d^2 &= 0.796^2 \sigma_v^2 + \sigma_{v_1}^2 \quad \text{or} \quad \sigma_d^2 = \frac{0.31}{(1-0.796^2)} = 0.8461 
\end{align*} \]  

(5.117)

From the second filter, we obtain the relation:

\[ d(n) = u(n) - 0.931 l u(n-1) \]  

(5.118)

Introducing the above equation, for the expression of \( d(n) \), into (5.116), we obtain

\[ u(n) - 0.135 u(n-1) - 0.7411 l u(n-2) = v_1(n) \]  

(5.119)

But \( x(n) = u(n) + v_2(n) \), and hence, the vector form of the set becomes \( x(n) = u(n) + v_2(n) \). Therefore, the autocorrelation matrix \( R_x \) becomes

\[ E\left\{ x(n)x^T(n) \right\} = R_x = E\left\{ [u(n) + v_2(n)][u^T(n) + v_2^T(n)] \right\} = R_u + R_{v_2} \]  

(5.120)

We used assumption that \( u(n) \) and \( v_2(n) \) are uncorrelated zero-mean random sequences, which implies that \( E\{v_2(n)u^T(n)\} = E\{u(n)v_2^T(n)\} = 0 \). Next, we multiply (5.119) by \( u(n-m) \) and take the ensemble average of both sides, which results in the following expression:

\[ r_{uu}(m) - 0.135 r_{uu}(m-1) - 0.7411 r_{uu}(m-2) = r_{v_1 u}(m) = E\{v_1(n)u(n-m)\} \]  

(5.121)

Setting \( m = 1 \) and then \( m = 2 \) in the above equation, we obtain

\[ \begin{bmatrix} r_{uu}(0) & r_{uu}(-1) \\ r_{uu}(1) & r_{uu}(0) \end{bmatrix} \begin{bmatrix} -0.1350 \\ -0.7411 \end{bmatrix} = \begin{bmatrix} -r_{v_1 u}(1) \\ -r_{v_1 u}(2) \end{bmatrix} \quad \text{Yule–Walker equation} \]  

(5.122)
because \( v_i(n) \) and \( u(n-m) \) are uncorrelated. If we set \( m = 0 \) in (5.121), it becomes

\[
r_{uu}(0) - 0.135r_{uu}(-1) - 0.7411r_{uu}(-2) = E\{v_i(n)u(n)\}
\]

(5.123)

Next, we substitute the value of \( u(n) \) from (5.119) in the above equation; taking into consideration that \( v \) and \( u \) are independent rv's, we obtain

\[
r_{uu}(1) = \frac{0.135}{1-0.7411} r_{uu}(0) = \frac{0.135}{0.2589} \sigma_u^2
\]

(5.124)

Substitute the above equation in the second equation of the set of (5.122), we obtain

\[
r_{uu}(2) = 0.135 - \frac{0.135}{0.2589} \sigma_v^2 + 0.7411 \sigma_u^2
\]

(5.125)

Therefore, the last three equations give the variance of \( u \):

\[
\sigma_u^2 = \frac{\sigma_v^2}{0.3282} = \frac{0.31}{0.3282} = 0.9445
\]

(5.126)

Using (5.126), (5.124), and the given value \( \sigma_v^2 = 0.12 \), we obtain the correlation matrix:

\[
\mathbf{R}_x = \mathbf{R}_u + \mathbf{R}_v = \begin{bmatrix} 0.9445 & 0.4925 \\ 0.4925 & 0.9445 \end{bmatrix} + \begin{bmatrix} 0.12 & 0 \\ 0 & 0.12 \end{bmatrix} = \begin{bmatrix} 1.0645 & 0.4925 \\ 0.4925 & 1.0545 \end{bmatrix}
\]

(5.127)

We multiply (5.118) by \( u(n) \) first and next by \( u(n-1) \), and taking the ensemble average of the results, we obtain the vector \( \mathbf{p} \) equal to

\[
\mathbf{p}_{dx} = [0.4860, -0.3868]^T
\]

(5.128)

\[\blacksquare\]

### 5.5.1 Minimum MSE

Introducing the above results in (5.87), we obtain the MSE surface (cost function) as a function of the filter coefficients, \( w \)'s. Hence, we write

\[
J(w) = 0.8461 - 2\left[w_0 \quad w_1 \right] \begin{bmatrix} 0.4860 \\ -0.3868 \end{bmatrix} + \left[w_0 \quad w_1 \right] \begin{bmatrix} 1.0645 & 0.4925 \\ 0.4925 & 1.0645 \end{bmatrix} \left[w_0 \quad w_1 \right]
\]

(5.129)

\[
= 0.8461 - 0.972w_0 + 0.7736w_1 + 1.0645w_0^2 + 1.0645w_1^2 + 0.985w_0w_1
\]

The MSE surface and its contour plots are shown in Figure 5.16.

### 5.5.2 Optimum Filter (\( w^o \))

The optimum filter was found above to be \( \mathbf{R}_w w^o = \mathbf{p}_{dx} \), and in this case, it takes the following form:
The Wiener Filter

\[ w^\omega = R_x^{-1} p_{dx} = \begin{bmatrix} 1.1953 & -0.5531 \\ -0.5531 & 1.1953 \end{bmatrix} \begin{bmatrix} 0.4860 \\ -0.3868 \end{bmatrix} = \begin{bmatrix} 0.7948 \\ -0.7311 \end{bmatrix} = \begin{bmatrix} w^\omega_0 \\ w^\omega_2 \end{bmatrix} \] (5.130)

The MMSE was found above, and for this case, we find

\[
J_{\text{min}} = \sigma_d^2 - p_{dx}^T R_x^{-1} p_{dx} \\
= 0.8461 - [0.4860 \quad -0.3868] \begin{bmatrix} 1.1953 & -0.5531 \\ -0.5531 & 1.1953 \end{bmatrix} \begin{bmatrix} 0.4860 \\ -0.3868 \end{bmatrix} \]

\[ = 0.1770 \] (5.131)

**Example 5.5.3 (System Identification)**

It is desired, using a Wiener filter, to estimate the unknown impulse response coefficients \( h \)'s of an unknown FIR system (see Figure 5.17). The input \( \{x(n)\} \) is a set of zero-mean independent identically distributed (iid) rv's with variance \( \sigma_x^2 \). Let the impulse response \( h \) of the filter be \( h = [0.9 \quad 0.6 \quad 0.2]^T \). Since the input sequence \( \{x(n)\} \) is a set of zero mean iid rv's, the correlation matrix \( R_x \) is a diagonal matrix with elements having values \( \sigma_x^2 \). The desired signal \( d(n) \) is the output
of the unknown filter and is given by \( d(n) = 0.9x(n) + 0.6x(n-1) + 0.2x(n-2) \) (see Figure 3.8a). Therefore, the cross-correlation output is given by

\[
p_{\Delta}(i) = E\{d(n)x(n-i)\} = E\{(0.9x(n) + 0.6x(n-1) + 0.2x(n-2))x(n-i)\}
\]

\[
= 0.9E\{x(n)x(n-i)\} + 0.6E\{x(n-1)x(n-i)\} + 0.2E\{x(n-2)x(n-i)\}
\]

\[
= 0.9r_{xx}(i) + 0.6r_{xx}(i-1) + 0.2r_{xx}(i-2)
\]

Hence, we obtain \([r_{xx}(m) = 0 \text{ for } m \neq 0]\): \(p_{\Delta}(0) = 0.9\sigma_i^2, p_{\Delta}(1) = 0.6\sigma_i^2\). The optimum filter is

\[
w^o = R_{\Delta}^{-1}p_{\Delta} = \frac{1}{\sigma_i^2} \begin{bmatrix} 1 & 0 & 0.9 \\ 0 & 1 & 0.6 \end{bmatrix} = \frac{1}{\sigma_i^2} \begin{bmatrix} 0.9 \\ 0.6 \end{bmatrix}
\]

and the MMSE is (assuming \(\sigma_i^2 = 1\))

\[
J_{\text{min}} = \sigma_d^2 - \left[ \begin{array}{cc} 0.9 & 0.6 \\ 0.6 & 0.9 \end{array} \right] = 0.81 + 0.36 + 0.04 = 1.21,
\]

and hence,

\[
J_{\text{min}} = 1.21 - (0.9^2 + 0.6^2) = 0.04.
\]

**Book m-Function for System Identification (Wiener filter) :** \([w,jm]=\text{lms\_wiener\_}

function(w,jm) = lms_wiener_fir_filter(x,h,M)

\%w = data entering both the unknown filter(system)
\%and the Wiener filter;
\%d = the desired signal = output of the unknown
\%system; length(d) = length(x);
\%M = number of coefficients of the Wiener filter;
\%w = Wiener filter coefficients;
\%jm = minimum mean square error;

\[
F_{I}gure\ 5.17
\]
The Wiener Filter

\[
d_1 = \text{conv}(x,h); \quad \% h = \text{vector with coefficients of the unknown system;}
\]
\[
d = d_1(1,1:\text{length}(x));
\]
\[
pdx = \text{xcorr}(d,x,'biased');
\]
\[
p = pdx(1,(\text{length(pdx)}+1)/2:(\text{length(pdx)}+1)/2+M-1);
\]
\[
rx = \text{lms\_sample\_biased\_autoc}(x,M); \quad \% M \text{ plays the role of the lag number in correlation functions;}
\]
\[
R = \text{toeplitz}(rx);
\]
\[
w = \text{inv}(R)*p';
\]
\[
\text{jm} = \text{var}(d) - p*w; \quad \% \text{var()} \text{ is a MATLAB function;}
\]

We set for the unknown system coefficients the vector \( h = [0.9 \quad 0.6 \quad 0.2] \). We also assumed \( M = 6 \) for the unknown filter number of coefficients. The input signal was random normally distributed with zero mean and variance of one, \( \mathcal{N}(0,1) \), or \( x = \text{randn}(1,512) \). The results were as follows:

\[
w = \begin{bmatrix} 0.8998 & 0.5998 & 0.2000 & 0.0000 & -0.0006 & -0.0016 \end{bmatrix}, \quad \text{jm} = -0.0052
\]
as we were expecting.

Example 5.5.4 (Noise Canceling)

In many practical applications, there exists a need to cancel the noise added to a signal. For example, we are talking on the cell phone inside the car, and the noise of the car, radio, and so on is added to the message when we are trying to transmit. Similar circumstance appears when pilots in planes and helicopters try to communicate or tank drivers try to do the same. Figure 5.18 shows pictorially the noise contamination situations. Observe that the noise added to the signal and the other component entering the Wiener filter emanate from the same source but follow different paths in the same environment. This indicates that there is somewhat difference in value. The output of the Wiener filter will approximate the noise added to the desired signal, and thus, the error will be close to the desired signal. The Wiener filter in this case is

\[
R_{v1}w^\ast = p_{v1v2}
\]  \hspace{1cm} (5.133)

because the signal from the Wiener filter \( \{ \hat{v}_1 \} \) should be equal to \( \{ v_1 \} \) so that the desired signal is recaptured, in this case the signal \( e(n) \). The individual components of the vector \( p_{v1v2} \) are

\[
e(n) = x(n) - \hat{v}_1(n)
\]
\[
= d(n) + v_1(n) - \hat{v}_1(n)
\]

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5_18.png}
\caption{Figure 5.18}
\end{figure}
\[ p_{v_1 v_2}(m) = E\{v_1(n)v_2(n-m)\} = E\{[x(n) - d(n)]v_2(n-m)\} \]
\[ = E\{x(n)v_2(n-m)\} - E\{d(n)v_2(n-m)\} \]
\[ = E\{x(n)v_2(n-m)\} - E\{d(n)\}E\{v_2(n-m)\} \]
\[ = E\{x(n)v_2(n-m)\} - E\{d(n)\}p_{v_2}(m) \]  

since \(d(n)\) and \(v_2\) are uncorrelated, and it was assumed that the noise had zero mean value. Therefore, (5.133) becomes

\[ R_{v_2}w^T = p_{v_2} \]  

(5.135)

To demonstrate the effect of the Wiener filter, let \(d(n) = 2[0.99^n \sin(0.1n\pi + 0.2\pi)]\), \(v_1(n) = 0.8v_1(n-1) + v(n)\), and \(v_2(n) = 0.2v_1\), where \(v(n)\) is white noise with zero mean value. The correlation matrix \(R_{v_2}\) and the cross-correlation vector \(p_{xv_2}\) are found using the sample biased correlation equation:

\[ \hat{r}_{v_2v_2}(k) = \frac{1}{N} \sum_{n=0}^{N-1} v_2(n)v_2(n-k) \quad k = 0, 1, \ldots, K - 1, \quad K << N \]  

(5.136)

\[ \hat{p}_{xv_2}(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n)v_2(n-k) \quad k = 0, 1, \ldots, K - 1, \quad K << N \]

Figure 5.19 shows simultaneous results for a sixth-order Wiener filter using the Book m-function given below. We set \(a_1 = 0.9\).
The Wiener Filter

Book m-Function for Noise Canceling

```matlab
function [d,w,xn] = lms_wiener_noise_canceling(dn,a1,M,N)
    %[d,w,xn] = lms_wiener_noise_canceling(dn,a1,a2,v,M,N);
    %dn = desired signal;
    %a1 = first-order IIR coefficient,a2 = first-order IIR coefficient;
    %v = noise;M = number of Wiener filter coefficients;
    %N = number of sequence elements of dn (desired signal) and v (noise);
    %d = output desired signal;
    %w = Wiener filter coefficients;xn = corrupted signal;en = xn-v1 = d;
    v1(1) = 0;v2(1) = 0;
    for n = 2:N
        v1(n) = a1*v1(n-1)+(rand-0.5);
        v2(n) = 0.2*v1(n);
    end;
    v2autoc = lms_sample_biased_autoc(v2,M);
    xn = dn+v1;
    Rv2 = toeplitz(v2autoc);
    p1 = xcorr(xn,v2,'biased');%MATLAB function;
    if M>N
        disp(['error:M must be less than N']);
    end;
    R = Rv2(1:M,1:M);
    p = p1(1,(length(p1)+1)/2: (length(p1)+1)/2+M-1);
    w = inv(R)*p';
    yw = filter(w,1,v2);%MATLAB function;
    d = xn-yw(:,1:N);
```

Example 5.5.5 (Book Proposed Self-Correcting Wiener filter)

We can also arrange the standard single Wiener filter in a series form as shown in Figure 5.20. This book proposed configuration permits us to process the signal using filters with fewer coefficients, thus saving in computation. Figure 5.21a shows the desired signal, which is a decaying cosine wave. Figure 5.21b shows the signal with a white zero-mean additive noise. Figure 5.21c shows the output of the first stage of a self-correcting Wiener filter (SCWF), \( x_n(1) \). Each stage Wiener filter has two coefficients. Figure 5.21d shows the output of the second stage of the SCWF.

Example 5.5.6

Use the proposed method of the extended input signal to improve the desired results.

Solution: We have used the system recognition approach (see Example 5.5.3) to verify the proposed method. The following Book MATLAB program was used.
Adaptive Filtering

Book MATLAB Program

```matlab
>> x = randn(1,8);
>> [w, jm] = lms_wiener_fir_filter(x, [0.9 0.6 0.2], 5);
w = [0.7808 0.5483 -0.1444 -0.3132]'
>> x1 = [x 0.2*x 0.1*x];
```

FIGURE 5.20

**(a)**

![Graph 1](image1)

**(b)**

![Graph 2](image2)

**(c)**

![Graph 3](image3)

**(d)**

![Graph 4](image4)

FIGURE 5.21
The Wiener Filter

\begin{verbatim}
>>[w1,jm1] = lms_wiener_fir_filter(x1,[0.9 0.6 0.2],5);
w1 = [0.8975 0.5990 0.1955 0.0002 -0.0040]'
\end{verbatim}

5.5.3 LINEAR PREDICTION

Let us investigate the prediction problem as shown in Figure 5.22. It is desired to predict the signal in the presence of noise. The input to Wiener filter is

\[ y(n) = x(n) + v(n) \]  \hspace{1cm} (5.137)

The goal is to design a filter such that the value of the signal \( x(n+1) \) is predicted based on \( p \) linear combination of previous values of the input signal \( y(n) \). Hence, we write

\[ \hat{x}(n+1) = \sum_{k=0}^{p-1} w(k)y(n-1) = \sum_{k=0}^{p-1} w(k)[x(n-k) - v(n-k)] \]  \hspace{1cm} (5.138)

The Wiener equation is

\[ R_yw = r_{dy} \]  \hspace{1cm} (5.139)

If the noise \( v(n) \) is uncorrelated with the signal \( x(n) \), then the autocorrelation matrix of the signal \( y(n) \) is

\[ r_{yy}(k) = E\{y(n)y(n-k)\} = E\{(x(n)+v(n))(x(n-k)+v(n-k))\} \]
\[ = E\{x(n)x(n-\cdot)\} + E\{v(n)x(n-k)\} + E\{x(n)v(n-k)\} + E\{v(n)v(n-k)\} \]  \hspace{1cm} (5.140)
\[ = r_{xx}(k) + 0 + 0 + r_{vy}(k) = r_{xx}(k) + r_{vy}(k) \]

The vector cross-correlation between \( d(n) \), in this case \( x(n+1) \), and \( y(n) \) is

\[ r_{dy}(k) = E\{d(n)y(n-k)\} = E\{x(n+1)y(n-k)\} \]
\[ = E\{x(n+1)[x(n-k)+v(n-k)]\} = E\{x(n+1)x(n-k)\} = r_{xx}(n+1) \]  \hspace{1cm} (5.141)

Equation 5.140, with \( k = 0, 1, 2, \ldots, p-1 \), becomes \( R_y = R_x + R_v \). Therefore, the MSE is
\[ J_{\text{min}} = \sigma_d^2 - p_d^T w_0 = \sigma_d^2 - p_d^T R_x^{-1} p_d \tag{5.142} \]
gives the minimum error between the desired and the output of the Wiener filter.

**Example 5.5.7**

Let the noise be a white noise with variance \( \sigma_v^2 \). The noise is assumed to be uncorrelated with the signal \( \{x(n)\} \). Assume that the autocorrelation function of \( \{x(n)\} \) is \( r_{xx}(k) = \alpha^k \). We also assume a two-coefficient predictor filter, \([w_0, w_1]\).

**Solution:** The Wiener equation is (see Problem 5.5.1)

\[
(R_x + \sigma_v^2 I) w = r_d
\tag{5.143}
\]

The above equation becomes

\[
\begin{bmatrix}
  r_{xx}(0) + \sigma_v^2 & r_{xx}(1) \\
  r_{xx}(1) & r_{xx}(0) + \sigma_v^2
\end{bmatrix}
\begin{bmatrix}
  w_0 \\
  w_1
\end{bmatrix}
= 
\begin{bmatrix}
  r_d(0) \\
  r_d(1)
\end{bmatrix},
\]

or

\[
\begin{bmatrix}
  1 + \sigma_v^2 & a \\
  a & 1 + \sigma_v^2
\end{bmatrix}
\begin{bmatrix}
  w_0 \\
  w_1
\end{bmatrix}
= 
\begin{bmatrix}
  a \\
  a^2
\end{bmatrix}
\]

or

\[
\begin{bmatrix}
  w_0 \\
  w_1
\end{bmatrix}
= 
\frac{a}{(1 + \sigma_v^2) - a^2}
\begin{bmatrix}
  1 + \sigma_v^2 - a^2 \\
  a\sigma_v^2
\end{bmatrix}
\]

Let \( a = 0.8 \), \( \sigma_v^2 = 1.2 \), \( x(n) = 2\sin(0.1n\pi) \), and \( y(n) = x(n) + 0.5(\text{rand} - 0.5) \). Then the Wiener filter coefficients are as follows: \( w_0 = 0.2971 \), \( w_1 = 0.1829 \). Hence, \( J_{\text{min}} = [1.08^2] [0.2971 \ 0.1829]^T = 0.5858 \).

**PROBLEMS**

5.2.1 Find the unknown constant \( c \), using the LS method, so that \( \hat{x}(n) = ch(n) \), where \( h(n) \) is known, is the best approximation to \( \{x(n)\} \).

5.2.2 Verify (5.24).

5.2.3 Find the vector coefficient of the filter and the minimum error if the following are given:

\[
d = [1 \ 1 \ 1 \ 1]^T; \ x_1 = [0.7 \ 1.4 \ 0.4 \ 1.3]^T;
\]

\[
x_2 = [1.2 \ 0.6 \ 0.5 \ 1.1]^T; \ x_3 = [0.9 \ 1.1 \ 0.95 \ 1.05]^T
\]

5.2.4 The time-averaged correlation matrix \( R = X^TX \) (must be divided by \( N \) to be true time averaged) is invertible if the columns \( x_k \) of \( X \) are linearly independent or, equivalently, if and only if \( R \) is positive definite.

5.2.5 Let us assume that the signal from LTI system is constant, \( y(n) = a \), and we observe the data \( x(n) \), for \( n = 1, 2, \ldots, N \). Find the estimate value of \( a \) and \( \hat{a} \).

5.4.1 Find the Wiener solution using the Matrix Analysis in Appendix 2.
5.4.2 Verify that \( J_{\text{min}} = E[\{e^2(n)\}] \).
5.4.3 Find \( J_{\text{min}} \) using the orthogonality principle.
5.4.4 Find the variance \( \sigma_{\hat{d}}^2 \) of the estimate \( \hat{d}(n) \).

5.5.1 Verify (5.143).

5.5.2 (Multiple-step prediction) Develop the multiple-step prediction in which \( x(n+m) \) is to be predicted in terms of linear combination of the \( p \) values: \( x(n), x(n+1), \ldots, x(n-p+1) \).

5.5.3 Let the data entering the Wiener filter be given by \( x(n) = d(n) + v(n) \). The noise \( v(n) \) has zero mean value and unit variance, and is uncorrelated with the desired signal \( d(n) \). Furthermore, assume \( r_{dd}(m) = 0.9^m \) and \( r_{v}(m) = 18(m) \). Find \( p_{de}, w, J_{\text{min}} \), signal power, noise power, and signal-to-noise power.

5.5.4 Find the cost function and the MSE surface for the two systems shown in Figure P5.5.4. Given \( E\{s^2(n)\} = 0.9 \), \( E\{s(n)s(n-1)\} = 0.4 \), \( E\{d^2(n)\} = 3 \), \( E\{d(n)s(n)\} = 0.5 \), and \( E\{d(n)s(n-1)\} = 0.9 \).

5.5.5 Consider a two-coefficient Wiener filter with the following statistics:

\[
E\{d^2(n)\} = 2, \quad R = \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix}, \quad p = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}
\]

a. Use the above information to obtain the performance function (cost function), \( J \), of the filter.

b. By direct evaluation of the performance function, obtain the optimum values of the filter coefficients (tap weights).

b. Use the results from (a) in the performance function expression to obtain the MMSE of the filter.

5.5.6 Consider a three-coefficient Wiener filter with the following data:

\[
E\{d^2(n)\} = 5, \quad R = \begin{bmatrix} 1 & 0.25 & 0.125 \\ 0.25 & 1 & 0.25 \\ 0.125 & 0.25 & 1 \end{bmatrix}, \quad p = \begin{bmatrix} 1.5 \\ 0.5 \\ 0 \end{bmatrix}
\]

Repeat steps (a) and (b) of Problem 5.5.5.

Repeat steps (a) and (b) of Problem 5.5.5.

FIGURE P5.5.4
ADDITIONAL PROBLEMS

1. A Wiener filter has an input \( \{x(n)\} \) with an autocorrelation matrix

\[
R_x = \begin{bmatrix}
1 & 0.5 \\
0.5 & 1
\end{bmatrix}
\]

and a cross-correlation between the input and the desired signal is

\[
p = \begin{bmatrix}
0.4 \\
0.2
\end{bmatrix}
\]

Find (a) the coefficients, (b) the MMSE, and (c) the expansion as a function of the eigenvalues and eigenvectors.

2. Prove that the MMSE \( J_{\text{min}} \) is given by

\[
J_{\text{min}} = \sigma_d^2 - \sum_{k=1}^{M} \frac{q_k^2 p^2}{\lambda_k}.
\]

HINTS–SOLUTIONS–SUGGESTIONS

5.2.1

\[
J(c) = \text{Error criterion} = \sum_{n=0}^{N-1} [x(n) - ch(n)]^2.
\]

\[
\frac{\partial J(c)}{\partial c} = 0 = -2 \sum_{n=0}^{N-1} [x(n)h(n) - h^2(n)]
\]

or \( \hat{c} = \frac{\sum_{n=0}^{N-1} x(n)h(n)}{\sum_{n=0}^{N-1} h^2(n)} \)

5.2.2 For simplicity and without loss of generality, we assume a \( 2 \times 2 \) matrix and a \( 2 \times 1 \) vector.

\[
\frac{\partial J}{\partial w} = \frac{\partial}{\partial w} [E_y - 2pw^T + w^T Rw] = \frac{\partial E_y}{\partial w} - 2p^T \frac{\partial w}{\partial w} + \frac{\partial}{\partial w} (w^T Rw) = 0
\]

\[
= -2 \begin{bmatrix}
(p_1w_1 + p_2w_2) \\
(p_1w_1 + p_2w_2)
\end{bmatrix}
\]

or \( R\hat{w} = p \)

5.2.3 \textbf{Hint}: Follow Example 5.2.4.
5.2.4 If the columns of $X$ are linearly independent, then for every $y \neq 0$, we have $Xy \neq 0$. Hence, $y^T (X^T X) y = (Xy)^T X y = \|Xy\|^2 > 0$, which indicates that $R = X^T X$ is positive definite and therefore nonsingular. If the columns of $X$ are linearly dependent, then there is a $y_0 \neq 0$. Therefore, $X^T X y_0 = 0$ and $R$ is singular.

5.2.5

$$J(a) = \sum_{n=1}^{N} [x(n) - a]^2 \Rightarrow \frac{\partial J(a)}{\partial a} = \sum_{n=1}^{N} (-)2[x(n) - \hat{a}] = 0 \quad \text{or}$$

$$\sum_{n=1}^{N} x(n) - N \hat{a} = 0 \quad \text{or} \quad \hat{a} = \frac{1}{N} \sum_{n=1}^{N} x(n)$$

which is the sample mean estimator.

5.4.1

$$\frac{\partial J(w)}{\partial w} = \frac{\partial}{\partial w} (\sigma_o^2) - 2 \frac{\partial}{\partial w} (w^T p_{dx}) + \frac{\partial}{\partial w} (w^T R w) = 0 - 2 \frac{p_{dx}}{\text{Eq. A.2.67 App2}} + 2 \frac{R w}{\text{Eq. A.2.73 App2}} = 0$$

or $R w^o = p_{dx}$

5.4.2

$$J_{\min} = E\{[e^o(n)]^2\}$$

$$= E\{e^o(n)(d(n) - \hat{d}^o(n))\} = E\{e^o(n)d(n)\} - E\{e^o(n)\hat{d}^o(n)\}$$

$$= E\{e^o(n)d(n)\}$$

where $E\{e^o(n)\hat{d}^o(n)\} = 0 \quad \text{[see (5.100)]}$. Substituting (5.101) in the form

$$e^o(n) = d(n) - \hat{d}^o(n) = d(n) - \sum_{i=0}^{N-1} w_i^o(n)x(n-i),$$

we find

$$J_{\min} = E\left\{d(n) - \sum_{i=0}^{M-1} w_i^o(n)x(n-i)\right\} d(n)$$

$$= E\{d(n)\} - \sum_{i=0}^{M-1} w_i^o(n)E\{d(n)x(n-i)\} \quad \text{or}$$

$$J_{\min} = E\left\{d^2(n) - \sum_{i=0}^{M-1} w_i^o(n)p_i\right\},$$

which is the expanded form of the desired results.
5.4.3

\[ J = E \left\{ e(n) \left[ d(n) - \sum_{m=0}^{M-1} w_m(n)x(n-m) \right] \right\} = E \{ e(n)d(n) \} \]
\[ - \sum_{m=0}^{M-1} w_m(n)E \{ e(n)x(n-m) \} \]

If the coefficients have their optimum value, the orthogonality principle states that \( E \{ e(n)x(n-m) \} = 0 \). Hence,

\[ J_{\text{min}} = E \{ e(n)d(n) \} = E \{ d(n)d(n) - \sum_{m=0}^{M-1} w_m^o \cdot x(n-m)d(n) \} = \sigma_d^2 \]
\[ - \sum_{m=0}^{M-1} w_m^o E \{ d(n)x(n-m) \} = \sigma_d^2 - \sum_{m=0}^{M-1} w_m^o p_{dx}(m) \]
\[ = r_d(0) - \mathbf{p}_d^T \mathbf{w} = r_d(0) - \mathbf{p}_d^T \mathbf{R}_d^{-1} \mathbf{p}_d \]

5.4.4

\[ \hat{d}^o(n) = (\mathbf{w}^o)^T \mathbf{x}(n) \]
\[ \sigma_d^2 = E \left\{ (\mathbf{w}^o)^T \mathbf{x}(n) \mathbf{x}^T(n) \mathbf{w}^o \right\} = (\mathbf{w}^o)^T E \{ \mathbf{x}(n) \mathbf{x}^T(n) \} \mathbf{w}^o = \mathbf{p}^T \mathbf{w} = \mathbf{p}^T \mathbf{R}^{-1} \mathbf{p}; \]
\[ J_{\text{min}} = \sigma_d^2 - \sigma_d^2 = \sigma_d^2 - (\mathbf{w}^o)^T \mathbf{R} \mathbf{w} = \sigma_d^2 - \mathbf{p}^T \mathbf{w} = \sigma_d^2 - \mathbf{p}^T \mathbf{R}^{-1} \mathbf{p} \]

5.5.1

\[ r_{yy}(k) = r_{xx}(k) + r_{vv}(k) \Rightarrow r_{yy}(0) = r_{xx}(0) + r_{vv}(0); r_{yy}(1) = r_{xx}(1) + r_{vv}(1); \]
\[ r_{yy}(-1) = r_{xx}(-1) + r_{vv}(-1) \text{ but } r_{yy}(1) = r_{yy}(-1), r_{xx}(1) = r_{xx}(-1), r_{vv}(1) = r_{vv}(-1) \Rightarrow \]
\[ \begin{bmatrix} r_{yy}(0) & r_{yy}(1) \\
0 & r_{yy}(0) \end{bmatrix} = \begin{bmatrix} r_{xx}(0) + r_{vv}(0) & r_{xx}(1) + r_{vv}(1) \\
0 & r_{xx}(0) + r_{vv}(0) \end{bmatrix} \]
\[ \begin{bmatrix} r_{xx}(0) + \sigma_v^2 & r_{xx}(1) + 0 \\
0 & r_{xx}(0) + \sigma_v^2 \end{bmatrix} = \begin{bmatrix} r_{xx}(0) & r_{xx}(1) \\
r_{xx}(1) & r_{xx}(0) \end{bmatrix} + \begin{bmatrix} \sigma_v^2 & 0 \\
0 & \sigma_v^2 \end{bmatrix} = \mathbf{R}_x + \sigma_v^2 \begin{bmatrix} 1 & 0 \\
0 & 1 \end{bmatrix} \]

ADDITIONAL PROBLEMS

1. Using MATLAB, we obtain
   \[ \text{inv}(\mathbf{R}) = \begin{bmatrix} 1.3333 & -0.6667 \\
-0.6667 & 1.3333 \end{bmatrix}, \]
b. \( w_o = \text{inv}(R) \ast p = [0.4; 0]; [Q, L] = \text{eig}(R); \)
\[
Q = [-0.7071 \quad 0.7071; 0.7071 \quad 0.7071], \quad L = [0.5 \quad 0; 0 \quad 1.5];
\]
c. \( w = [(1 / 0.5) \ast Q(:, 1) \ast Q(:, 1) + (1 / 1.5) \ast Q(:, 2) \ast Q(:, 2)] \ast p = [0.4000; 0]. \)

\[
J_{\text{min}} = \sigma_d^2 - p^H R^{-1} p = \sigma_d^2 - p^H \left( Q \Lambda Q^H \right)^{-1} p = \sigma_d^2 - p^H \left( \sum_{k=1}^{M} \lambda_k q_k q_k^H \right)^{-1} p
\]

\[
= \sigma_d^2 - p^H \sum_{k=1}^{M} \frac{1}{\lambda_k} q_k q_k^H p = \sigma_d^2 - \sum_{k=1}^{M} \frac{1}{\lambda_k} p^H q_k q_k^H p = \sigma_d^2 - \sum_{k=1}^{M} \frac{1}{\lambda_k} \left| p^H q_k \right|^2
\]

5.5.2 Let the noise signal \( \{x(n)\} \) to enter the Wiener filter. Then the output is \( \hat{x}(n + m) = \sum_{k=0}^{p-1} w(k)x(n-k) \). The desired signal is \( d(n) = x(n + m) \). Therefore, \( r_{dx}(k) = E \{x(n + m)x(n-k)\} = r_{xx}(m + k) \). The Wiener equation, \( R_x w = r_{xx,m} \), becomes

\[
\begin{bmatrix}
  r_{xx}(0) & r_{xx}(1) & \cdots & r_{xx}(p-1) \\
  r_{xx}(1) & r_{xx}(0) & \cdots & r_{xx}(p-2) \\
  \vdots & & \ddots & \vdots \\
  r_{xx}(p-1) & r_{xx}(p-2) & \cdots & r_{xx}(0)
\end{bmatrix}
\begin{bmatrix}
  w(0) \\
  w(1) \\
  \vdots \\
  w(p-1)
\end{bmatrix} = \begin{bmatrix}
  r_{xa}(m) \\
  r_{xa}(m+1) \\
  \vdots \\
  r_{xa}(m+p-1)
\end{bmatrix}
\]

The MMSE is \( J_{\text{min}} = r_{xx}(0) - \sum_{k=0}^{p-1} w(k)x(k + m) = r_{xx}(0) - r_{xx,m}^T w \).

5.5.3

\( p_{dx}(m) = E \{d(n)x(n - m)\} = E \{d(n)d(n - m)\} + E \{d(n)\} E \{v(n - m)\} \)

\( = r_{dx}(m) \)

(1)

since \( d(n) \) and \( v(n) \) are independent and WSS. Also, \( r_{xa}(m) = E \{x(n)x(n - m)\} = r_{dx}(m) + r_{xx}(m) \), where again the independence and the zero mean property of the noise were introduced. Therefore, the Wiener equation becomes \( R_x + R_v = p_{dx} \). From the relation \( p_{dx}(m) = r_{dx}(m) \) [see (1)] and the given relations in the problem, we find that the Wiener equation and its inverse are given by

\[
\begin{bmatrix}
  2 & 0.9 \\
  0.9 & 2
\end{bmatrix}
\begin{bmatrix}
  w_0 \\
  w_1
\end{bmatrix} = \begin{bmatrix}
  1 \\
  0.9
\end{bmatrix},
\]
\[
\begin{bmatrix}
  w_0 \\
  w_1
\end{bmatrix} = \frac{1}{3.19} \begin{bmatrix}
  1.19 \\
  0.9
\end{bmatrix} = \begin{bmatrix}
  0.373 \\
  0.282
\end{bmatrix}
\]
The minimum error is \( J_{\text{min}} = r_{dd}(0) - p_{ds}^T R_s^{-1} w^o = r_{dd}(0) - p_{ds}^T w^o = 0.373 \).
Since \( r_{dd}(0) = \sigma_s^2 = 1 \) and \( \sigma_n^2 = 1 \), the power of the desired signal and noise are equal, and hence, \( 10\log(1/1) = 0 \). After filtering \( x(n) \), we find that the signal power is

\[
E[\hat{d}^2(0)] = w^o^T R_s w^o = \begin{bmatrix} w_0^o & w_1^o \end{bmatrix} \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix} \begin{bmatrix} w_0^o \\ w_1^o \end{bmatrix} = 0.408
\]

and the noise power is

\[
E[\hat{v}^2(0)] = w^o^T R_s w^o = \begin{bmatrix} w_0^o & w_1^o \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_0^o \\ w_1^o \end{bmatrix} = 0.285
\]

Therefore, the signal-to-noise ratio \( \text{SNR} = 10\log(0.408/0.285) = 1.56 \), which means that the Wiener filter increases the SNR by 1.56 dB.

5.5.4

a. \( J = E \{ \hat{d}(n) - [s(n) + w_0s(n-1)] \}^2 \)

\[
= E \{ d^2(n) - [s(n) + w_0s(n-1)]^2 - d(n)[s(n) + w_0s(n-1)] \}
= E \{ d^2(n) \} - E \{ s^2(n) \} - w_0^2 E \{ s^2(n-1) \} - 2w_0 E \{ s(n)s(n-1) \}
- 2E \{ d(n)s(n) \} - 2w_0 E \{ s(n)s(n-1) \}
= 3 - 0.9 - w_0^2 - 2w_0 0.4 + 2 \times 0.5 - 2w_0 0.4 = 3.1 - 0.9w_0^2 - 0.6w_0
\]

\[
\Rightarrow \frac{\partial J}{\partial w_0} = -0.9 \times 2w_0 - 1.6 = 0 \Rightarrow w_0 = 1.125
\]

b. Similarly as (a).

5.5.5

a. \( J = w^T R w - 2w^T p + E \{ d^2(n) \} \)

\[
= \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} - 2 \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} + 2
(1)
\]

\[
= w_0^2 + w_1^2 + 1.2w_0w_1 - 2w_0 - w_1 + 2
\]

\[
\frac{\partial J}{\partial w_0} = 0 \Rightarrow 2w_0^o + 1.2w_1^o = 2
(2)
\]

\[
\frac{\partial J}{\partial w_1} = 0 \Rightarrow 2w_1^o + 1.2w_0^o = 1
(3)
\]
Solving (2) and (3), we obtain
\[ w_0^o = 1.0938; w_1^o = -0.1563 \] (4)

b. Substituting (4) in (1), we get \( J_{\text{min}} = 0.9844 \).

5.5.6

a. \[ J = w^T R w - 2 w^T p + E[d^2(n)] \]

\[
\begin{bmatrix} w_0 & w_1 & w_2 \\
0.25 & 1 & 0.25 \\
0.125 & 0.25 & 1
\end{bmatrix}
\begin{bmatrix} w_0 \\
w_1 \\
w_2
\end{bmatrix} = 2\begin{bmatrix} w_0 \\
w_1 \\
w_2
\end{bmatrix} \begin{bmatrix} 0.5 \\
1 \end{bmatrix} + 5
\]

\[ = w_0^2 + w_1^2 + w_2^2 + 0.5w_0w_1 + 0.25w_0w_2 + 0.5w_1w_2 - 3w_0 - w_1 + 5 \] (1)

\[
\frac{\partial J}{\partial w_0} = 0 \Rightarrow 2w_0^o + 0.5w_1^o + 0.25w_2^o = 3
\]

(2)

\[
\frac{\partial J}{\partial w_1} = 0 \Rightarrow 2w_1^o + 0.5w_0^o + 0.5w_2^o = 1
\]

(3)

\[
\frac{\partial J}{\partial w_2} = 0 \Rightarrow 2w_2^o + 0.25w_0^o + 0.5w_1^o = 0
\]

(4)

b. Substituting (4) in (1), we get \( J_{\text{min}} = 0.3661 \).
6 Eigenvalues of $R_x$

Properties of the Error Surface

6.1 THE EIGENVALUES OF THE CORRELATION MATRIX

Let $R_x$ be an $M \times M$ correlation matrix of a wide-sense (or weakly) stationary (WSS) discrete-time process computed from a data vector $x(n)$. This symmetric and non-negative matrix (see Problem 6.1.1) can be found as follows: We find the data vector $x(n)$ using the MATLAB command: $x = \text{randn}(1, 64)$. From this data vector, we first find its autocorrelation function $r_{xx}(m) = [1.0414 \quad 0.0136 \quad 0.0823]$ using the Book m-function: $r_x = \text{lms\_sample\_biased\_autoc}(x, 3)$. We can also use the Book m-function: $r_x = \text{lms\_sample\_biased\_cross\_cor}(x, x, 3)$. Next, we obtain the correlation matrix $R_x$:

$$
R_x = \begin{bmatrix}
1.0414 & 0.0136 & 0.0823 \\
0.0136 & 1.0414 & 0.0136 \\
0.0823 & 0.0136 & 1.0414
\end{bmatrix}
$$

The above matrix was obtained using the MATLAB function: $R_x = \text{toeplitz}(r_x)$. Having found the $M \times M$ correlation matrix, we desire to find an $M \times 1$ vector $q$ such that

$$
R_x q = \lambda q
$$

for some constant $\lambda$. The above equation indicates that the left-hand-side transformation does not change the direction of the vector $q$ but only its length. The characteristic equation of $R_x$ is given by

$$
\det(R_x q - \lambda I) = 0
$$

(6.2)

The roots $\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_M$ of the above equation are called eigenvalues of $R_x$. Therefore, to each eigenvalue corresponds an eigenvector $q_i$ such that

$$
R_x q_i = \lambda_i q_i, \quad i = 1, 2, \ldots, M
$$

(6.3)
The MATLAB function

\[
[Q, D] = \text{eig}(R)
\]  

(6.4)
gives the diagonal matrix \( D (D = \Lambda) \) containing the eigenvalues of \( R (R = R_x) \) and gives the matrix \( Q \) with its columns equal to the eigenvectors of \( R \). Using the \( 3 \times 3 \) matrix in (6.4), we obtain

\[
Q = \begin{bmatrix}
0.7071 & 0.1531 & 0.6903 \\
-0.0000 & -0.9763 & 0.2165 \\
-0.7071 & 0.1531 & 0.6903
\end{bmatrix}, \quad D = \begin{bmatrix}
0.9592 & 0 & 0 \\
0 & 1.0372 & 0 \\
0 & 0 & 1.1280
\end{bmatrix}
\]

In Table 6.1, we give the eigenvalue properties of WSS processes. The reader should also refer to Section 2.6 for more details on eigenvalue properties. One of the properties is that any two different columns of the \( Q \) matrix are orthogonal. Hence, we write

\[
\gg Q(:,1)'*Q(:,3) = 3.6115e-016; \ %Q(:,1)' = \text{row vector of the} \\
\ %\text{first column of } Q, Q(:,3) = \text{third column of } Q, \\
\ %\text{the result verify the orthogonality property} \\
\ %\text{of the columns};
\]

One of the properties is

\[
Q*D*Q^T = \begin{bmatrix}
1.0414 & 0.0136 & 0.0823 \\
0.0136 & 1.0414 & 0.0136 \\
0.0823 & 0.0136 & 1.0414
\end{bmatrix} = R_x
\]

as it should be.

### 6.1.1 Karhunen–Loeve Transformation

If the observed vector \( x(n) \) is an observation vector with the correlation \( R_x \). If \( Q \) is the matrix as defined earlier, then the elements of the vector

\[
x^{(y)}(n) = Q^H x(n)
\]

(6.5)

constitute a set of uncorrelated random variables. The above transformation is called the Karhunen–Loeve transform.
Using (6.5), we obtain

\[ E \{ x^{(kv)}(n) x^{(kv)H}(n) \} = Q^H E \{ x(n) x^H(n) \} Q = Q^H R_x Q = Q^H \Lambda Q^H = \Lambda \quad (6.6) \]

We have assumed that the eigenvectors were normalized to unit length. Because \( \Lambda \) is diagonal, it is apparent that the elements of \( x^{(kv)}(n) \) are uncorrelated. Premultiply (6.5) by \( Q \) and use the relation \( QQ^H = I \) to find

\[ x(n) = \sum_{i=0}^{N-1} \lambda_i^{(k)}(n) q_i \quad (6.7) \]

The above expansion is known as the Karhunen–Loeve expansion.
Suppose that a white noise with zero mean and variance one has passed a filter which for $a = 0$ has $H(z) = 1$, an all-pass filter. The output of this filter is the vector $\{x(n)\}$ with the correlation function $r_{xx}(k) = a^k$ for $k = 0, 1, 2, \ldots, 255$. Figure 6.1 shows the power spectral density of the output signal $\{x(n)\}$ for different values of $a$. It shows that as $a$ approaches 0, the output is white noise since its spectrum becomes flat.

6.2 GEOMETRICAL PROPERTIES OF THE ERROR SURFACE

The performance function of a transversal [finite impulse response (FIR)] Wiener filter with a real-valued input sequence $\{x(n)\}$ and a desired output sequence $\{d(n)\}$ is

\[
J = w^T R w - 2 p^T w + E\{d^2(n)\}
\]

where \( w = [w_0 \quad w_1 \quad \cdots \quad w_{M-1}] \), \( R_x = E\{x(n)x^T(n)\} \),

\[
x = [x(n) \quad x(n-1) \quad \cdots \quad x(n-M+1)]
\]

\[
p = E\{d(n)x(n)\}
\]

The optimum Wiener filter coefficients are given by

\[
R_x w^o = p
\]
Using the optimum solution of the Wiener filter from the above equation, and since \( w^T p = p^T w = \text{constant} \) and \( R_s^T = R_s \), the performance function in (6.8) becomes

\[
J = w^T R_s w - w^T R_s w^o - w_o^o R_s^T w + w_o^o R_s w^o + E[d^2(n)] - w_o^o R_s w^o
\]

\[
= (w - w^o)^T R_s (w - w^o) + E[d^2(n)] - w_o^o R_s w^o
\]

\[
= (w - w^o)^T R_s (w - w^o) + J_{\text{min}}
\]

\[
= J_{\text{min}} + (w - w^o)^T R_s (w - w^o)
\]

(6.10)

The cost function (performance function) can be written in the form:

\[
w^T R_s w - 2 p^T w - (J - E[d^2(n)]) = 0 \quad \text{or} \quad w^T R_s w - 2 p^T w - (J - \sigma_d^2) = 0 \quad (6.11)
\]

If we set the values of \( J > J_{\text{min}} \), the \( w \)-plane will cut the second-order surface, along a line whose projection on the \( w \)-plane is ellipses arbitrarily oriented as shown in Figure 6.2. The contours were found using the following Book MATLAB program.

FIGURE 6.2
Adaptive Filtering

Book MATLAB Program

```matlab
>>w0 = -3:0.05:3;w1 = -3:0.05:3;
>>[x,y] = meshgrid(w0,w1);%MATLAB function;
>>j=0.8461+0.972*x-0.773*y+1.0647*y.^2+1.064*y.*y+0.985*x.*y;
>>contours(x,y,j,30);%MATLAB function,30 is the number of
%contours;
```

If we set `contour(j,[2.3 3.1 5])`, we will produce three contours at heights 2.3, 3.1, and 5. The first simplification we can do is to shift the origin of the $w$-axis to another one whose origin is on the center of the ellipses. This is accomplished using the transformation: $\xi = w - w^o$. If we introduce this transformation in (6.11) and set $J = 2J_{\text{min}}$, we obtain the relationship (see Problem 6.2.1):

$$\xi^T R_x \xi - 2J_{\text{min}} + \sigma_d^2 - p^T w^o = 0$$

(6.12)

But $\sigma_d^2 - p^T w^o = J_{\text{min}}$, and thus, the above equation becomes

$$\xi^T R_x \xi = J_{\text{min}}$$

(6.13)

The matrix is a diagonal matrix whose elements are the eigenvalues of the correlation matrix $R_x$.

**Example 6.2.1**

Let $\lambda_1 = 1$, $\lambda_2 = 0.5$, and $J_{\text{min}} = 0.67$. The ellipse in the $(\xi_0', \xi_1')$ plane is found by solving the system:

$$\begin{bmatrix} \xi_0' & \xi_1' \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} \xi_0' \\ \xi_1' \end{bmatrix} = 0.67$$

or

$$\left( \frac{\xi_0'}{\sqrt{0.67/1}} \right)^2 + \left( \frac{\xi_1'}{\sqrt{0.67/0.5}} \right)^2 = 1$$

(6.14)

where:

- $0.67/0.5$ is the major axis
- $0.67/1$ is the minor axis

Hence, for the case $J = 2J_{\text{min}}$, the ellipse intersects the $\xi_0'$-axis at 0.67 and the $\xi_1'$-axis at 2.68. To find, for example, the intersection between the $\xi_1'$-axis and the ellipse, we must set $\xi_0'$ equal to zero because the projection on the $\xi_1'$-axis of the intersection point is at $\xi_0' = 0$.

If we start with (6.11) and apply the shift and rotation transformations, we obtain the relationships:

$$J = J_{\text{min}} + (w - w^o)^T R_x (w - w^o) = J_{\text{min}} + \xi^T R_x \xi$$

$$J = J_{\text{min}} + \xi^T (Q \Lambda Q^T) \xi = J_{\text{min}} + \xi^T \Lambda \xi = J_{\text{min}} + \sum_{i=0}^{M-1} \lambda_i \xi_i^2$$

(6.15)
Note:

- The contours intersect the $\xi'$-axes at values depending on the eigenvalues of $R_x$ and the specific mean-square error (MSE) value chosen. The rotation and translation do not alter the shape of the MSE surface.
- If the successive contours for the values $2J_{\text{min}}$, $3J_{\text{min}}$, $\ldots$, are close to each other, the surface is steep, which in turn indicates that the mean-square estimation error is very sensitive to the choice of the filter coefficients.
- Choosing the filter values $w$ is equivalent to choosing a point in the $w$-plane. The height of the MSE surface above the plane at that point is determined only by the signal correlation properties.

Let rearrange (6.15) in the form:

$$ J - J_{\text{min}} = \xi' A \xi' = \begin{bmatrix} \xi_0' & \xi_1' \end{bmatrix} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \lambda_1 \end{bmatrix} \begin{bmatrix} \xi_0' \\ \xi_1' \end{bmatrix} = \lambda_0 \xi_0'^2 + \lambda_1 \xi_1'^2 \quad (6.16) $$

Next, we plot the contours of the performance surface (see Figure 6.3) for two different ratios of the eigenvalues. The contours with shapes closer to a circle were produced by the ratio $0.8/0.6 = 1.333$ and the most elongated ones by the ratio $1.2/0.2 = 6$. Instead of $\xi_i'$, we renamed them to $\xi_i$ for the plotting purpose.

![Figure 6.3](image-url)
PROBLEMS

6.1.1 Prove that the correlation matrix of a WSS process is positive definite, that is, \( a^T R a \geq 0 \) for any vector \( a \).

6.1.2 Verify the property \( \text{tr}(R_x) = \sum_{i=0}^{M-1} \lambda_i \), where \( \text{tr}(A) \) stands for the trace of the matrix \( A \), which is equal to the sum of the main diagonal.

6.2.1 Verify (6.12).

6.2.2 Find the contours of the performance surface if the following data are given:

\[
R_x = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \quad p_{dk} = \begin{bmatrix} 6 \\ 7 \end{bmatrix}, \quad \sigma_d^2 = 28
\]

6.2.3 A Wiener filter is characterized by the following parameters:

\[
R = \begin{bmatrix} d & a \\ a & d \end{bmatrix}, \quad p = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \sigma_d^2 = 2
\]

It is requested to explore the performance surface as the ratio \( \lambda_0/\lambda_1 \) varies. This is accomplished using different values for \( a \) and \( b \).

6.2.4 Consider the performance function: \( J_{\text{min}} = w_0^2 + w_1^2 + w_0w_1 - w_0 + w_1 + 2 \).

(a) Convert this equation to its canonical form and (b) plot the set of contour ellipses of the performance surface of \( J \) for values of \( J = 1, 2, 3, \) and \( 4 \).

6.2.5 If \( R \) is a correlation matrix, show that (a) \( R^n = Q \Lambda^n Q^T \). (b) If \( R^{1/2} R^{1/2} = R \), show that \( R^{1/2} = Q \Lambda^{1/2} Q^T \).

HINTS–SOLUTIONS–SUGGESTIONS

6.1.1

\[
a^T R a \geq 0 = a^T E[xx^T] a = E\{ (a^T x)(x^T a) \} = E\{ (a^T x)^2 \} \geq 0, \quad a^T x = x^T a
\]

\[
= \text{Number}
\]

6.1.2 \( \text{tr}(R_x) = \text{tr}(Q \Lambda Q^T) = \text{tr}(Q^T Q \Lambda) = \text{tr}(I \Lambda) = \text{tr}(\Lambda), \quad \text{tr}(AB) = \text{tr}(BA) \); for Hermitian matrices, \( T \) changes to \( H \).

6.2.1

\[
(w - w^o + w^o)^T R(w - w^o + w^o) - 2 p(w - w^o + w^o) - 2 J_{\text{min}} + \sigma_d^2
\]

\[
= (\xi + w^o)^T R(\xi + w^o) - 2 p^T (\xi + w^o) - 2 J_{\text{min}} + \sigma_d^2
\]

With \( Rw^o = p, w^o p = p^T w^o = \text{Number}, \) and \( \sigma_d^2 - p^T w^o = J_{\text{min}}, \) the equation is easily proved.
6.2.2

\[ J = J_{\min} + \xi' \Lambda \xi \] and \( J_{\min} = \sigma_d^2 - p^T w^o \),

\[
|R - \lambda I| = \begin{vmatrix} 2 - \lambda & 1 \\ 1 & 3 - \lambda \end{vmatrix} = (2 - \lambda)(3 - \lambda) - 1 = 0
\]

\[ \lambda_1 = 1.382, \quad \lambda_2 = 3.618 \]

\[ \begin{bmatrix} w_0^o \\ w_1^o \end{bmatrix} = \begin{bmatrix} 11/5 \\ 8/5 \end{bmatrix}, \quad J_{\min} = 28 - [6 \quad 7] \begin{bmatrix} 11/5 \\ 8/5 \end{bmatrix} = 3.6 \]

\[ J = 3.6 + [\xi_0, \xi_1'] \begin{bmatrix} 1.328 & 0 \\ 0 & 3.618 \end{bmatrix} [\xi_0, \xi_1'] = 3.6 + 1.382\xi_0 + 3.618\xi_1 \]

6.2.3 The MSE surface is given by

\[ J = 2 - 2[w_0 \quad w_1] \begin{bmatrix} 1 \\ 1 \end{bmatrix} + [w_0 \quad w_1] \begin{bmatrix} d & a \\ a & d \end{bmatrix} [w_0 \quad w_1], \]

and the optimum tap weights are

\[ \begin{bmatrix} w_0^o \\ w_1^o \end{bmatrix} = R^{-1} p = \begin{bmatrix} d & a \\ a & d \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d + a \\ 1/d + a \end{bmatrix}. \]

Therefore, the minimum MSE surface is

\[ J_{\min} = \sigma_d^2 - w^{oT} p = 2 - \left[ \frac{1}{d + a} \quad \frac{1}{d + a} \right] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 \left[ \frac{d + a - 1}{d + a} \right]. \]

We also have the relation

\[ J = J_{\min} + (w - w^o)^T R(w - w^o) = 2 \left[ \frac{d + a + 1}{d + a} \right] + [\xi_0, \xi_1'] \begin{bmatrix} d & a \\ a & d \end{bmatrix} [\xi_0, \xi_1']. \]

From the relation

\[ \begin{vmatrix} d - \lambda \\ a \\ d - \lambda \end{vmatrix} = 0, \]

we obtain the two eigenvalues \( \lambda_0 = d + a \) and \( \lambda_1 = d - a \). By incorporating different values for \( a \) and \( d \), we can create different ratios of the eigenvalues. The larger the ratio, the more elongated the ellipses are.
6.2.4

\[ J = \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} - 2 \begin{bmatrix} 1 \\ -1/2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} + 2, \quad E(d^2(n)) = 2, \]

\[ R = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \quad p = \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix} \]

\[ w^o = R^{-1} p = \begin{bmatrix} 1 \\ 0.75 \end{bmatrix} \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix} \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad J_{\text{min}} = 2 - p^T w^o = 2 - 1 = 1, \]

\[ J = 1 + \xi^T \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \xi, \quad |\lambda I - R| = 0 \]

\[ \Rightarrow (\lambda - 1)^2 - 0.25 = 0 \Rightarrow \lambda_0 = 1.5, \lambda_1 = 0.5, \]

Eigenvectors

\[ \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \begin{bmatrix} q_{00} \\ q_{01} \end{bmatrix} = \begin{bmatrix} 1.5 q_{00} \\ q_{01} \end{bmatrix} \]

\[ \Rightarrow q_{00} + 0.5q_{01} = 1.5q_{00} \Rightarrow q_{00} = q_{01} \quad (\text{also} \quad 0.5q_{00} + q_{01} = 1.5q_{00}) \]

\[ \Rightarrow q_{00} = q_{01} \Rightarrow \text{normalized} \quad q_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}; \quad \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \begin{bmatrix} q_{10} \\ q_{11} \end{bmatrix} = 0.5 \begin{bmatrix} q_{10} \\ q_{11} \end{bmatrix} \]

\[ \Rightarrow q_{10} = -q_{11} \Rightarrow \text{normalized} \quad q_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}; \]

Eigenvector matrices are

\[ \Lambda = \begin{bmatrix} 1.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad Q = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \]

\[ J = 1 + \xi^T Q \Lambda Q^T \xi = 1 + \xi^T \begin{bmatrix} 1.5 & 0 \\ 0 & 0.5 \end{bmatrix} \xi' \]

\[ = 1 + 1.5\xi_0'^2 + 0.5\xi_1'^2 = 1 + \left( \frac{\xi_0'}{\sqrt{1.5}} \right)^2 + \left( \frac{\xi_1'}{\sqrt{0.5}} \right)^2 \]
Eigenvalues of $R_x$

We can use the following Book MATLAB program to produce four contours at different heights of the performance surface.

\[
\text{>> } x = -5:0.1:5; \quad y = -5:0.1:5; \\
\text{>> } [X,Y]=\text{meshgrid}(x,y); \\
\text{>> } Z=1+1.5*X.^2+0.5*Y.^2; \\
\text{>> } V=[1.2 \ 2 \ 3 \ 4]; \quad \text{if we set 1 instead of 1.2, the first} \\
\quad \text{circle will be a dot since this indicates one point} \\
\quad \text{of the bottom of the surface, as it should be;} \\
\text{>> } \text{contour}(X,Y,Z,V,'k'); \quad \text{the circles will be black;}
\]

6.2.5

(a) \( R^n = Q \Lambda^n Q^T Q \Lambda^n Q^T \cdots Q \Lambda^n Q^T = Q \Lambda^n Q^T \), because \( Q^T Q = I = QQ^T \)

(b) Let \( P = Q^T R^{1/2} Q \Rightarrow R^{1/2} = QPQ^T \Rightarrow R^{1/2} R^{1/2} = R \)

\[ \Rightarrow QPQ^T QPQ^T = Q \Lambda Q^T \Rightarrow QP^2 Q^T = Q \Lambda Q^T \]

\[ \Rightarrow Q^T QP^2 Q^T \quad \Rightarrow \quad Q^T Q \Lambda Q^T \quad \Rightarrow \quad P^2 = \Lambda \]

\[ \Rightarrow P = \Lambda^{1/2} \Rightarrow R^{1/2} = Q \Lambda^{1/2} Q^T \]
7 Newton’s and Steepest Descent Methods

7.1 ONE-DIMENSIONAL GRADIENT SEARCH METHOD

In general, we can say that adaptive algorithms are nothing but iterative search algorithms derived from minimizing a cost function with the true statistics replaced by their estimates. To study the adaptive algorithms, it is necessary to have a thorough understanding of the iterative algorithms and their convergence properties. In this chapter, we discuss the steepest descent and the Newton’s method.

The one-coefficient mean-square error (MSE) surface (line in this case) is given by [see (6.15)]

\[ J(w) = J_{\text{min}} + r_{xx}(0)(w - w^o)^2; \quad J(w^o) \leq J(w) \quad \text{for all } w \text{'s} \]  

and it is pictorially shown in Figure 7.1. The first and second derivatives are

\[ \frac{\partial J(w)}{\partial w} = 2r_{xx}(0)(w - w^o) \]  

(7.2a)

\[ \frac{\partial^2 J(w)}{\partial w^2} = 2r_{xx}(0) > 0 \]  

(7.2b)

At \( w = w^o \), the first derivative is zero and the second derivative is positive and greater than zero, which indicates that the surface has a global minimum and is concave upward. To find the optimum value of \( w \), we can use an iterative approach. We start first with an arbitrary value \( w(0) \) and measure the slope of the curve \( J(w) \) at \( w(0) \). Next, we find \( w(1) \) to be equal to \( w(0) \) and the negative of an increment proportional to the slope of \( J(w) \) at \( w(0) \). Proceeding with the iteration procedure, we will eventually find the minimum value \( w^o \). The values \( w(0), w(1), \ldots \), are known as the gradient estimates.

7.1.1 GRADIENT SEARCH ALGORITHM

Based on the above development in Section 7.1, the filter coefficient at iteration \( n, w(n) \), is found using the relation:

\[ w(n+1) = w(n) + \mu \left[ -\frac{\partial J(w)}{\partial w} \right]_{w=w(n)} \]

(7.3)

\[ = w(n) + \mu \{ -\nabla J[w(n)] \} = w(n) - 2\mu r_{xx}(0)[w(n) - w^o] \]
where: 
\( \mu \) is a constant to be determined

Rearranging (7.3), we obtain

\[
\begin{align*}
\begin{bmatrix}
\text{New coefficient} \\
\text{Old coefficient}
\end{bmatrix} &= 
\begin{bmatrix}
\mu \\
1
\end{bmatrix}
\begin{bmatrix}
\text{Adaptation} \\
\text{gain}
\end{bmatrix}
\end{align*}
\]

\( w(n + 1) = [1 - 2\mu r_{xx}(0)]w(n) + 2\mu r_{xx}(0)w^o \) (7.4)

The solution of the above difference equation, using the iteration approach (see Problem 7.1.1), is

\[
\begin{align*}
w(n) &= w^o + [1 - 2\mu r_{xx}(0)]^n [w(0) - w^o] \\
\end{align*}
\]

\( w(n + 1) = aw(n) + bw^o \) (Difference equation) (7.5)

The above equation gives \( w(n) \) explicitly at any iteration \( n \) in the search procedure. This is the solution to the gradient search algorithm. Note that if we had initially guessed \( w(0) = w^o \), which is the optimum value, we would have found \( w(1) = w^o \), which gives the optimum value in one step.

To have convergence of \( w(n) \) in (7.5), we must impose the condition:

\[
|1 - 2\mu r_{xx}(0)| < 1
\]

(7.6)
Newton’s and Steepest Descent Methods

The above inequality defines the range of the step-size constant $\mu$ so that the algorithm will converge. Hence, we obtain

$$-1 < 1 - 2\mu r_{xx}(0) < 1 \quad \text{or} \quad 0 < 2\mu r_{xx}(0) < 2 \quad \text{or} \quad 0 < \mu < \frac{1}{r_{xx}(0)} \quad \text{(7.7)}$$

Under the above condition, (7.5) converges to the optimum value $w^o$ as $n \to \infty$. If $\mu > 1/r_{xx}(0)$, the process is unstable and no convergence takes place.

When the filter coefficient has a value $w(n)$ (i.e., at iteration $n$), then the MSE surface (here a line) is [see (7.1)]

$$J(n) = J_{\min} + r_{xx}(0)[w(n) - w^o]^2 \quad \text{(7.8)}$$

Substituting the quantity $w(n) - w^o$ in (7.8) from (7.5), we obtain

$$J(n) = J_{\min} + r_{xx}(0)[w(0) - w^o]^2[1 - 2\mu r_{xx}(0)]^{2n} \quad \text{(7.9)}$$

The above two equations show that $w(n) - w^o$ as $n$ increases to infinity, and the MSE undergoes a geometric progression toward $J_{\min}$. The plot of the performance surface $J(n)$ versus the iteration number $n$ is known as the **learning curve**.

### 7.1.2 Newton’s Method in Gradient Search

The Newton’s method finds the solution (zeros) of the equation $f(w) = 0$. From Figure 7.2, we observe that the slope at $w(0)$ is

$$f'(w(0)) = \frac{df(w)}{dw} \bigg|_{w=w(0)} \approx \frac{f[w(0)]}{w(0) - w(1)} \quad \text{(7.10)}$$

where:

- $w(0)$ is the initial guessing.

![Figure 7.2](image-url)
The above equation is the result of retaining the first two terms of Taylor’s expansion and setting the rest of them equal to zero. This equation can be written in the form:

\[ w(1) = w(0) - \frac{f[w(0)]}{f'[w(0)]} \]  

(7.11)

From (7.11), it is clear that if we know the values of the function and its derivative at the same point, in this case at \( n = 0 \), we find \( w(1) \). Hence, the \( n \)th iteration of the above equation takes the form:

\[ w(n + 1) = w(n) - \frac{f[w(n)]}{f'[w(n)')} = w(n) - \frac{f[w(n)][w(n) - w(n - 1)]}{f[w(n)] - f[w(n - 1)]} \]

(7.12)

But we have \( f'[w(n)] \cong [f[w(n)] - f[w(n - 1)]]/[w(n) - w(n - 1)] \), and hence, (7.12) becomes

\[ w(n + 1) = w(n) - \frac{\partial J[w(n)]}{\partial w(n)} \]

(7.13)

As we have mentioned earlier, the Newton’s method finds the roots of the function \( f(w) \), which solves the polynomial \( f(w) = 0 \). However, in our case we need to find the minimum point of the performance surface (here line). This is equivalent to setting \( \partial J(w)/\partial w = 0 \). Therefore, we substitute the derivative \( \partial J(w)/\partial w \) for \( f(w) \) in (7.12) and the second-order derivative \( \partial^2 J(w)/\partial w^2 \) for \( f'(w) \). Hence, (7.13) becomes

\[ w(n + 1) = w(n) - \frac{\partial J[w(n)]}{\partial w(n)} \]

(7.14)

### 7.2 STEEPEST DESCENT ALGORITHM

To find the minimum value of the MSE surface, \( J_{\text{min}} \), using the steepest descent algorithm, we proceed as follows: (1) We start with the initial value \( w(0) \), usually using the null vector; (2) at the MSE surface point that corresponds to \( w(0) \), we compute the gradient vector, \( \nabla J[w(0)] \); (3) we compute the value \( -\mu \nabla J[w(0)] \) and add it to \( w(0) \) to obtain \( w(1) \); (4) we go back to step (2) and continue the procedure till we find the optimum value of the vector coefficient, \( w^* \).

If \( w(n) \) is the filter coefficient vector at step \( n \) (time), then its updated value \( w(n + 1) \) is given by [see also (7.3); it is the first order of Taylor’s expansion]

\[ w(n + 1) = w(n) - \mu \nabla w J[w(n)] \]

(7.15)

\[ \nabla J[w(n)] = \begin{bmatrix} \partial J[w(n)] \\ \partial w_0 \\ \partial J[w(n)] \\ \partial w_1 \\ \vdots \\ \partial J[w(n)] \\ \partial w_{M-1} \end{bmatrix} \]

\[ w(n) = [w_0 \ w_1 \ w_2 \ldots w_{M-1}]^T \]
7.2.1 Steepest Descent Algorithm Applied to Wiener Filter

The cost function $J[w(n)]$ at time $n$ is a quadratic function of the Wiener coefficients. Hence, we write

$$
J[w(n)] = \sigma_d^2 - w^T(n)p - p^T w(n) + w^T(n)R \cdot w(n)
$$

where:
- $\sigma_d^2$ is the variance of the desired response $d(n)$
- $p$ is the cross-correlation vector between the Wiener coefficients vector $x(n)$ and the desired response $d(n)$
- $R_x$ is the correlation matrix of the coefficient vector $x(n)$

Using (7.15) and Section A2.6, we obtain

$$\nabla_w J[w(n)] = -2p + 2R_x \cdot w(n)$$

Example 7.2.1

Verify (7.17) with a two-length vector.

Solution:

$$\nabla_w \{J(w(n))\} = \nabla_w \{\sigma_d^2\} - \nabla_w \{w^T(n)p\} - \nabla_w \{p^T w(n)\} + \nabla_w \{w^T(n)R \cdot w(n)\}$$

Hence, we obtain

$$\nabla_w \{\sigma_d^2\} = 0, \quad -\nabla_w \{w^T \cdot p\} = - \begin{bmatrix}
\frac{\partial(w_0 p_0 + w_1 p_1)}{\partial w_0} \\
\frac{\partial(w_0 p_0 + w_1 p_1)}{\partial w_1}
\end{bmatrix} = - \begin{bmatrix}
p_0 \\
p_1
\end{bmatrix} = -p$$

$$\nabla_w \{p^T \cdot w\} = \nabla_w \left[ \begin{array}{cc}
r_{xx}(0) & r_{xx}(1) \\
r_{xx}(1) & r_{xx}(0)
\end{array} \right] \begin{bmatrix}
w_0 \\
w_1
\end{bmatrix}$$

$$\nabla_w \{w^T \cdot R \cdot w\} = \nabla_w \left[ \begin{array}{cc}
r_{xx}(0) & r_{xx}(1) \\
r_{xx}(1) & r_{xx}(0)
\end{array} \right] \begin{bmatrix}
w_0 \\
w_1
\end{bmatrix} = \begin{bmatrix}
w_0^2 r_{xx}(0) + w_0 w_1 r_{xx}(1) + w_0 r_{xx}(1) + w_1 r_{xx}(1) \\
2w_1 r_{xx}(0) + 2w_1 r_{xx}(1) + w_1^2 r_{xx}(1)
\end{bmatrix}$$

$$= \begin{bmatrix}
w_0^2 r_{xx}(0) + w_0 w_1 r_{xx}(1) + w_0 r_{xx}(1) + w_1 r_{xx}(1) \\
2w_1 r_{xx}(0) + 2w_1 r_{xx}(1) + w_1^2 r_{xx}(1)
\end{bmatrix} = 2 \begin{bmatrix}
r_{xx}(0) & r_{xx}(1) \\
r_{xx}(1) & r_{xx}(0)
\end{bmatrix} \begin{bmatrix}
w_0 \\
w_1
\end{bmatrix} = 2R_x \cdot w$$

All vectors are taken at time $n$. 

■
Introducing the results in Example 7.2.1 in (7.15a), we find
\[
\begin{align*}
    w(n+1) &= w(n) + 2\mu \left[ p_{dx} - R_x w(n) \right] = w(n) + \mu' \left[ p_{dx} - R_x w(n) \right] \\
    &= \left[ I - \mu'R_x \right] w(n) + \mu' p_{dx}, \quad \mu' = 2\mu \quad n = 0, 1, 2, \ldots
\end{align*}
\] (7.18)

where:

- \( I \) is the identity matrix

The value of the primed step-size parameter must be much less than 1/2 for convergence. We may view (7.18) as a feedback model, as illustrated by the signal-flow graph in Figure 7.3.

To apply the method of steepest descent, we must first find the estimate of the autocorrelation matrix \( R_x \) and the cross-correlation vector \( p_{dx} \) from the finite data. This is necessary since we do not have in practice an ensemble of data to obtain \( R_x \) and \( p_{dx} \).

### 7.2.2 Stability (Convergence) of the Algorithm

Let
\[
\xi(n) = w(n) - w^o
\] (7.19)

be the difference between the filter coefficient vector and its optimum Wiener value \( w^o \). Next we write the first part of (7.18) in the form:
\[
\begin{align*}
    w(n+1) - w^o &= w(n) - w^o + \mu' \left[ R_x w^o - R_x w(n) \right] \\
    \xi(n+1) &= \left[ I - \mu'R_x \right] \xi(n)
\end{align*}
\] (7.20)
But since \( \mathbf{R}_x = \mathbf{Q} \Lambda \mathbf{Q}^T \) (see Table 6.1.1) and \( \mathbf{I} = \mathbf{QQ}^T \), (7.20) becomes
\[
\xi(n+1) = [\mathbf{I} - \mu' \mathbf{Q} \Lambda \mathbf{Q}^T] \xi(n) \\
\text{or}
\mathbf{Q}^T \xi(n+1) = [\mathbf{I} - \mu' \Lambda] \mathbf{Q}^T \xi(n) \\
\text{or}
\xi'(n+1) = [\mathbf{I} - \mu' \Lambda] \xi'(n)
\]
where the coordinate axis defined by \( \xi' \) is orthogonal to ellipsoids (see Chapter 6) and is a diagonal matrix with its diagonal elements equal to the eigenvalues of \( \mathbf{R}_x \). The \( k \)th row of (7.21), which represents the \( k \)th natural mode of the steepest descent algorithm, is (see Problem 7.2.1)
\[
\xi_k(n+1) = [1 - \mu' \lambda_k] \xi_k(n)
\]
The above equation is a homogeneous difference equation, which has the following solution (see Problem 7.1.1):
\[
\xi_k(n) = (1 - \mu' \lambda_k)^n \xi_k(0) \quad k = 1, 2, \ldots, M - 1, \quad n = 1, 2, 3, \ldots
\]
For the above equation to converge (to be stable) as \( n \to \infty \), we must set
\[
-1 < 1 - \mu' \lambda_k < 1 \quad k = 0, 1, 2, \ldots, M - 1
\]
First, subtract \(-1\) from all the three elements of the inequality. Next, multiply by \(-1\) and reverse the inequalities. Hence, we obtain
\[
0 < \mu' < \frac{2}{\lambda_k} \quad \text{or} \quad 0 < \mu < \frac{1}{\lambda_k} \quad k = 0, 1, 2, \ldots, M - 1
\]
Under the above conditions and as \( n \to \infty \), (7.23) becomes \( \lim_{n \to \infty} \xi_k(n) = 0 \) or \( \xi = 0 \) since \( \mathbf{Q}^T \neq \mathbf{0} \), and thus, \( \mathbf{w}(\infty) = \mathbf{w}^o \). Because (7.23) decays exponentially to zero, there exists a time constant that depends on the value of \( \mu' \) and the eigenvalues of \( \mathbf{R}_x \). Furthermore, (7.23) implies that immaterially of the initial value \( \xi(0) \), \( \mathbf{w}(n) \) always converges to \( \mathbf{w}^o \), provided, of course, that (7.25) is satisfied. This is an important property of the steepest descent algorithm. Since each row of (7.22) must decay as \( n \to \infty \), it is necessary and sufficient that \( \mu' \) obeys the following relationship:
\[
0 < \mu' < \frac{2}{\lambda_{\max}}
\]
Since \((1 - \mu' \lambda_k)^n\) decays exponentially, there exists an exponential function with time constant \( \tau_k \) such that \( e^{-(i/\tau_k)} = (1 - \mu' \lambda_k)^n \) or \( (1 - \mu' \lambda_k) = e^{-1/(\tau_k)} = 1 - (1/\tau_k) + (1/2! \tau_k^2) - \cdots \). Therefore, for small \( \mu' \) and \( \lambda_k \) (larger \( \tau_k \)), we have
\[
\tau_k \approx \frac{1}{\mu' \lambda_k} \quad \mu' \lambda_k \ll 1
\]
In general, the \( k \)th time constant \( \tau_k \) can be expressed in the form:

\[
\tau_k = -\frac{1}{\ln(1 - \mu' \lambda_k)}
\]  

(7.28)

### 7.2.3 Transient Behavior of MSE

Using (6.15) at time \( n \), we obtain the relation:

\[
J[k \mathbf{w}(n)] = J_{\text{min}} + \xi_k^T \Lambda \xi_k' = J_{\text{min}} + \sum_{k=0}^{M-1} \lambda_k \xi_k'^2(n)
\]

(7.29)

Substituting the solution of \( \xi_k'(n) \) from (7.23) in (7.29), we find the relation:

\[
J[k \mathbf{w}(n)] = J_{\text{min}} + \sum_{k=0}^{M-1} \lambda_k (1 - \mu' \lambda_k)^{2n} \xi_k'^2(0)
\]

(7.30)

It is obvious from the above equation (the factor in parentheses is less than one) that

\[
\lim_{n \to \infty} J[k \mathbf{w}(n)] = J_{\text{min}}
\]

(7.31)

From (7.29), we observe that the learning curve [the plot of \( J[k \mathbf{w}(n)] \) vs. \( n \)] consists of a sum of exponentials, each one corresponding to a natural mode of the algorithm.

**Example 7.2.2**

Let us have the following data:

\[
R_x = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \quad p_{\text{dc}} = \begin{bmatrix} 0 \\ 0.294 \end{bmatrix}, \quad w_0(1) = -1.2, \quad w_1(1) = -2.2, \quad \mu = 0.2
\]

(7.32)

It is desired (1) to find the equation of the MSE and plot the contours, and (2) to plot the convergence path of the steepest descent algorithm in the MSE surface.

**Solution:** The MSE function is

\[
J = E\{d^2(k)\} - 2\mathbf{w}^T p_{\text{dc}} + \mathbf{w}^T R_x \mathbf{w}
\]

\[
= \sigma_d^2 - 2\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \begin{bmatrix} 0 \\ 0.294 \end{bmatrix} + \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}
\]

\[
= 0.48 + w_0^2 + w_1^2 + 0.8w_0w_1 - 0.588w_1
\]

The following Book MATLAB program was used to obtain the changes in the filter coefficient:
Newton’s and Steepest Descent Methods

Book MATLAB Program

```matlab
>>w0(1) = -1.2; w1(1) = -2.2; m = 0.2;
>>for k = 1:50
    >> w0(k+1) = w0(k)-2*m*(w0(k)+0.4*w1(k));
    >> w1(k+1) = w1(k)-2*m*(0.4*w0(k)+w1(k))+2*m*0.294;
>>end
>>x=-2:0.2:2; y = -2:0.2:2;
>>[X,Y]=meshgrid(x,y);
>>Z=0.48+X.^2+Y.^2+0.8*X*Y-0.588*Y;
>>contour(X,Y,Z,15,'k');
>>hold on; plot(w0,w1,'k');
```

Figure 7.4 shows the contours and the trajectory of the filter coefficients. The eigenvalue ratio for this case was 1.4/0.6 = 2.3333.

As the eigenvalue ratio increases, the contours become elongated ellipses.

7.2.4 Learning Curve

Accepting a small value of the constant $\mu$, converging condition, the terms under the summation in (7.30) converge to zero as $k$ increases. As a result, the minimum MSE is achieved after a sufficient number of iterations.

The curve obtained by plotting $J$ as a function of the iteration index $k$ is called the learning curve. A learning curve of the steepest descent algorithm, as can be...
seen from (7.30), consists of the sum of $M$ exponentially decaying terms, each of which corresponds to one of the modes of convergence of the algorithm. Assuming $\mu = 0.02$, $\xi(0) = 1$, and the two eigenvalues, 0.1 and 2.4, we obtain the curve shown in Figure 7.5. We observe that the two straight sections in the graph indicate two different time constants (see Problem 7.2.2).

7.3 NEWTON’S METHOD

Using $p_{dx} = R w^\omega$, (7.18) becomes

$$w(n+1) = w(n) - 2\mu R_x (w - w^\omega)$$  \hspace{1cm} (7.33)

The presence of the correlation matrix $R_x$ in the above equation causes the eigenvalue spread problem in the steepest descent algorithm. The Newton’s method overcomes this problem by replacing the scalar step-size parameter $\mu$ with the matrix step size given by $\mu R_x^{-1}$. Using Appendix 2, we obtain from (7.16) the relation:

$$\nabla w J[w(n)] = -2 p_{dx} + R_x w(n)$$  \hspace{1cm} (7.34)

Using the above equation in (7.33), we obtain the equation:

$$w(n+1) = w(n) - 2\mu R_x [w(n) - w^\omega] = w(n) - \mu \nabla w J(w)$$  \hspace{1cm} (7.35)

Replacing $\mu$ with the Newton’s relation $\mu R_x^{-1}$ overcomes the eigenvalue spread, which may produce large values of its elements, and this will produce difficulties.
in solving equations that involve inverse correlation matrices. In such cases, we say that the correlation matrices are **ill-conditioned**. This substitution has the effect of rotating the gradient vector to the direction pointing toward the minimum point of the MSE surface as shown in Figure 7.6.

Substituting (7.34) in (7.35) and the Newton’s relation, we obtain

\[
\mathbf{w}(n + 1) = \mathbf{w}(n) - \mu \mathbf{R}_x^{-1} \left[ 2 \mathbf{R}_x \mathbf{w}(n) - 2 \mathbf{p}_{dx} \right] = (1 - 2\mu)\mathbf{w}(n) + 2\mu \mathbf{p}_{dx} \tag{7.36}
\]

Subtracting \( \mathbf{w}^o \) from both sides of the above equation, we find

\[
\mathbf{w}(n + 1) - \mathbf{w}^o = (1 - 2\mu)[ \mathbf{w}(n) - \mathbf{w}^o ] \tag{7.37}
\]

For \( n = 0 \) and \( \mu = 1/2 \), we obtain \( \mathbf{w}^o \) in one step. However, in practice, \( \nabla_{\mathbf{w}} J(\mathbf{w}) \) and \( \mathbf{R}_x^{-1} \) are estimated, and therefore, the value of the step-size parameter must be less than 0.5.

Introducing \( \mathbf{w} - \mathbf{w}^o = \xi \) in (7.37), we obtain the relation:

\[
\xi(n + 1) = (1 - 2\mu)\xi(n) \tag{7.38}
\]

which has the solution (see Problem 7.3.1):

\[
\xi(n) = (1 - 2\mu)^n \xi(0) \quad \text{or} \quad \mathbf{w}(n) - \mathbf{w}^o = (1 - 2\mu)^n [ \mathbf{w}(0) - \mathbf{w}^o ] \tag{7.39}
\]
Using (6.15), we obtain

\[ J = J_{\text{min}} + (w - w^o)^T \mathbf{R}_x (w - w^o) = J_{\text{min}} + \xi^T \mathbf{R}_x \xi \]

\[ = J_{\text{min}} + \xi^T (Q \Lambda Q^T) \xi = J_{\text{min}} + \xi'^T \Lambda \xi' \]

In connection with (7.39), we find the equation (see Problem 7.3.2):

\[ J(n) = J_{\text{min}} + (1 - 2\mu)^2 n [J(0) - J_{\text{min}}] \] (7.40)

where:

\[ J(n) \] is the value of the performance function, \( J \), when \( w = w(n) \)

To obtain a decaying equivalent expression, we introduce the relation:

\[ (1 - 2\mu)^2 n = e^{-n/\tau} \] (7.41)

where:

\( \tau \) is the \textbf{time constant}

Under the condition \( 2\mu << 1 \), we can use the approximation \( \ln(1 - 2\mu) \approx -2\mu \).

Therefore, the time constant has the value

\[ \tau \approx \frac{1}{4\mu} \] (7.42)

The above equation shows that the Newton’s algorithm has only \textbf{one mode of convergence} (one time constant).

### 7.4 SOLUTION OF THE VECTOR DIFFERENCE EQUATION

If we set \( n = 0 \) in (7.20), we obtain

\[ w(1) = \left[ I - \mu' \mathbf{R}_x \right] w(0) + \mu' \mathbf{p}_{dx} \] (7.43)

Similar, if we set \( n = 1 \), we find

\[ w(2) = \left[ I - \mu' \mathbf{R}_x \right] w(1) + \mu' \mathbf{p}_{dx} \] (7.44)

If we, next, substitute (7.43) in (7.44), we obtain

\[ w(2) = \left[ I - \mu' \mathbf{R}_x \right] w(1) + \mu' \mathbf{p}_{dx} + \sum_{j=0}^{1} \left[ I - \mu' \mathbf{R}_x \right] \mu' \mathbf{p}_{dx} \]

\[ = \left[ I - \mu' \mathbf{R}_x \right] w(0) + \left( \sum_{j=0}^{1} \left[ I - \mu' \mathbf{R}_x \right] \right) \mu' \mathbf{p}_{dx} \] (7.45)
Therefore, the \( n \)th step is easily recognized to be equal to

\[
w(n) = [I - \mu^t R_x] w(0) + \left( \sum_{j=0}^{n-1} [I - \mu^t R_x]^j \right) \mu^t p_{dx}
\]  

(7.46)

The above equation does not provide us with a way to study the convergence of \( w(n) \) to \( w^o \) as \( n \to \infty \). We must decouple the equations and then describe them in a different coordinate system. To accomplish this task, we translate and then rotate the coordinate system.

Finding the eigenvalues and eigenvectors of \( R_x \), we create a diagonal matrix \( \Lambda \) consisting of the eigenvalues of \( R_x \) and a matrix \( Q \) made up of the eigenvectors of \( R_x \) (see Chapter 6). Since \( Q^T Q = Q Q^T = I \), \eqref{7.18} takes the form:

\[
w(n + 1) = Q[I - \mu^t \Lambda]Q^T w(n) + \mu^t p_{dx}
\]

(7.47)

To uncouple the weights (coefficients), we multiply both sides of the above equations by \( Q^T \). Hence,

\[
w'(n + 1) = [I - \mu^t \Lambda]w'(n) + \mu^t p_{dx}'
\]

(7.48)

where we define the following quantities:

\[
w'(n + 1) = Q^T w(n + 1), \quad w'(n) = Q^T w(n), \quad p_{dx}' = Q^T p_{dx}, \quad w^o = Q^T w^o
\]

Next, we obtain the relation:

\[
w'^o = Q^T w^o = Q^T R_x^{-1} p_{dx} = Q^T (Q \Lambda^T Q^T)^{-1} p_{dx}
\]

(7.50)

since \( Q^{-1} = Q^T \) and \((Q \Lambda Q^T)^{-1} = (Q \Lambda Q^T)^{-1} Q^{-1} = (Q^T)^{-1} \Lambda^{-1} Q^T = Q \Lambda^{-1} Q^T \).

The \( i \)th equation of the system given in \eqref{7.47} is

\[
w'_i(n + 1) = [1 - \mu^t \lambda_i] w'_i(n) + \mu^t p'^{i}_{dx} \quad 0 \leq i \leq M - 1
\]

(7.51)

By iteration, the above equation has the following solution (see Problem 7.4.1):

\[
w'_i(n) = (1 - \mu^t \lambda_i) w'_i(0) + \mu^t p'^{i}_{dx} \sum_{j=0}^{n-1} (1 - \mu^t \lambda_i)^j
\]

(7.52)

If we set \( \alpha_i = 1 - \mu^t \lambda_i \), then \eqref{7.52} becomes

\[
w'_i(n) = \alpha_i^n w'_i(0) + \mu^t p'^{i}_{dx} \sum_{j=0}^{n-1} \alpha_i^j = \alpha_i^n w'_i(0) + \mu^t p'^{i}_{dx} \frac{1 - \alpha_i^n}{1 - \alpha_i}
\]

(7.53)
\[ w'_i(n) = \alpha'_n w'_i(0) + \mu' p'_{ds} \sum_{j=0}^{n-1} \alpha'_j = \alpha'_n w'_i(0) + \mu' p'_{ds} \frac{1 - \alpha'_{n}}{1 - \alpha'_i} \]

(7.54)

since the sum is a finite geometric series.

**Example 7.4.1**

For the development of this example, we used the MATLAB program. Hence, the data were found using \( x = \text{randn}(1,10) \) and the desired ones were found using \( d = \text{conv}([1 \ 0.2], x) \) or \( d = \text{filter}([1 \ 0.2], 1, x) \). The correlation matrix was found using the function \( R = \text{toeplitz}([xc(1,10 : 11)]) \), where \( xc = \text{xcorr}(x, \text{'biased'}) \). Similarly, we found the \( \rho dx \) cross-correlation. Hence,

\[
R_i = \begin{bmatrix} 0.7346 & 0.0269 \\ 0.0269 & 0.7346 \end{bmatrix}; \quad [Q, \Lambda] = \text{eig}(R_i)
\]

\[
Q = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}; \quad \Lambda = \begin{bmatrix} 0.7077 & 0 \\ 0 & 0.7071 \end{bmatrix}
\]

and \( \mu' < 2/0.7071 = 2.8285 \) for the solution to converge. We next choose \( [w_0(0) \ w_i(0)]' = [0 \ 0]' \), and hence, \( w'(0) = Q^T w(0) = 0 \). Therefore, (7.54) becomes

\[ w'_i(n) = \mu' p'_{ds} \frac{1 - \alpha'_n}{1 - \alpha'_i} = \frac{p'_{ds}}{\lambda_i} \left[ 1 - (1 - \mu' \lambda_i)^n \right] \]

From (7.49), we obtain

\[
p'_{ds} = Q^T p_{ds} = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix} \begin{bmatrix} 0.7399 \\ -0.0003 \end{bmatrix} = \begin{bmatrix} -0.5234 \\ 0.5230 \end{bmatrix}
\]

Therefore, the system is (we set \( \mu' = 3 \) for convergence)

\[
w'_0(n) = \frac{1}{0.7077} (-0.5234) \left[ 1 - (1 - 2 \times 0.7071)^n \right]
\]

\[
w'_i(n) = \frac{1}{0.7614} (0.5230) \left[ 1 - (1 - 2 \times 0.7614)^n \right]
\]

Since \( w'(n) = Q^T w(n) \) and \( Q^T = Q^{-T} \), we find \( w(n) = Q_T w'(n) \), and at the limit value when \( n \) approaches infinity, the filter coefficients take the values:
Newton’s and Steepest Descent Methods

\[
\begin{bmatrix}
    w_0^o \\
    w_1^o
\end{bmatrix} = Q'w' = \begin{bmatrix}
    -0.7071 & 0.7071 \\
    0.7071 & 0.7071
\end{bmatrix} \begin{bmatrix}
    -0.5234 \\
    0.7077
\end{bmatrix} \begin{bmatrix}
    0.5230 \\
    0.7614
\end{bmatrix} = \begin{bmatrix}
    1.0087 \\
    -0.0373
\end{bmatrix}
\]

PROBLEMS

7.1.1 Verify (7.5).
7.1.2 Plot (7.5) for the following positive values: (a) \(0 < \mu < 1/2r_{xx}(0)\),
(b) \(\mu \equiv 1/2r_{xx}(0)\), and (c) \(1/2r_{xx}(0) < \mu < 1/r_{xx}(0)\), \(r_{xx}(0) = 1\).
7.1.3 Using the one-dimensional Newton’s algorithm, find the third root of 8.
Start with \(x(1) = 1.2\).
7.1.4 Let the MSE be given by the nonquadratic equation
\[J = 2 - (1/35)[(1 - w^2)(4.5 + 3.5w)]\]. Find and plot the recursive Newton’s
algorithm for the coefficients \(\{w(n)\}\) for \(n = 0, 1, 2, \ldots\). Start with \(w(0) = 2\).

7.2.1 Verify (7.22).
7.2.2 Find the time constant for one eigenvalue.
7.3.1 Verify (7.39).
7.3.2 Verify (7.40).
7.3.3 Find the equivalent expression of \(J[w(n)] - J_{\text{min}} + \xi^T(n) \Lambda \xi(n)\) as a
function of \(w(n), p_{dx},\) and \(R_x\).
7.3.4 The correlation matrix of the filter input is
\[
R_x = \begin{bmatrix}
    1 & 0.85 \\
    0.85 & 1
\end{bmatrix}
\]
with eigenvalues \(\lambda_0 = 1.85\) and \(\lambda_1 = 0.15\). Plot the learning curve in a semi-
log format and find the two time constants. Compare your results obtained
from the graph and those given analytically [see (7.27)].

7.4.1 Verify (7.52).
7.4.2 Let
\[
R_p = \begin{bmatrix}
    1 & 0.7 \\
    0.7 & 1
\end{bmatrix}
\]
\[p_{dx} = [0.7 \quad 0.5]^T\]
are the estimates derived from data. Find the vector \(w(n)\).

EDITION PROBLEMS

1. Let \(R = [1 \quad 0.5; 0.25 \quad 1]\) and \(p = [0.5 \quad 0.2]^T\). (a) Find a value of the step
size that ensures the convergence of the steepest descent method; (b) find the
recursion for computing the elements \(w_1(n)\) and \(w_2(n)\) of the \(w(n)\).
2. Assuming one weight coefficient \(w(n)\), find the following: (a) find the MSE
\(J(n)\) as a function of \(w(n)\); (b) find the Wiener solution and the minimum
MSE \(J_{\text{min}}(n)\); (c) sketch a plot of \(J(n)\) versus \(w(n)\).
HINTS–SOLUTIONS–SUGGESTIONS

7.1.1

\[ w(1) = aw(0) + bw^o \]  
\[ w(2) = aw(1) + bw^o = a[aw(0) + bw^o] + bw^o = a^2w(0) + abw^o + bw^o \]

\[ \Rightarrow w(n) = a^n w(0) + (a^{n-1} + a^{n-2} + \cdots + 1)bw^o = a^n w(0) + \frac{1-a^n}{1-a}bw^o, \text{ but} \]
\[ 1-a = b \Rightarrow w(n) = a^n w(0) + w^o - a^n w^o = w^o + [w(0) - w^o]a^n \]

7.1.2 (a) Overdamped case, (b) critically damped, and (c) underdamped case.

7.1.3 Set \( f(x) = x^3 - 8 \) to obtain \( x(n+1) = x(n) - [x(n)^3 - 8]/[3x(n)^2] \). For \( x(1) = 1.2 \), MATLAB gives us the results: \( x(1) = 1.2000, x(2) = 2.6519, x(3) = 2.1471, x(4) = 2.0098, x(5) = 2.0000, x(6) = 2.0000 \). The Book MATLAB program is given as follows:

```matlab
>>x(1) = 1.2;
>>for n = 1:10
>>x(n+1) = x(n)-(x(n)^3-8)/(3*x(n)^2);
>>end;
```

7.1.4

\[ w(n+1) = w(n) - \left\{ [\partial J(w)/\partial w]/[\partial J^2(w)/\partial w^2] \right\} = w(n) \]
\[ = \frac{10.5w^2(n) + 9w(n) - 3.5}{21w(n) + 9} \]

Using MATLAB, we obtain 2.0000, 0.8922, 0.4275, 0.3014, 0.2905, and 0.2905.

7.2.1

\[
\begin{bmatrix}
\xi'_0(n+1) \\
\xi'_1(n+1) \\
\vdots \\
\xi'_{M-1}(n+1)
\end{bmatrix} = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & & & \\
0 & 0 & \cdots & 1
\end{bmatrix} \begin{bmatrix}
\lambda_0 & 0 & \cdots & 0 \\
0 & \lambda_1 & \cdots & 0 \\
\vdots & & & \\
0 & 0 & \cdots & \lambda_{M-1}
\end{bmatrix} \begin{bmatrix}
\xi'_0(n) \\
\xi'_1(n) \\
\vdots \\
\xi'_{M-1}(n)
\end{bmatrix}
\]

\[
= \begin{bmatrix}
(1-\mu\lambda_0)\xi'_0(n) \\
(1-\mu\lambda_1)\xi'_1(n) \\
\vdots \\
(1-\mu\lambda_{M-1})\xi'_{M-1}(n)
\end{bmatrix}
\]
and the results are found since the equality of two vectors is the equality of each corresponding element.

7.2.2

\[(1 - 2\mu\lambda_i)^{2k} = e^{-(k/\tau_i)} \Rightarrow \tau_i = \frac{-1}{2\ln(1 - 2\mu\lambda_i)} \] (1) for small \( \mu \Rightarrow 2\mu\lambda_i << 1 \Rightarrow \\

\ln(1 - 2\mu\lambda_i) \approx -2\mu\lambda_i \), substituting in (1) \( \Rightarrow \tau_i \approx \frac{1}{4\mu\lambda_i} \)

7.3.1

\[w(1) = aw(0) + bw^o \] \( [a = 1 - 2\mu r(0), b = 2\mu r(0)] \)

\[w(2) = aw(1) + bw^o = a[aw(0) + bw^o] + bw^o = a^2w(0) + abw^o + bw^o \]

\[\Rightarrow w(n) = a^n w(0) + (a^{n-1} + a^{n-2} + \cdots + 1)bw^o = a^n w(0) + \left(1 - \frac{a^n}{1-a}\right)bw^o \]

But \( 1 - a = b \Rightarrow w(n) = a^n w(0) + w^o - a^n w^o = w^o + [w(0) - w^o]a^n \).

7.3.2 Equations 6.5 and 7.39 were used in \( J(n) = J_{\text{min}} + \xi^T R_i \xi = J_{\text{min}} + (1 - 2\mu)^n \xi^T(0) R_i \xi(0) \). But \( \xi^T(0) R_i \xi(0) = J(0) - J_{\text{min}} \), and thus Problem 7.3.2 is proved.

7.3.3

\[J(w(n)) = \sigma_d^2 - p_{dx}^T w^o + \xi(n) QQ^T R_i QQ^T \xi(n) \]

\[= \sigma_d^2 - p_{dx}^T w^o + [w(n) - w^o] R_i [w(n) - w^o] \]

\[= \sigma_d^2 - p_{dx}^T w^o + \left[w^T(n) - w^o\right] R_i \left[w(n) - w^o\right] \]

\[= \sigma_d^2 - p_{dx}^T w^o + w^o T R_i w(n) - w^o T R_i w(n) - w^T(n) R_i w^o + w^T R_i w^o \]

But \( (R_i w^o)^T = w^o T R_i^T = w^o T R_x = p_{dx}^T, R_i w^o = p_{dx}, p_{dx}^T w^o = w^o T p_{dx} \), and hence, \( J[w(n)] = \sigma_d^2 - 2w^T(n) p_{dx} + w^T(n) R_i w(n) \).

7.3.4 The time constants are found from the two slopes of the graph.

7.4.1 Start with \( n = 0 \Rightarrow w'_i(1) = (1 - \mu' \lambda_i)^2 w'_i(0) + \mu' p_{dx}' \), and then set

\[n = 1 \Rightarrow w'_i(2) = (1 - \mu' \lambda_i)\left[(1 - \mu' \lambda_i)w'_i(0) + \mu' p_{dx}'\right] + \mu' p_{dx}' \]

\[= (1 - \mu' \lambda_i)^2 w'_i(0) + \mu' p_{dx}' \left[1 + (1 - \mu' \lambda_i)\right] \]

and therefore, at the \( n \)th iteration, we obtain (7.52).
7.4.2 Using the MATLAB function \([v, d] = \text{eig}(R)\), we obtain the following eigenvalues and eigenvectors: \(\lambda_1 = 0.3000, \lambda_2 = 1.7000\), \(q_1 = [-0.7071 \ 0.7071]^T, q_2 = [0.7071 \ 0.7071]^T\). The step-size factor must be set as follows: \(\mu' < 2/1.7 = 1.1765\), so that the solution converges. Choosing \(w(0) = 0\) implies that

\[
w'(0) = Q^T w(0) = 0
\]

\[
\Rightarrow p'_{dx} = Q^T p_{dx} = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix} = \begin{bmatrix} -0.1414 \\ 0.8485 \end{bmatrix}
\]

\[
w'_0(n) = \frac{1}{0.3}(-0.1414)[1 - (1 - \mu'0.3)^n]; \ w'_1(0) = \frac{1}{1.7}0.8485[1 - (1 - \mu'1.7)^n]
\]

Since

\[
w'(n) = Q^T w(n), \ Q^T = Q^{-1}, \ w(n) = Q^T w'(n)
\]

\[
\begin{bmatrix} w_0(n) \\ w_1(n) \end{bmatrix} = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix} \begin{bmatrix} \frac{1}{0.3}(-0.1414)[1 - (1 - \mu'0.3)^n] \\ \frac{1}{1.7}0.8485[1 - (1 - \mu'1.7)^n] \end{bmatrix}
\]

\[
\begin{bmatrix} \frac{1}{0.3}(-0.1414)[1 - (1 - \mu'0.3)^n] + \frac{0.7071}{1.7}0.8485[1 - (1 - \mu'1.7)^n] \\ \frac{0.7071}{0.3}(-0.1414)[1 - (1 - \mu'0.3)^n] + \frac{0.7071}{1.7}0.8485[1 - (1 - \mu'1.7)^n] \end{bmatrix}
\]

**ADDITIONAL PROBLEMS**

1. \(Q, L = \text{eig}([1 \ 0.5; 0.25 \ 1]); \ Q = [0.8165 \ 0.8165; 0.5774 \ 0.5774]; \ L = [1.3536 \ 0.6464].\) Therefore, \(0 < \mu < 1/\lambda_{\text{max}},\) and thus, \(0 < \mu < 1/1.3536 = 0.7388.\)

b. \(w(n + 1) = w(n) + \mu [p - R w(n)]\)

\[
w(n + 1) = w(n) + [0.5; 0.25; 1]* [0.5; 0.25; 1]* w(n))
\]

or

\[
w(n + 1) = w(n) + (1 \ 0; 0 \ 1 - [1 \ 0.5; 0.25 \ 1]) * w(n) - [0.5 \ 0.2]
\]

\[
= [0 \ -0.5; -0.25 \ 0] * w(n) - [0.5 \ 0.2]
\]
Hence,
\[
\begin{bmatrix}
w_1(n+1) \\
w_2(n+1)
\end{bmatrix} = \begin{bmatrix} 0 & -0.5 \\ -0.25 & 0 \end{bmatrix} \begin{bmatrix} w_1(n) \\ w_2(n) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.2 \end{bmatrix}
\]

or
\[
w_1(n+1) = -0.5w_2(n) - 0.5; \quad w_2(n+1) = -0.25w_1(n) - 0.2
\]

For \( n = 0 \), \( w_1(1) = -0.5 \) and \( w_2(1) = -0.2 \) with \( w_1(0) = 0 \) and \( w_2(0) = 0 \)
\[
w_1(2) = -0.5(-0.2) - 0.5 = -0.4, \quad w_2(2) = -0.25(-0.5) - 0.2 = -0.075, \text{ etc.}
\]

2.

\[
J(n) = \sigma_d^2 - 2p_{sd}w(n) + r_{xx}w^2(n);
\frac{\partial J(n)}{\partial w(n)} = 0 - 2p_{sd} + 2r_{xx}w(n) = 0 \implies w^* = \frac{p_{dx}}{r_{xx}}
\]

\[
\Rightarrow J_{\text{min}} = \sigma_d^2 - 2p_{sd} \frac{p_{dx}}{r_{xx}} + r_{xx}\left( \frac{p_{dx}}{r_{xx}} \right)^2 = \sigma_d^2 - \frac{p_{dx}^2}{r_{xx}}
\]

The curve is concave upward with minimum \( J_{\text{min}} \).
The Least Mean-Square Algorithm

8.1 INTRODUCTION

In this chapter, we present the calibrated least mean-square (LMS) algorithm
developed by Widrow and Hoff in 1960. This algorithm is a member of stochastic
gradient algorithms, and because of its robustness and low computational complex-
ity, it has been used in a wide spectrum of applications.

The LMS algorithm has the following most important properties:

1. It can be used to solve the Wiener–Hopf equation without finding matrix
   inversion. Furthermore, it does not require the availability of the autocor-
   relation matrix of the filter input and the cross-correlation between the filter
   input and its desired signal.
2. Its form is simple as well as its implementation, yet it is capable of delivering
   high performance during the adaptation process.
3. Its iterative procedure involves the following:
   a. Computing the output of a finite impulse response (FIR) filter produced
      by a set of tap inputs (filter coefficients)
   b. Generating an estimated error by computing the output of the filter to a
      desired response
   c. Adjusting the tap weights (filter coefficients) based on the estimation error
4. The correlation term needed to find the values of the coefficients at the \( n + 1 \)
   iteration contains the stochastic product \( x(n)e(n) \) without the expectation
   operation that is present in the steepest descent method.
5. Since the expectation operation is not present, each coefficient goes through
   sharp variations (noise) during the iteration process. Therefore, instead of
   terminating at the Wiener solution, the LMS algorithm suffers random vari-
   ation around the minimum point (optimum value) of the error-performance
   surface.
6. It includes a step-size parameter, \( \mu \), that must be selected properly to control
   stability and convergence speed of the algorithm.
7. It is stable and robust for a variety of signal conditions.

8.2 THE LMS ALGORITHM

In Chapter 7, we developed the following relations using the steepest descent method:

\[
\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \nabla_{\mathbf{w}} J[\mathbf{w}(n)]
\] (8.1a)
\[ \nabla_w J[w(n)] = -2p_{dx} + 2R_x w(n) \]  

(8.1b)

The simplest choices of the estimators \( R_x \) and \( p_{dx} \) are the instantaneous estimates defined by

\[ R_x \cong x(n)x^T(n), \quad p_{dx} \cong d(n)x(n) \]  

(8.2)

Substituting the above values in (8.1b) and then combining (8.1a) and (8.1b), we obtain

\[
w(n + 1) = w(n) + 2\mu x(n)[d(n) - x^T(n)w(n)] \\
= w(n) + 2\mu x(n)[d(n) - w^T(n)x(n)] \\
= w(n) + 2\mu e(n)x(n)
\]  

(8.3)

where:

\[ y(n) = w^T(n)x(n) \quad \text{Filter output} \]
\[ e(n) = d(n) - y(n) \quad \text{Error} \]
\[
w(n) = [w_0 \quad w_1 \quad \cdots \quad w_{M-1}]^T \quad \text{Filter taps (coefficients) at time } n \\
x(n) = [x(n) \quad x(n-1) \quad \cdots \quad x(n-M+1)]^T \quad \text{Input data}
\]  

(8.4)

The algorithms defined by (8.3) and (8.4) constitute the LMS algorithm. The algorithm at each iteration requires that \( x(n), d(n), \) and \( w(n) \) are known. The LMS algorithm is a stochastic gradient algorithm if the input signal is a stochastic process. This results in varying the pointing direction of the coefficient vector during the iteration. An FIR adaptive filter realization is shown in Figure 8.1. Figure 8.2 presents the block diagram representation of the LMS filter. Table 8.1 presents the steps for executing the LMS algorithm.

**Book LMS m-Function**

```matlab
function [w,y,e,J,w1,Js] = lms1(x,dn,mu,M)
    %function [w,y,e,J,w1] = lms1(x,dn,mu,M);
    %all quantities are real-valued;
    %x = input data to the filter; dn = desired signal;
    %M = order of the filter;
    %mu = step-size factor; x and dn must be
    %of the same length;
    %Js = smooths the learning curve;
    %w1 = a matrix of dimensions: length(x)xM,
    %each column represents the variation of
    %each filter coefficient;
    N=length(x);w=zeros(1,M);w1=zeros(1,M);
    for n=M:N
        x1=x(n:-1:n-M+1);%for each n the vector x1 is
        %of length M with elements from x in
        %reverse order;
```


The Least Mean-Square Algorithm

\[ 2\mu e(n) = 2\mu [d(n) - y(n)] \]

\[ w_0(n+1) = w_0(n) - \mu \frac{d(n) - y(n)}{x(n)} \]

\[ w_1(n+1) = w_1(n) - \mu \frac{d(n) - y(n)}{x(n-1)} \]

\[ \vdots \]

\[ w_M(n+1) = w_M(n) - \mu \frac{d(n) - y(n)}{x(n-M+1)} \]

**Figure 8.1**
y(n) = w*x1';
e(n) = d(n)-y(n);
w = w+2*mu*e(n)*x1;
w1(n-M+1,:) = w(1,:);
end;
J = e.^2;%J is the learning curve of the adaptation;
for n = 1:length(x)-5
    Js(n) = (J(n)+J(n+1)+J(n+2))/3;
end;

8.3 EXAMPLES USING THE LMS ALGORITHM

The following examples will elucidate the use of the LMS algorithm to different areas of engineering and will bring forth the versatility of this important algorithm.

Example 8.3.1 (Linear Prediction)

We can use an adaptive LMS filter as a predictor as shown in Figure 8.3. The data \{x(n)\} were created by passing a zero-mean white noise \{v(n)\} through the adaptive filter.
The Least Mean-Square Algorithm

through an autoregressive (AR) process described by the difference equation: 
\[ x(n) = 0.6010x(n-1) - 0.7225x(n-2) + v(n) \]. The LMS filter is used to predict the values of the AR filter parameters 0.6010 and -0.7225. A two-coefficient LMS filter predicts \( x(n) \) by 

\[ \hat{x}(n) = \sum_{i=0}^{1} w_i(n)x(n-1-i) \equiv y(n) \]  

(8.6)

Figure 8.4 shows the trajectory of \( w_0 \) and \( w_1 \) versus the number of iterations for two different values of step-size parameter (\( \mu = 0.02 \) and \( \mu = 0.005 \)). The adaptive filter is a two-coefficient filter. The noise is white and Gaussian distributed. The figure shows fluctuations in the values of coefficients as they converge to a neighborhood of their optimum values 0.6010 and -0.7225. As the step-size parameter \( \mu \)
becomes smaller, the fluctuations are not as large, but the convergence speed to the optimal values is slower.

**Book One-Step LMS Predictor m-Function**

```matlab
function [w, y, e, J, w1, Js] = lms_one_step_predictor(x, mu, M)
    % function [w, y, e, J, w1, Js] = lms_one_step_predictor(x, mu, M);
    % x = data = signal plus noise; mu = step-size factor;
    % M = number of filter coefficients; w1 is a matrix and each column
    % is the history of each
    % filter coefficient versus time (iteration) n;
    % N = number of elements
    % of the vector x; Js = smoother learning curve;
    % w1 = a matrix and each column depicts the
    % variation of each coefficient;
    N = length(x);
    y = zeros(1,N);
    w = zeros(1,M);
    for n = M:N-1
        x1=x(n:-1:n-M+1);
        y(n)=w*x1';
        e(n)=x(n+1)-y(n);
        w=w+2*mu*e(n)*x1;
        w1(n-M+1,:)=w(1,:);
    end;
    J=e.^2; % J is the learning curve of the
    % adaptive process;
    for n=1:length(x)-5
        Js(n)=(J(n)+J(n+1)+J(n+2))/3;
    end; % Js=smoothed out learning curve J;

    % For example, to produce two of the four curves, we used the following Book
    % program:
    % mu=0.02; x(1)=0; x(2)=0; M=2;
    % for n=1:2000
    %     x(n+2)=0.6010*x(n+1)-0.7225*x(n)+0.2*randn;
    % end;
    >>[w, y, e, J, w1, Js]=lms_one_step_predictor(x, mu, M);
    >>plot(w1(:,1),’k’);
    >>hold on;
    >>plot(w1(:,2),’k’);xlabel(‘n’);

    % We can use a similar program for the other two curves.
```

**Example 8.3.2 (Modeling)**

Adaptive filtering can also be used to find the coefficients of an unknown filter as shown in Figure 8.5. The data \( x(n) \) were created similar to those in Example 8.3.1. The desired signal is given by \( d(n) = x(n) - 2x(n-1) + 4x(n-2) \). If the output \( y(n) \)
The Least Mean-Square Algorithm

is approximately equal to \( d(n) \), it implies that the coefficients of the LMS filter are approximately equal to those of the unknown system. Figure 8.6 shows the ability of the LMS filter to identify the unknown system. After 1000 iterations, the system is practically identified. In this example, we used \( \mu = 0.15 \) and \( M = 4 \). It is observed that the fourth coefficient is zero as it should be since the system to be identified has only three coefficients and the rest are zero. The Book program to produce Figure 8.6 is given as follows:

**Book Program for System Identification**

```matlab
>>x(1)=0; x(2)=0; mu=0.15; M=4;
>>for n=1:1500
    >> x(n+2)=0.6010*x(n+1)-0.7225*x(n)+0.2*randn;
>>end;
>>for n=1:1500
    >> d(n+2)=x(n+2)-2*x(n+1)+4*x(n);
>>end;
```

**FIGURE 8.5**

![System Identification Diagram](image)

**FIGURE 8.6**

![System Identification Graph](image)
Example 8.3.3 (Noise Cancellation)

A noise cancellation scheme is shown in Figure 8.7. We introduce in this example the following values: \( H_1(z) = 1 \) [or \( h(n) = \delta(n) \)], \( v_1(n) = \) white noise = \( v(n) \), \( L = 1 \), \( s(n) = 0.98^5 \sin(0.2\pi n) \). Therefore, the input signal to the filter is \( x(n) = s(n-1) + v(n-1) \) and the desired signal is \( d(n) = s(n) + v(n) \). The Book LMS function algorithm \texttt{lms1} was used. Figure 8.8 shows the signal, the signal plus noise, and the outputs of the filter for two different sets of coefficients: \( M = 4 \) and \( M = 14 \). The following noise was used: \( v = 0.4[\text{rand}(1,300) - 0.5] \).

Example 8.3.4 (Power Spectrum Approximation)

If a stochastic process is the output of an AR system when its input is a white noise with variance \( \sigma_v^2 \), for example,

\[
x(n) = \sum_{k=1}^{M-1} a_k x(n-k) + v(n) \tag{8.7}
\]

The power of the spectrum corresponding to the stochastic process is given by (see Kay 1993)

\[
S_x(e^{j\omega}) = \frac{\sigma_v^2}{|A(e^{j\omega})|^2} \quad A(e^{j\omega}) = 1 - \sum_{k=1}^{M-1} a_k e^{-j\omega k} \tag{8.8}
\]

Equation 8.7 is also written in the form:

\[
v(n) = x(n) - \sum_{k=1}^{M-1} a_k x(n-k) \tag{8.9}
\]
The Least Mean-Square Algorithm

where:

\( v(n) \) is the nonpredictable portion of the signal or the innovations of the AR process.

Because \( v(n) \) is the nonpredictable portion of \( x(n) \), it suggests to use an adaptive linear predictor for spectrum estimation. If the stochastic process is the result of an AR process, the LMS filter coefficients will be very close to those of the AR system and the two spectra will also be very close to each other.

The steps that closely approximate the coefficients of (8.7) using an LMS adaptive filter are as follows:

1. Use the adaptive LMS filter in the predictive mode (see Figure 8.9).
2. Average the \( K \) most recent values of \( \hat{\mathbf{w}} \).
3. Compute the power spectrum.

FIGURE 8.8

FIGURE 8.9
Let an exact stochastic process be created by an AR system having poles at 
\((z_1, z_2) = 0.95e^{\pm j\pi/4}\). To find the difference equation, which characterizes the AR system, apply the definition of the system in the \(z\)-domain. Hence, we write

\[
H(z) = \frac{\text{Output}}{\text{Input}} = \frac{X(z)}{\sigma_v^2 V(z)} = \frac{1}{(1 - 0.95e^{-j\pi/4}z^{-1})(1 - 0.95e^{j\pi/4}z^{-1})} = \frac{1}{(1 - 1.3435z^{-1}) + (0.9025z^{-2})}
\] (8.11)

The above equation can be written as

\[
X(z) - 1.3435z^{-1}X(z) + 0.9025z^{-2}X(z) = \sigma_v^2V(z)
\]

Taking the inverse \(z\)-transform of both sides of the above equation (remember that a negative exponent of \(z\) means time shifting in the time domain), we obtain the difference equation describing the AR system given by

\[
x(n) = 1.3435x(n-1) - 0.9025x(n-2) + \sigma_v^2v(n)
\] (8.12)

The power spectrum is given by [see (8.8)]

\[
S_x(e^{j\omega}) = \frac{\sigma_v^2}{\left|1 - 1.3435e^{-j\omega} + 0.9025e^{-j2\omega}\right|^2}
\]

\[
= \frac{\sigma_v^2}{\left|1 - 1.3435z^{-1} + 0.9025z^{-2}\right|^2}
\] (8.13)

Figure 8.10 shows the true spectrum and the approximate one. We assumed that the desired signal was produced by the AR filter given by (8.12). The approximate spectrum was found using the following constants: \(\mu = 0.02\), \(M = 3\), \(N = 1000\), \(avn = 3\), \(x(n) = dn(n-1)\), and \(\sigma_v^2 = 0.3385\). The function \texttt{lms\_power\_spectra\_avl} will average the output \(w\) over a number of times as desired by the reader. If we had guessed \(M = 2\) and \(avn = 5\), the two curves will be approximately equal and the filter coefficients are also approximately equal: 1.3452 and \(-0.8551\).

**Book m-Function for Obtaining Power Spectra**

```matlab
function [wa1,v]=lms_power_spectra_av1(a1,a2,a3,\mu,M,N,avn,\nu)
    %lms_power_spectra_av1(a1,a2,a3,\mu,M,N,avn,\nu);
    wa = zeros(1,M);
    dn = zeros(1,N);x = zeros(1,N);
    for k=1:avn
        for n=4:N
            v(n)=\nu*(rand-0.5);
```
The Least Mean-Square Algorithm

\[ dn(n) = -a_1*dn(n-1) - a_2*dn(n-2) - a_3*dn(n-3) + v(n); \]
\[ x(n) = dn(n-1); \]
end;
wa=lms1(x,dn,mu,M);
wa1=wa/avn;

% this function gives averaged w's to be used for finding the approximate spectrum of the output of an AR filter up to the third order; M = number of LMS coefficients;
% N = length of desired signal and input signal to LMS filter;
% avn = number of times w's are averaged; the function is easily modified for AR filter with more coefficients; vr = % controls the variance of the white noise, multiplies the quantity (rand-0.5); the function up to the first end produces the output from the AR filter with coefficients a1, a2, and a3;

\[ [sx,W]=freqz(var(v),[1 -[wa1]],512); \]
% freqz() is a MATLAB function;

n=0:pi/512:pi-(pi/512);
plot(n,abs(sx),'k'); xlabel('Radians'); ylabel('Power Spectrum');

% this function is useful up to three coefficients AR model;
% var controls the variance of the input noise;

\[ 3.0 \]
\[ 2.5 \]
\[ 2.0 \]
\[ 1.5 \]
\[ 1.0 \]
\[ 0.5 \]
\[ 0.0 \]
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5

FIGURE 8.10
The constants that were used in the above example and produced Figure 8.10 were as follows: $a_1 = -1.3435$, $a_2 = 0.9025$, $a_3 = 0.051$, $v_r = 1.5$, $a_{vn} = 5$, $N = 1000$, and $M = 3$. The plot of (8.13) was done using the following expressions:

\[
\begin{align*}
\text{>>om=0:pi/512:pi-(pi/512);} \\
\text{>>plot(om,abs(0.2./(1-1.3435*exp(-j*om)+0.9025*exp(-j*om*2))),'k');}
\end{align*}
\]

**Example 8.3.5 (Sinusoid Plus Noise)**

Assume that the noisy signal is given as $x(n) = a\sin(\omega n) + v(n)$, where $v(n)$ is a white noise sequence. It is desired to obtain the sinusoid with the smallest amount of noise.

*Solution:* Figure 8.11 shows such a situation. The constants and the signals that were used were as follows: $\mu = 0.01$, $M = 30$, $x = \sin(0.1\pi n) + 2(\text{rand} - 0.5)$, and $d = \sin(0.1\pi n)$.

![Figure 8.11](image-url)
Example 8.3.6 (Adaptive Channel Estimation)

Figure 8.12 shows a communication noisy channel that is required to be estimated by the adaptive processing technique.

Solution: In this example, we assumed that the channel is represented by an FIR system with four unknown coefficients \( c = [0.6 \ -0.9 \ 0.2 \ -0.1] \). In addition, we used the following values: \( M = 20 \), random channel noise \( 0.5(\text{rand} - 0.5) \), and the signal \( s(n) = \sin(0.1\pi n) \). Figure 8.13 presents the noisy channel output signal and the reproduction of the input signal to the channel. It is obvious that we recapture the input to the channel signal. In practice, we can create an adaptive filter by sending a known signal, and after the establishment of the proper adaptive filter and assuming that the channel varies slowly, we can use the adaptive filter for a while. Then, periodically, we can find the appropriate adaptive filters.

The Book m-function with which we produced Figure 8.13 is given below.

\[
\begin{align*}
\text{Channel} & \quad + \quad u(n) \quad + \quad y(n) = \text{adpt filter output} \\
\end{align*}
\]

FIGURE 8.12

FIGURE 8.13
**Book m-Function:** \[w,e,J,y,u]=lms\_ex8\_3\_6(s,\mu,c,M)\]

```matlab
function[w,e,J,y,u]=lms_ex8_3_6(s,\mu,c,M)
    % s = signal; \mu = step-size coefficient;
    % c = vector with the channel coefficients;
    % M = number of adaptive filter coefficients
    w=zeros(1,M);
    u=conv(s,c)+0.5*(rand(1,length(conv(s,c)))-0.5);
    for n=M:length(u)-length(c)
        u1=u(n:-1:n-M+1);
        y(n)=w*u1';
        e(n)=s(n+1)-y(n);
        w=w+2*\mu*e(n)*u1;
    end;
    J=e.^2;
end;
```

**Example 8.3.7 (Inverse System Identification)**

To find the inverse of an unknown filter, we place the adaptive filter in series with the unknown system as shown in Figure 8.14. The delay is needed so that the system is causal. Figure 8.15 shows the noisy signal, the output from the adaptive filter, and a typical learning curve. In this example, we used four FIR filter coefficients, and the input to the unknown system was a sine function with a white Gaussian noise. The following Book script m-file was used:

**Book Script m-File**

```matlab
%script file ex8_3_7
n=1:1500;xi=zeros(1,1500);
    xi=xi+sin(0.1*pi*n)+0.06*randn(1,1500);
    x=conv(xi,[0.6 -0.9 0.2 -0.1]);
    for n=1:1500
d(n+1)=x(n);
    end;
[w,y,e,J]=lms1(x(1,1:1450),d(1,1:1450),0.02,8);
```

The functions and constant used were as follows: \(s(n) = \sin(0.1\pi n)\), \(v(n) = 0.06\text{rand}(1,1500)\), unknown system \(c = [0.6\ 0.9\ 0.2\ -0.1]\), \(\mu = 0.02\), and \(M = 8\).
Example 8.3.8 (Adaptive Prediction)

Based on the Figure 8.16, which presents a one-tap AR filter, find the mean-square error (MSE) of a single and an ensemble average result.

**Solution:** The results in Figures 8.17 and 8.18 are found using the Book m-function given below.
Book m-Function: \([w,y,e,Jav]=lms_{ex8\_3\_8}(mu,av,N)\)

function \([w,y,e,Jav]=lms_{ex8\_3\_8}(mu,av,N)\)

for \(m=1:av\)
The Least Mean-Square Algorithm

\% all quantities are real-valued;
\% x = input data to the filter; d = desired signal;
\% M = order of the filter;
\% \mu = step-size factor; x and d must be
\% of the same length;
\% w(1)=0; x(1)=0; w(2)=0;
for r=1:N+2
    x(r+1)=0.95*x(r)+3*(rand-0.5);
    d(r+1)=x(r+1);
end;
for n=1:N
    y(n+1)=w(n+1)*x(n);
    e(n+1,m)=x(n+1)-y(n);
    w(n+2)=w(n+1)+\mu*x(n)*e(n+1,m);
end;
y=zeros(1,N);
end;
Jav=sum((e.\^2),2)/av; \% Jav is the learning curve;

Figure 8.17 was produced using \( \mu = 0.005 \) and \( N = 1000 \). The number of the ensemble was 100. Figure 8.18 shows the creation of the final value of the adaptive filter coefficient versus the number of iterations.

8.4 PERFORMANCE ANALYSIS OF THE LMS ALGORITHM*

By subtracting the Wiener filter \( w^o \) (column vector of the optimum adaptive filter) from both sides of (8.3), we obtain the following equation:

\[
\begin{align*}
    w(n+1) - w^o &= w(n) - w^o + 2\mu e(n)x(n) \\
    \text{(8.14)}
\end{align*}
\]

The vectors \( \xi(n+1) = w(n+1) - w^o \) and \( \xi(n) = w(n) - w^o \) are known as the weight errors, which are described on a coordinate system shifted by \( w^o \) on the \( w \) plane. Therefore, (8.14) becomes

\[
\begin{align*}
    \xi(n+1) &= \xi(n) + 2\mu e(n)x(n) \\
    \text{(8.15)}
\end{align*}
\]

\[
\begin{align*}
    e(n) &= d(n) - y(n) = d(n) - w^T(n)x(n) = d(n) - x^T(n)w(n) \\
         &= d(n) - x^T(n)w^o - x^T(n)[w(n) - w^o] \\
         &= e^o(n) - x^T(n)\xi(n) = e^o(n) - \xi^T(n)x(n) \\
         \text{(8.16)}
\end{align*}
\]

where:

\[
\begin{align*}
    e^o(n) &= d(n) - x^T(n)w^o \\
    \text{(8.17)}
\end{align*}
\]

* The reader can skip the section at the first reading.
is the error when the filter is optimum. Substituting (8.16) in (8.15) and rearranging, we obtain

\[ \xi(n+1) = \xi(n) + 2\mu[e^o(n) - x^T(n)\xi(n)]x(n) \]

\[ = \xi(n) + 2\mu x(n)[e^o(n) - x^T(n)\xi(n)] \]

\[ = [I - 2\mu x(n)x^T(n)]\xi(n) + 2\mu e^o(n)x(n) \tag{8.18} \]

where:

- \( I \) is the identity matrix (ones in the diagonal)
- \( e^o(n) = x^T(n)\xi(n) \) is a scalar

Next, we take the expectation of both sides of (8.18):

\[ E[\xi(n+1)] = E[[I - 2\mu x(n)x^T(n)]]\xi(n) + 2\mu E[e^o(n)x(n)] \]

\[ = E[[I - 2\mu x(n)x^T(n)]]\xi(n) \tag{8.19} \]

Since \( e^o(n) \) is orthogonal to all data (see Section 5.5), the last expression is identically zero. The expression

\[ E[x(n)x^T(n)\xi(n)] = E[x(n)x^T(n)]E[\xi(n)] \tag{8.20} \]

is simplified by incorporating the independence assumption, which states that the present observation of the data \( \{x(n),d(n)\} \) are independent of the past observations \( \{x(n-1),d(n-1),x(n-2),d(n-2)\}, \ldots \), where

\[ x(n) = [x(n) \ x(n-1) \ x(n-2) \cdots x(n-N+1)] \tag{8.21} \]

\[ x(n-1) = [x(n-1) \ x(n-2) \ x(n-3) \cdots x(n-N)] \tag{8.22} \]

Another way to justify the independence assumption is through the following observation: The LMS coefficients \( \mathbf{w}(n) \) at any given time are affected by the whole past history of the data \( \{x(n-1),d(n-1),x(n-2),d(n-2)\}, \ldots \), and therefore, for smaller step-size parameter \( \mu \), the past \( N \) observations of the data have small contribution to \( \mathbf{w}(n) \), and thus, we can say that \( \mathbf{w}(n) \) and \( x(n) \) are weakly dependent. This observation clearly suggests the approximation given by (8.20). Substituting (8.20) in (8.19), we obtain

\[ E[\xi(n+1)] = [I - 2\mu E[x(n)x^T(n)]]E[\xi(n)] \]

\[ = [I - 2\mu \mathbf{R}_e]E[\xi(n)] \tag{8.23} \]

The mathematical forms of (8.23) and (7.20) of the steepest descent method are identical except that the deterministic weight-error vector \( \xi(n) \) in (7.20) has been replaced by the average weight-error vector \( E[\xi(n)] \) of the LMS algorithm. This suggests that, on average, the present LMS algorithm behaves just like the steepest descent algorithm. Like the steepest descent method, the LMS algorithm is controlled by \( M \) modes of convergence, which are dependent on the eigenvalues.
of the correlation matrix $R$. In particular, the convergence behavior of the LMS algorithm is directly related to the eigenvalue spread of $R$, and hence to the power spectrum of the input data $x(n)$. The more flatness of the power spectrum, the higher speed of convergence of the LMS algorithm is attained.

### 8.4.1 Learning Curve

In the development below, we assume the following: (1) the input signal to LMS filter $x(n)$ is a zero-mean stationary process, (2) the desired signal $d(n)$ is a zero-mean stationary process, (3) $x(n)$ and $d(n)$ are jointly Gaussian-distributed random variables for all $n$, and (4) at times $n$, the coefficients $w(n)$ are independent of the input vector $x(n)$ and the desired signal $d(n)$. The validity of (4) is justified for small values of $\mu$ (independent assumption). Assumptions (1) and (2) simplify the analysis. Assumption (3) simplifies the final results so that the third-order and higher moments that appear in the derivation are expressed in terms of the second-order moments due to their Gaussian distribution.

If we take the mean-square average of the error given by (8.16), we obtain

$$J(n) = E[\xi^2(n)] = E[(e^o(n) - \xi^T(n)x(n))[e^o(n) - x^T(n)\xi(n)]]$$

$$= E[e^o^2(n)] + E[\xi^T(n)x(n)x^T(n)\xi(n)] - 2E[e^o(n)\xi^T(n)x(n)]$$

(8.24)

For independent random variables, we have the following relations:

$$E\{xy\} = E\{x\}E\{y\} = E\{xE\{y\}\}$$

(8.25)

$$E\{x^2y^2\} = E\{x^2\}E\{y^2\} = E\{x^2E\{x^2\}\} = E\{xE\{y^2\}x\}$$

(8.26)

Based on the above two equations, the second term of (8.24) becomes

$$E\{[\xi^T(n)x(n)]^2\} = E[\xi^T(n)x(n)x^T(n)x(n)] = E[\xi^T(n)E\{x(n)x^T(n)\}]\xi(n)$$

$$= E[\xi^T(n)R] = \text{tr}\{E[\xi^T(n)R]\}$$

$$= E[\text{tr}\{\xi^T(n)R\}] = E[\text{tr}\{\xi^T(n)\xi\}]$$

(8.27)

$$= \text{tr}\{E[\xi(n)\xi^T(n)]R\} = \text{tr}\{K(n)R\}$$

where we used the following properties: (1) the trace of a scalar is the scalar itself; (2) the trace, $\text{tr}$, and the expectation, $E$, operators are linear and can be exchanged; and (3) the trace of two matrices having $N \times M$ and $M \times N$ dimensions, respectively, is given by

$$\text{tr}\{AB\} = \text{tr}\{BA\}$$

(8.28)

The third term in (8.24), due to the independence assumption and due to the fact that $e^o(n)$ is a constant, becomes

$$E\{e^o(n)\xi^T(n)x(n)\} = E\{\xi^T(n)x(n)e^o(n)\} = E\{\xi^T(n)E\{x(n)e^o(n)\}\} = 0$$

(8.29)
The second term is equal to zero due to the orthogonality property (see Section 5.5). Substituting (8.29) and (8.28) in (8.24), we obtain

\[
J(n) = E\{e^2(n)\} = J_{\text{min}} + \text{tr}[K(n)R_i], \quad J_{\text{min}} = E\{([e^n(n)]^2)\}
\] (8.30)

However, \( R_i = Q\Lambda Q^T \), where \( Q \) is the eigenvector matrix and \( \Lambda \) is the diagonal eigenvalue one. Hence, (8.30) becomes

\[
J(n) = J_{\text{min}} + \text{tr}[K(n)Q\Lambda]Q^T = J_{\text{min}} + \text{tr}[Q^TK(n)Q]\Lambda
\] (8.31)

\[
= J_{\text{min}} + \text{tr}[E\{[\xi'(n)Q^T(n)\Lambda]\}
\]

where:

\[
K'(n) = E\{[\xi'(n)\xi^{RT}(n)]\}
\] (8.32)

Recall that \( \xi'(n) \) is the weight-error vector in the coordinate system defined by the basis vectors, which are specified by the eigenvectors of \( R_i \). Since \( \Lambda \) is diagonal, (8.31) becomes

\[
J(n) = J_{\text{min}} + \sum_{i=0}^{M-1} \lambda_i k_{ij}(n) = J_{\text{min}} + \sum_{i=0}^{M-1} \lambda_i E\{\xi_{ij}^2(n)\}
\] (8.33)

where:

\( k_{ij}(n) \) is the \( ij \)th element of the matrix \( K'(n) \)

The learning curve can be obtained by any one of the equations: (8.30), (8.31), and (8.33). It turns out that, on average, the learning curve above is similar to the one given by the steepest descent algorithm.

The general solution of \( \xi'(n) \) is given by (7.30), and hence, (8.33) becomes

\[
J(n) = J_{\text{min}} + \sum_{i=0}^{M-1} \lambda_i (1 - 2\mu \lambda_i)^2 E\{\xi_{ii}^2(0)\}
\] (8.34)

**Example 8.4.1**

The filter \( H_1(z) \), which produces the desired signal \( d(n) = x(n) \), is represented by the difference equation

\[
x(n) + a_1x(n-1) + a_2x(n-2) = v(n)
\]

where \( a_1 \) and \( a_2 \) are the system coefficients and \( v(n) \) is a zero-mean white noise process of variance \( \sigma^2 \).

To simulate the system coefficients \( a_1 \) and \( a_2 \), we use the adaptive predictor (see Figure 8.3a). The LMS algorithm is

\[
w(n) = w(n-1) + 2\mu x(n-1)\tilde{e}(n)
\]

\[
e(n) = d(n) - w^T(n)x(n-1) = x(n) - w^T(n)x(n-1)
\] (8.35)

\[
J(n) = e^2(n) = \text{Learning curve}
\]
In this example, we used the following constants: $M = 6$, $a_1 = -0.96$, $a_2 = 0.2$, averaging number $= 200$, $\sigma_v^2 = 0.33$. Figure 8.19 shows the results. The following Book m-function was used.

**Book m-Function**

```matlab
function [J] = lms_ex8_4_1(mu, M, avn)
    %function [J] = lms_ex8_4_1(mu, M, avn);
    %M = number of filter coefficients;
    %avn = number of times the MSE (J) to be averaged;
    dn(1) = 0; dn(2) = 0; x(1) = 0; x(2) = 0;
    for k = 1:avn
        for n = 3:2000
            dn(n) = 0.96*dn(n-1) - 0.2*dn(n-2) + 2*(rand-0.5);
            x(n) = dn(n-1);
        end;
        [w, y, e, Jl] = lms1(x, dn, mu, M);
        Jk(k,:) = Jl;%this expression creates a
        %matrix avn by 2000;
    end;
    J = sum(Jk, 1)/100;
end;
```

![Image of Figure 8.19](image-url)
8.4.2 The Coefficient-Error or Weighted-Error Correlation Matrix

Since the MSE is related to weight-error correlation matrix $K(n)$, the matrix is closely related to the stability of the LMS algorithm. Therefore, $J(n)$ is bounded if the elements of $K(n)$ are also bounded. Since $K(n)$ and $K'(n)$ are related, the stability can be studied by using either one. Multiply both sides of (8.18) by $Q^T$ and use the definitions $\xi'(n) = Q^T \xi(n)$ and $x'(n) = Q^T x(n)$ to obtain (see Problem 7.4.1)

$$\xi'(n+1) = [I - 2\mu x'(n)x'^T(n)]\xi'(n) + 2\mu e^o(n)x'(n)$$

(8.36)

Next, multiply (8.36) by its transpose and take the ensemble average of both sides to obtain (see Problem 8.4.2)

$$K'(n+1) = K'(n) - 2\mu E[x'(n)x'^T(n)\xi(n)\xi'^T(n)]$$

$$-2\mu E[\xi'(n)\xi'^T(n)x'(n)x'^T(n)]$$

$$+4\mu^2 E[x'(n)x'^T(n)\xi(n)\xi'^T(n)x'(n)x'^T(n)]$$

$$+2\mu E[e^o(n)x'(n)\xi'^T(n)]$$

$$+2\mu E[e^o(n)\xi'(n)x'^T(n)]$$

$$-4\mu^2 E[e^o(n)\xi'(n)x'^T(n)\xi(n)x'^T(n)]$$

$$-4\mu^2 E[e^o(n)\xi(n)x'^T(n)\xi'(n)x'^T(n)]$$

(8.37)

$$+4\mu^2 E[e^o^2(n)x'(n)x'^T(n)]$$

Based on the previous independent assumption, we note the following: (1) $\xi(n)$ is independent of the data $x(n)$ and the desired signal $d(n)$, which is also true for the transformed variables $x'(n)$; (2) $\xi'(n)$ is independent of $x'(n)$ and $d(n)$; (3) $d(n)$ and $x'(n)$ are zero mean and are jointly Gaussian since $d(n)$ and $x(n)$ have the same properties; (4) by applying the orthogonality relationship, we obtain $E[e^o(n)x'(n)] = E[e^o(n)Q^T x(n)] = Q^T E[e^o(n)x(n)] = Q^T 0 = 0$; (5) $e^o(n)$ depends only on $d(n)$ and $x(n)$; (6) from (4) $e^o(n)$ and $x'(n)$ are uncorrelated (Gaussian variables are also independent); (7) $e^o(n)$ has zero mean. With the above assumptions in mind, the factors of (8.37) become (see Problem 8.4.3)

$$E[x'(n)x'^T(n)\xi'(n)\xi'^T(n)] = E[x'(n)x'^T(n)]E[\xi'(n)\xi'^T(n)]$$

(8.38)

$$E[Q^T x(n)x'^T(n)Q]K'(n) = Q^T R_{x}\Lambda K'(n) = \Lambda K'(n)$$

(8.39)

$$E[x'(n)x'^T(n)\xi'(n)\xi'^T(n)x'(n)x'^T(n)] = 2\Lambda K'(n) + \text{tr} \{\Lambda K'(n)\}$$

(8.40)

Because $e^o(n)$, $x'(n)$, and $\xi'(n)$ are mutually independent, $E[e^o(n)] = 0$ are then true:

$$E[e^o(n)x'(n)\xi'^T(n)] = E[e^o(n)]E[x'(n)\xi'^T(n)] = 0$$

(M $\times$ M matrix)
The Least Mean-Square Algorithm

\[ E[c^n(n)\xi'(n)x'^T(n)] = 0 \]  
\[ E[c^n(n)x'(n)\xi'^T(n)x'^T(n)] = 0 \]  
\[ E[c^n(n)x'(n)x'^T(n)\xi'(n)] = 0 \]  
\[ E[c^{o^2}(n)x'(n)x'^T(n)] = E[c^{o^2}(n)]E[x'(n)x'^T(n)] = J_{min}\Lambda \]  

Substituting (8.38) through (8.45) in (8.37), we obtain

\[ K'(n + 1) = K'(n) - 2\mu(\Lambda K'(n) + K'(n)\Lambda) + 8\mu^2\Lambda K'(n)\Lambda + 4\mu^2\text{tr}(\Lambda K'(n))\Lambda + 4\mu^2J_{min}\Lambda \]  

Concentrating on the \(i\)th component of both sides of (8.46), we obtain (see Problem 8.4.4)

\[ k_i'(n + 1) = (1 - 4\mu\lambda_i + 8\mu^2\lambda_i^2)k_i'(n) + 4\mu^2\lambda_i \sum_{j=0}^{M-1} \lambda_j k_{ij}'(n) + 4\mu^2J_{min}\lambda_i \]  

Since \(K'(n)\) is a correlation matrix, \(k^2_{ij} \leq k_i'(n)k_{ij}'(n)\) for all values of \(i\) and \(j\), and since the update of \(k_i'(n)\) is independent of the off-diagonal elements of \(K'(n)\), the convergence of the diagonal elements is sufficient to secure the convergence of all the elements, and thus guarantees the stability of the LMS algorithm. Therefore, we concentrate on (8.47) with \(i = 0, 1, 2, \ldots, M - 1\).

Equation 8.47 can be written in the following matrix form (see Problem 8.4.4):

\[ k(n + 1) = FK'(n) + 4\mu^2J_{min}\lambda \]  

where:

\[ k(n) = \begin{bmatrix} k_{00}'(n) & k_{01}'(n) & \cdots & k_{0,M-1}'(n) \\ k_{10}'(n) & k_{11}'(n) & \cdots & k_{1,M-1}'(n) \\ \vdots & \vdots & \ddots & \vdots \\ k_{M-1,0}'(n) & k_{M-1,1}'(n) & \cdots & k_{M-1,M-1}'(n) \end{bmatrix}^T \]  

\[ \lambda = \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{M-1} \end{bmatrix}^T \]  

\[ F = \text{diag}[f_0, f_1, \ldots, f_{M-1}]^T \]  

\[ f_i = 1 - 4\mu\lambda_i + 8\mu^2\lambda_i^2 \]  

It has been found that if the eigenvalues of \(F\) are less than one, the LMS algorithm is stable or equivalently the elements \(k(n)\) remain bounded. An indirect approach to obtain stability is given in Section 8.4.3.

### 8.4.3 Excess MSE and Misadjustment

We may write the expression in (8.30) as the difference between the MSE and the minimum MSE as follows:

\[ J_{ex}(n) = J(n) - J_{min} = \text{tr}[K(n)R_s] \]
The steady-state form of (8.52) is
\[
J_{\text{ex}}(\infty) = J(\infty) - J_{\text{min}} = \text{tr}[K(\infty)R_x]
\] (8.53)
and it is known as excess MSE. Equation 8.33 gives another equivalent form, which is
\[
J_{\text{ex}}(\infty) = \sum_{i=0}^{M-1} \lambda_i k_i'(\infty) = \xi^T K'(\infty)
\] (8.54)
As \( n \to \infty \), we set \( k'(n+1) = k'(n) \), and hence, (8.48) gives another equivalent form, which is
\[
k'(\infty) = 4\mu^2 J_{\text{min}}(I - F)^{-1} \lambda
\] (8.55)
As a result, (8.54) becomes
\[
J_{\text{ex}}(\infty) = 4\mu^2 J_{\text{min}} \lambda^T (I - F)^{-1} \lambda
\] (8.56)
which indicates that \( J_{\text{ex}} \) is proportional to \( J_{\text{min}} \). The normalized \( J_{\text{ex}}(\infty) \) is equal to
\[
\mp\% = \frac{J_{\text{ex}}(\infty)}{J_{\text{min}}} = 4\mu^2 \lambda^T (I - F)^{-1} \lambda
\] (8.57)
which is known as the misadjustment factor.

If \( A(N \times N) \), \( B(M \times M) \), and \( C(N \times M) \) are matrices that have an inverse, then
\[
(A + CBC^T)^{-1} = A^{-1} - A^{-1}C(B^{-1} + C^TA^{-1}C)^{-1}C^TA^{-1}
\] (8.58)
But
\[
I - F = \begin{bmatrix}
1 - f_0 & 0 & \cdots & 0 \\
0 & 1 - f_1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 - f_{M-1}
\end{bmatrix}
= 4\mu^2 \lambda^T
\] (8.59)
where:
\[
a = -4\mu^2
\]
Therefore, (8.57) takes the form (see Problem 8.4.5)
\[
\mp\% = -a\lambda^T (F_1 + a\lambda \lambda^T)^{-1} \lambda = -a\lambda^T \left[ F_1^{-1} - \frac{aF_1^{-1} \lambda \lambda^T F_1^{-1}}{1 + a^2 \lambda^T F_1^{-1} \lambda} \right] \lambda
\] (8.60)
where in (8.58) we set \( C = \lambda \), \( B = I \), \( a' = -a \), and \( A = F_1 \). Small \( \mp\% \) implies that the summation on the numerator is small. If, in addition, \( 2\mu \lambda_i \ll 1 \), we obtain the following results:
The Least Mean-Square Algorithm

\[ \sum_{i=0}^{M-1} \frac{\mu \lambda_i}{1 - \mu \lambda_i} \cong \mu \sum_{i=0}^{M-1} \mu \text{tr}[R_i] \quad (8.61) \]

Hence, (8.60) becomes

\[ \mathcal{M} = \frac{\mu \text{tr}[R_i]}{1 - \mu \text{tr}[R_i]} \quad (8.62) \]

In addition, for \( \mathcal{M} \ll 0.1 \), the quantity \( \mu \text{tr}[R_i] \) is small and (8.62) becomes

\[ \mathcal{M} = \mu \text{tr}[R_i] \quad (8.63) \]

Since \( r_{xx}(0) \) is the mean-square value of the input signal to an \( M \)-coefficient filter, we write

\[ \mathcal{M} = \mu M r_{xx}(0) = \mu M E[x^2(0)] = \mu M \text{(power input)} \quad (8.64) \]

The above equation indicates that to keep the misadjustment factor small, and at a specific desired value as the signal power changes, we must adjust the value of \( \mu \).

**8.4.4 Stability**

If we set

\[ L = \sum_{i=0}^{M-1} \frac{\mu \lambda_i}{1 - \mu \lambda_i} \quad (8.65) \]

then

\[ \mathcal{M} = \frac{L}{1 - L} \quad (8.66) \]

We observe that \( L \) and \( \mathcal{M} \) are increasing functions of \( \mu \) and \( L \), respectively (see Problem 8.4.6). Since \( L \) reaches 1 as \( \mathcal{M} \) goes to infinity, this indicates that there is a value \( \mu_{\text{max}} \) that \( \mu \) cannot surpass. To find the upper value of \( \mu \), we must concentrate on the expression that can easily be measured in practice. Therefore, from (8.66) we must have

\[ \sum_{i=0}^{M-1} \frac{\mu \lambda_i}{1 - \mu \lambda_i} \leq 1 \quad (8.67) \]

The above equation indicates that the maximum value of \( \mu \) must make (8.67) an equality. It can be shown that the following inequality holds:

\[ \sum_{i=0}^{M-1} \frac{\mu \lambda_i}{1 - \mu \lambda_i} \leq \frac{\mu \sum_{i=0}^{M-1} \lambda_i}{1 - 2\mu \sum_{i=0}^{M-1} \lambda_i} \quad (8.68) \]
Hence, if we solve the equality

\[ \frac{\mu \sum_{i=0}^{M-1} \lambda_i}{1 - 2\mu \sum_{i=0}^{M-1} \lambda_i} = 0 \tag{8.69} \]

for \( \mu \), then (8.67) is satisfied. The solution of the above equality is

\[ \mu_{\text{max}} = \frac{1}{3} \sum_{i=0}^{M-1} \lambda_i = \frac{1}{3 \text{tr} \{ \mathbf{R} \}} = \frac{1}{3(\text{Input power})} \tag{8.70} \]

and hence,

\[ 0 < \mu < \frac{1}{3} \sum_{i=0}^{M-1} \lambda_i = \frac{1}{3 \text{tr} \{ \mathbf{R} \}} \tag{8.71} \]

### 8.4.5 The LMS and Steepest Descent Methods

The following similarities and differences exist between the two methods:

1. The steepest descent method reaches the minimum MSE \( J_{\text{min}} \) as \( n \to \infty \) and \( \mathbf{w}(n) \to \mathbf{w}_o \).
2. The LMS method produces an error \( J(\infty) \) that approaches \( J_{\text{min}} \) as \( n \to \infty \) and remains larger than \( J_{\text{min}} \).
3. The LMS method produces a \( \mathbf{w}(n) \), as the iterations \( n \to \infty \), which is close to the optimum \( \mathbf{w}_o \).
4. The steepest descent method has a well-defined learning curve consisting of a sum of decaying exponentials.
5. The LMS learning curve is a sum of noisy decaying exponentials and the noise, in general, decreases the smaller values the step-size parameter \( \mu \) takes.
6. In the steepest descent method, the correlation matrix \( \mathbf{R}_x \) of the data \( x(n) \) and the cross-correlation vector \( \mathbf{p}_{dx}(n) \) are found using ensemble averaging operations from the realizations of the data \( x(n) \) and the desired signal \( d(n) \).
7. In the LMS filter, an ensemble of learning curves is found under identical filter parameters and then averaged point by point.

### 8.5 Complex Representation of the LMS Algorithm

In some practical applications, it is mathematically attractive to have complex representation of the underlying signals. For example, baseband signals in a quadrature amplitude modulation (QAM) format are written as a summation of two components: real \textbf{in-phase} component and imaginary \textbf{quadrature} component. Furthermore, signals detected by a set of antennas are also written in their complex form for easy mathematical manipulation. For this reason, we shall present in this section the most rudimentary derivation of the LMS filter, assuming that the signals are complex.
The Least Mean-Square Algorithm

In the case where complex-type signals must be processed, we write the output of the adaptive filter in the form:

$$y(n) = \sum_{k=0}^{M-1} w_k^*(n)x(n-k) \quad (8.72)$$

and the error is given by

$$e(n) = d(n) - y(n) \quad (8.73)$$

Therefore, the MSE is

$$J = E\{e(n)e^*(n)\} = E\{e(n)^2\} \quad (8.74)$$

Let us define the complex filter coefficient as follows:

$$w_k = a_k(n) + jb_k(n) \quad k = 0, 1, \ldots, M - 1 \quad (8.75)$$

Then the gradient operator $\nabla$ has the following $k$th element:

$$\nabla w_k \triangleq \nabla_k = \frac{\partial}{\partial a_k(n)} + j\frac{\partial}{\partial b_k(n)} \quad k = 0, 1, \ldots, M - 1 \quad (8.76)$$

which will produce the following $k$th element of the multielement gradient vector $\nabla J$:

$$\nabla_k J = \frac{\partial J}{\partial a_k(n)} + j\frac{\partial J}{\partial b_k(n)} \quad k = 0, 1, \ldots, M - 1 \quad (8.77)$$

It is noted that the gradient operator is always used to find the minimum points of a function. The above equation indicates that a complex constraint must be converted to a pair of real constraints. Hence, we set

$$\frac{\partial}{\partial a_k(n)} = \frac{\partial}{\partial b_k(n)} = 0 \quad k = 0, 1, \ldots, M - 1 \quad (8.78)$$

The $k$th element of the gradient vector, using (8.74), is

$$\nabla_k J = E\left\{ \frac{\partial e(n)}{\partial a_k(n)}e^*(n) + \frac{\partial e^*(n)}{\partial a_k(n)}e(n) + j\frac{\partial e(n)}{\partial b_k(n)}e^*(n) + j\frac{\partial e^*(n)}{\partial b_k(n)}e(n) \right\} \quad (8.79)$$

Taking into consideration (8.72) and (8.73), we obtain

$$\frac{\partial e(n)}{\partial a_k(n)} = \frac{\partial d(n)}{\partial a_k(n)} - \sum_{k=0}^{M-1} \left[ \frac{\partial w_k^*(n)}{\partial a_k(n)} \right] x(n-k)$$

$$= 0 - \sum_{k=0}^{M-1} \left[ \frac{\partial [a_k(n) - jb_k(n)]}{\partial a_k(n)} \right] x(n-k) = -x(n-k) \quad (8.80)$$
Similarly, we obtain

$$\frac{\partial e^*(n)}{\partial b_k(n)} = jx(n-k), \frac{\partial e^*(n)}{\partial b_k(n)} = -jx^*(n-k), \frac{\partial e(n)}{\partial b_k(n)} = -jx(n-k)$$  \hspace{1cm} (8.81)

Introducing the last three equations into (8.79), we obtain the relationship:

$$\nabla J \equiv \nabla = -2E\{x(n-k)e^*(n)\}$$  \hspace{1cm} (8.82)

and thus, the gradient vector becomes

$$\nabla = -jx(n-k) + \mu$$  \hspace{1cm} (8.83)

Next, we replace the ensemble average in (8.83) by the instantaneous estimate $e^*(n)x(n)$ to obtain

$$w(n+1) = w(n) + 2\mu e^*(n)x(n)$$  \hspace{1cm} (8.85)

which is the LMS recursion formula when we are involved with complex-valued processes. The LMS algorithm for complex signals is given in Table 8.2.

### Example 8.5.1

With the input signal $x(n) = \sin(0.1\pi n) + j1.5(\text{rand} - 0.5)$, the desired signal $d(n) = \sin(0.1\pi n)$, $\mu = 0.01$, and the number of coefficients $M = 16$, we obtain the results shown in Figure 8.20. The Book m-function that produced the results is given below.

**Book m-Function for Complex LMS Algorithm**

```matlab
function [w,y,e,J,w1]=lms_complex_norm_lms(x,dn,mubar,M,c)
%function [w,y,e,J,w1]=lms_complex_norm_lms(x,dn,mubar,M,c)
%x = input data to the filter;dn = desired signal;
```
The Least Mean-Square Algorithm

% M = filter order; c = small constant;  
% mubar = step-size equivalent parameter;  
% x and dn must be of the same length; J = learning curve;  
N=length(x);  
y=zeros(1,N);  
w=zeros(1,M)+j*zeros(1,M); % initialized filter coefficient vector;  
for n=M:N  
x1=x(n:-1:n-M+1); % for each n vector x1 is of length M with elements from x in reverse order;  
y(n)=conj(w)*x1';  
e(n)=dn(n)-y(n);  
w=w+(mubar/(c+conj(x1)*x1'))*conj(e(n))*x1;  
w1(n-M+1,:)=w(1,:);  
end;  
J=e.^2; % the columns of the matrix w1 depict the history of the filter coefficients;

FIGURE 8.20

PROBLEMS

8.2.1 Develop the LMS algorithm for complex-valued functions.

8.3.1 If an AR system has poles $0.85e^{\pm j(\pi/4)}$ with input white noise $v(n)$, find its discrete-time representation.

8.4.1 Verify (8.36).
8.4.2 Verify (8.37).
8.4.3 Verify (8.40).
8.4.4 Verify (8.47).
8.4.5 Verify (8.60).
8.4.6 Verify that (8.65) and (8.66) are increasing functions of \( \mu \) and \( L \).
8.4.7 Use the average eigenvalue to obtain the average constant for the LMS algorithm and state your observations for \( M \).
8.4.8 Discuss the effect of the initial value \( w(0) \) on the transient behavior of the LMS algorithm.
8.4.9 Develop the LMS algorithm using the steepest descent method and setting \( J(n) = e^2(n) \) instead of \( E[e^2(n)] \).
8.4.10 Based on the formula \( L = \sum_{i=0}^{M-1} (\mu \lambda_i / 1 - \mu \lambda_i) \) developed above [Misadjustment = \( L/(1-L) \)], plot \( L \) versus \( \mu \) if the input signal to an LMS filter has the following three sets of eigenvalues given below. For stability, we must have \( L < 1 \). We have also developed above the relation \( 0 < \mu \leq (1/3 \sum_{i=0}^{M-1} \lambda_i) = \mu_{max} \).

\[
\begin{array}{cccccccc}
\lambda_0 & \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 & \lambda_6 & \lambda_7 & \lambda_8 & \lambda_9 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

(1) (2) (3)

8.4.11 For the communication channel shown in Figure P8.4.11. Vary the value of the delay, \( L \), in the range from 4 to 15 for the two channels shown in the figure. For each case, find the optimal delay value \( L \) that gives the minimum MSE. The number of the coefficients is set equal to \( M = 16 \).

HINTS–SOLUTIONS–SUGGESTIONS

8.2.1
\[
R_{xx} = x(n)x^H(n), \quad p_{dxx} = x(n)d^*(n), \quad \nabla J = -2x(n)d^*(n) + 2x(n)x^H(n)w(n) \\
w(n+1) = w(n) + 2\mu x(n)[d^*(n) - x^H(n)w(n)] \\
y(n) = w^H(n)x(n) = x^H(n)w(n), \quad e(n) = d(n) - y(n) \\
w(n+1) = w(n) + 2\mu x(n)e^*(n). \quad H = \text{conjugate transpose(or Hermitian)}.
\]
8.3.1

\[ H(z) = \frac{X(z)}{V(z)} = \frac{1}{\left\{ 1 - 0.85e^{j(\pi/4)z^{-1}} \right\} \left\{ 1 - 0.85e^{-j(\pi/4)z^{-1}} \right\}} \]

or

\[ V(z) = X(z) \left[ 1 - 1.7 \left( \frac{\sqrt{2}}{2} \right) z^{-1} + (0.85)^2 z^{-2} \right] \]

Hence, the inverse z-transform gives \( v(n) = x(n) - 1.202x(n-1) + 0.7225x(n-2) \).

8.4.1

\[ Q^T \xi(n+1) = Q^T [\xi(n) - 2\mu x(n)x^T(n)\xi(n)] + 2\mu e^o(n)Q^T x(n) \]

\[ = Q^T [\xi(n) - 2\mu Q^T x(n)x^T(n)Q^{-1} Q^T \xi(n)] + 2\mu e^o(n)x'(n) \]

\[ = \xi'(n) - 2\mu Q^T x(n)(Q^{-1} Q^T) \xi(n) + 2\mu e^o(n)x'(n) \]

\[ = [I - 2\mu x'(n)x^T(n)]\xi(n) + 2\mu e^o(n)x'(n) ; \]

\[ Q^{-1} = (Q^{-1})^T = Q \]

8.4.2 Hint: \( \xi^T(n+1) = \xi^T(n) - 2\mu \xi^T(n)x'(n)x^T(n) + 2\mu e^o(n)x^T(n) \).

8.4.3 Because \( x'(n) \) and \( \xi'(n) \) are independent.

\[
E\{x'(n)x^T(n)\xi'(n)(n)\xi(n)x^T(n)\} = E\{x'(n)x^T(n)\} E\{\xi'(n)(n)\xi(n)x^T(n)\} \\
= \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} x_i'(n)x_j'(n)k_{ij}(n) \]

(2)

\[ C(n) = M \times M \text{ matrix} = x'(n)x^T(n)K'(n)x'(n)x^T(n) \]  

(3)

\[ c_{lm}(n) = x_i'(n)x_m'(m) \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} x_i'(n)x_j'(n)k_{ij}(n) \]  

(4)

\[ E\{c_{lm}(n)\} = \sum_{i=0}^{M-1} E\{x_i'(n)x_m'(m)x_i'(n)x_j'(n)k_{ij}(n)\} \]  

(5)
For Gaussian random variables, we have

\[ E[x_1x_2x_3x_4] = E[x_1x_2]E[x_3x_4] + E[x_1x_3]E[x_2x_4] + E[x_1x_4]E[x_2x_3] \]  

(6)

\[ E[x_i(n)x'_j(n)] = \lambda_i \delta(i - j) \]  

(7)

The above equation is due to the relation

\[ E[Q^T x(n)x(n)^T Q] = Q^T R Q = \Lambda, R_{ij} = \lambda_i q_i \]

\[ E[x_i(n)x'_m(n)x'_j(n)x'_k(n)] = \lambda_i \lambda_m \delta(l - m) \delta(i - j) + \lambda_i \lambda_m \delta(l - i) \delta(m - j) + \lambda_i \lambda_m \delta(l - j) \delta(m - i) \]  

(8)

Substitute (8) in (5) to find

\[ E\{c_{lm}(n)\} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \lambda_i \lambda_m \delta(l - m) \delta(i - j) k'_{ij}(n) \]

\[ + \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \lambda_i \lambda_m \lambda_e \delta(l - i) \delta(m - j) k''_{ij}(n) \]

\[ + \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \lambda_i \lambda_m \delta(l - j) \delta(m - i) k'''_{ij}(n) \]

\[ = \lambda_i \delta(l - m) \sum_{i=0}^{M-1} \lambda_i k'_{ii}(n) + \lambda_i \lambda_m k'_{lm}(n) + \lambda_l \lambda_m k''_{ml}(n) \]

for \( l = 0, 1, \ldots, M - 1 \) and \( m = 0, 1, \ldots, M - 1 \). But \( k'_{lm}(n) = k''_{ml}(n) \) and \( \sum_{i=0}^{M-1} \lambda_i k''_{ml}(n) = \text{tr}[\Lambda K'(n)] \) and \( \lambda_i \lambda_m k'_{lm}(n) + \lambda_l \lambda_m k''_{ml}(n) = 2\lambda_i \lambda_m k'''_{ml}(n) \)

Based on these results, (8.40) is apparent.

8.4.4

\[ \begin{bmatrix} k'_{00}(n+1) & k'_{01}(n+1) & \cdots \\ k'_{10}(n+1) & k'_{11}(n+1) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} f_0 + 4\mu^2 \lambda_0^2 & 4\mu^2 \lambda_1 \lambda_0 & \cdots & 4\mu^2 \lambda_{M-1} \lambda_0 \\ \lambda_0 f_1 & f_1 + 4\mu^2 \lambda_1^2 & \cdots & 4\mu^2 \lambda_{M-1} \lambda_1 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_0 \lambda_{M-1} f_{M-1} & \lambda_{M-1} f_{M-1} & \cdots & f_{M-1} + 4\mu^2 \lambda_{M-1}^2 \end{bmatrix} \]

\[ \begin{bmatrix} k'_{00}(n+1) & k'_{01}(n+1) & \cdots \\ k'_{10}(n+1) & k'_{11}(n+1) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ 0 \\ \cdots \\ \lambda_{M-1} \end{bmatrix} + 4\mu^2 f_{mn} \begin{bmatrix} \lambda_0 \\ 0 \\ \cdots \\ \lambda_{M-1} \end{bmatrix} \]
The Least Mean-Square Algorithm

where the $i$th component of both sides is $k_i'(n + 1) = f_i k_i'(n) + 4\mu^2 \lambda_i [\lambda_0 k_0'(n) + 4\mu^2 \lambda_0 k_1'(n) + \cdots + 4\mu^2 \lambda_{M-1} k_{M-1}'(n)] + 4\mu^2 J_{\text{min}} \lambda_i$ which is identical to (8.47).

8.4.5

\[
a[\lambda_0, \lambda_1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda_0 \\ \lambda_1 \end{bmatrix}
\]

\[
a^2[\lambda_0, \lambda_1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_0 \lambda_0 \\ \lambda_0 \lambda_1 \end{bmatrix} = \begin{bmatrix} \lambda_0 \\ \lambda_1 \end{bmatrix}
\]

\[
1 - a[\lambda_0, \lambda_1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda_0 \\ \lambda_1 \end{bmatrix}
\]

\[
a \left( \frac{\lambda_0^2}{1 - f_0} + \frac{\lambda_1^2}{1 - f_1} \right) \left[ 1 - a \left( \frac{\lambda_0^2}{1 - f_0} + \frac{\lambda_1^2}{1 - f_1} \right) \right] + \cdots
\]

\[
= \frac{a \left( (\lambda_0^2/(1 - f_0)) + (\lambda_1^2/(1 - f_1)) \right)}{1 - 4\mu^2 \left( \frac{\lambda_0^2}{1 - (1 - 4\mu \lambda_0 + 8\mu^2 \lambda_0^2)} \right) - 4\mu^2 \left( \frac{\lambda_1^2}{1 - (1 - 4\mu \lambda_1 + 8\mu^2 \lambda_1^2)} \right)}
\]

\[
= \frac{\sum_{i=0}^{M-1} (\mu \lambda_i/(1 - 2\mu \lambda_i))}{1 - \sum_{i=0}^{M-1} (\mu \lambda_i/(1 - 2\mu \lambda_i))}
\]

which confirms (8.60) for $M = 2$.

8.4.6 Hint: Take derivatives with respect to $\mu$ and $L$, and note the positive values ($\partial L/\partial \mu$) and ($\partial^2 L/\partial L$).

8.4.7

\[
\lambda_{av} = \frac{1}{M} \sum_{i=0}^{M-1} \lambda_i, \quad \tau_{\text{mse,av}} \approx \frac{1}{2\mu \lambda_{av}} [\text{see (7.27)}]
\]
and, therefore, $\mathbb{M} \cong \mu M \lambda_{av} \cong M/(2\tau_{mse,av})$. Observations are as follows: (1) $\mathbb{M}$ increases linearly with filter length for a fixed $\lambda_{av}$; (2) the setting time of the LMS algorithm is proportional to $\tau_{mse,av}$, and hence, $\mathbb{M}$ is inversely proportional to the settling time; and (3) $\mathbb{M}$ is proportional to $\mu$ and inversely proportional to $\tau_{mse,av}$, but $\mu$ and $\tau_{mse,av}$ are inversely proportional to each other. Careful consideration must be given when we are considering the values for $\mathbb{M}$, $\mu$, and $\tau_{mse,av}$.

**8.4.8** The averaged learning curve of the LMS algorithm is close to the steepest descent method. Hence, we write (7.30) in the form:

$$J(n) \cong J_{\text{min}} + \sum_{k=0}^{M-1} \lambda_k (1-2\mu \lambda_k)^2 \xi_k^2(0)$$

(1)

If we set $\xi(0) = w(n) - w^o = 0 - w^o$, then

$$\xi'(n) = Q^T \xi(0) = -Q^T w^o = -w^o$$

(2)

Hence, (1) becomes

$$J(n) \cong J_{\text{min}} + \sum_{k=0}^{M-1} \lambda_k (1-\mu \lambda_k)^2 w_k^{(o)^2}$$

(3)

Equation 3 indicates that if $w_{k}^{(o)}$'s corresponding to the smaller eigenvalues of $R$ are all close to zero, then the transient behavior of the LMS algorithm is determined by the larger eigenvalues whose associated time constants are small, and thus, a fast convergence takes place. If, however, $w_{k}^{(o)}$'s corresponding to the smaller eigenvalues of the correlation matrix are significantly large, then we will observe that the slower modes will dominate.

**8.4.9** From (7.15), we find $w(n+1) = w(n) - \mu \nabla e^2(n)$ (1), where $n$ is the iteration number, and $\nabla = [(\partial/\partial w_0) (\partial/\partial w_1) \cdots (\partial/\partial w_{M-1})]^T$, $w(n) = [w_0(n) \ w_1(n) \ \cdots \ w_{M-1}(n)]^T$. But

$$\partial e^2(n)/\partial w_i = 2e(n)\partial e(n)/\partial w_i = 2e(n)\partial[d(n) - y(n)]/\partial w_i$$

$$= -2e(n)\partial y(n)/\partial w_i = -2e(n)\partial \left[ \sum_{i=0}^{M-1} w_i(n)x(n-i) \right]/\partial w_i$$

$$= -2e(n)x(n-i)$$
The Least Mean-Square Algorithm

and hence, $\nabla e^2(n) = -2e(n)x(n)$, where $x(n) = [x(n) \ x(n-1) \ x(n-2) \cdots x(n-M+1)]^T$. Therefore, (1) becomes $w(n+1) = w(n) + 2\mu e(n)x(n)$.

8.4.10 See Figure P8.4.10 for the three cases. We observe that for case (3), for example, the step-size parameter must be less than 0.35.
9 Variants of Least Mean-Square Algorithm

9.1 THE NORMALIZED LEAST MEAN-SQUARE ALGORITHM

Consider the conventional least mean-square (LMS) algorithm with the fixed step-size parameter $\mu$ replaced with a time-varying variable $\mu(n)$ as follows (we substitute $2\mu$ with $\mu$ for simplicity):

$$w(n+1) = w(n) + \mu(n)e(n)x(n)$$  \hspace{1cm} (9.1)

Next, define a *posteriori* error, $e_{ps}(n)$, as

$$e_{ps}(n) = d(n) - w^T(n)x(n)$$  \hspace{1cm} (9.2)

Substituting (9.1) in (9.2), and taking into consideration the error equation $e(n) = d(n) - w^T(n)x(n)$, we obtain

$$e_{ps}(n) = \left[1 - \mu(n)x^T(n)x(n)\right]e(n)$$  \hspace{1cm} (9.3)

Minimizing $e_{ps}(n)$ with respect to $\mu(n)$ results in (see Problem 9.1.1)

$$\mu(n) = \frac{1}{\|x(n)\|^2}$$  \hspace{1cm} (9.4)

Substituting (9.4) in (9.1), we find

$$w(n+1) = w(n) + \frac{1}{\|x(n)\|^2}e(n)x(n)$$  \hspace{1cm} (9.5)

However, the most common normalized LMS (NLMS) algorithm is

$$w(n+1) = w(n) + \frac{\mu}{\|x(n)\|^2}e(n)x(n)$$  \hspace{1cm} (9.6)

Note that the NLMS is actually a variable step-size algorithm in which this step size is inversely proportional to the total instantaneous energy of the input signal $x(n)$, estimated over values within the tapped delay line length.

A modified version of the NLMS algorithm (sometimes called $\varepsilon$-NLMS) is given by
\[ w(n+1) = w(n) + \frac{\mu}{\varepsilon + \|x(n)\|^2} e(n)x(n) \] \tag{9.7}

\[ = w(n) + \frac{\mu}{\varepsilon + \|x(n)\|^2} \left[ d(n) - w^T(n)x(n) \right] x(n) \]

In the above equation, \( \varepsilon \) is a small positive number that was introduced to avoid division by zero at a time when the signal becomes zero or extremely small. Table 9.1 gives the LMS and NLMS formulas.

### TABLE 9.1
Some Variants of the LMS Formulas

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS with constant step size</td>
<td>( w(n+1) = w(n) + \mu e(n)x(n) );</td>
</tr>
<tr>
<td></td>
<td>( w(n+1) = w(n) + \mu [d(n) - w^T(n)x(n)]x(n) )</td>
</tr>
<tr>
<td>LMS with time-varying step size</td>
<td>( w(n+1) = w(n) + \mu [d(n) - w^T(n)x(n)]x(n) )</td>
</tr>
<tr>
<td>( \varepsilon )-NLMS</td>
<td>( w(n+1) = w(n) + \frac{\mu}{\varepsilon + |x(n)|^2} [d(n) - w^T(n)x(n)]x(n) )</td>
</tr>
<tr>
<td>( \varepsilon )-NLMS with power normalization</td>
<td>( w(n+1) = w(n) + \frac{\mu}{\varepsilon + p(n+1)} [d(n) - w^T(n)x(n)]x(n) )</td>
</tr>
<tr>
<td></td>
<td>( p(n+1) = ap(n) + (1 - a) |x(n+1)|^2 ) ( p(0) = 0 )</td>
</tr>
</tbody>
</table>

### TABLE 9.2
The Normalized Real and Complex LMS Algorithms

<table>
<thead>
<tr>
<th>Real-Valued Functions</th>
<th>Complex-Valued Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td></td>
</tr>
<tr>
<td>Initialization vector</td>
<td>( w(n) = 0 )</td>
</tr>
<tr>
<td>Input vector</td>
<td>( x(n) )</td>
</tr>
<tr>
<td>Desired output</td>
<td>( d(n) )</td>
</tr>
<tr>
<td>Step-size parameter</td>
<td>( \mu )</td>
</tr>
<tr>
<td>Constant</td>
<td>( \varepsilon )</td>
</tr>
<tr>
<td>Filter length</td>
<td>( M )</td>
</tr>
<tr>
<td><strong>Procedure</strong></td>
<td></td>
</tr>
<tr>
<td>1. ( y(n) = w^T(n)x(n) = w(n)x^T(n) )</td>
<td>1. ( y(n) = w^H(n)x(n) )</td>
</tr>
<tr>
<td>2. ( e(n) = d(n) - y(n) )</td>
<td>2. ( e(n) = d(n) - w^H(n)x(n) )</td>
</tr>
<tr>
<td>3. ( w(n+1) = w(n) + \frac{\mu}{\varepsilon + x^T(n)x(n)} e(n)x(n) )</td>
<td>3. ( w(n+1) = w(n) + \frac{\mu}{\varepsilon + x^H(n)x(n)} e^H(n)x(n) )</td>
</tr>
<tr>
<td></td>
<td>(H = Hermitian, conjugate transpose)</td>
</tr>
</tbody>
</table>
Example 9.1.1 (Noise Cancellation)

Compare the LMS and NLMS formulas for a line enhancer setup shown in Figure 9.1.

Solution: The two Book m-functions are given below.

Book m-Function for Unnormalized LMS

```matlab
function [w,y,x,J]=lms_ex9_1_1_lms(s,mu,M)
    %s=sinusoidal signal;mu=step-size factor;
    %M=length of adaptive filter;
    N=length(s);y=zeros(1,N);w=zeros(1,M);
    e=zeros(1,N);
    x=[0 s+rand(1,N)-0.5];
    for n=M:N
        x1=x(n:-1:n-M+1);
        y(n)=w'*x1;
        d(n+1)=x(n);
        e(n)=d(n+1)-y(n);
        w=w+2*mu*e(n)*x1;
    end;
    J=e.^2;
end;
```

Book m-Function for NLMS

```matlab
function [w,y,x,J,Jav]=lms_ex9_1_1_norm_lms(s,mu,M,c)
    %s=sinusoidal signal;mu=step-size factor;
    %M=length of adaptive filter;c=very small number;
    N=length(s);y=zeros(1,N);w=zeros(1,M);
    e=zeros(1,N);
    x=[d(1,2:N) 0];
    for n=M:N
        x1=x(n:-1:n-M+1);
        y(n)=w'*x1;
        e(n)=d(n+1)-y(n);
        w=w+((2*mu*e(n)*x1)/(c+x1*x1'));
    end;
    J=e.^2;
end;
```

FIGURE 9.1
Figure 9.2 shows the output of the adaptive filter for the LMS and NLMS filters. The constants that were used were as follows: $\mu = 0.01$, $M = 30$, $c = 0.0001$, $s = \sin(0.2\pi n)$. We observe that the NLMS filter output is closer to the sine signal than the LMS filter. The signal $v(n)$ identifies the white noise.

Example 9.1.2

If we desire to plot the learning function $J$, it is recommended to use an average process for a better presentation. The following Book m-function cancels the noise embedded in the signal.

**Book m-Function for Noise Canceling in a Sine Function**

```matlab
function [w, jav, js] = lms_denoising_Jav1(N, mu, M, av, a)
es = zeros(1, N);
for m=1:av%av=integer equal to the number of desired
  %averaging;a=multiplier of the random noise,
  %a*randn(1,N);
w = zeros(1, M);
```

![Figure 9.2](image_url)
Variants of Least Mean-Square Algorithm

\[
dn = \sin(0.1\pi [1:N]) + a \cdot \text{randn}(1,N);
\]
\[
x = [0 0 \ dn(1,1:N-2)]; \quad \text{% delay 2 units;}
\]
for \( n=M:N \)
\[
x1 = x(n:-1:n-M+1);
\]
\[
y(n) = w \cdot x1';
\]
\[
e(n) = dn(n) - y(n);
\]
\[
w = w + \mu \cdot e(n) \cdot x1;
\]
end;

\[
es = es + e;
\]
\[
y = \text{zeros}(1,N);
\]
end;
\[
jav = (es/av)^2;
\]
for \( n=1:N-5 \)
\[
js(n) = (jav(n)+jav(n+1)+jav(n+2)+jav(n+3))/4;
\]
end;

Figure 9.3 was produced using the following constants: \( \mu = 0.005 \), \( M = 40 \), \( av = 1 \) and 300, \( a = 0.5 \), and \( N = 1500 \). The signal was a sine function with a white Gaussian additive noise. The learning curves were smoothed for a better understanding of the variations of the two learning curves. The averaging produces a lower mean-square error as it was expected.

\[\begin{array}{c}
\text{Zero averaging; no. of iterations}
\end{array}\]

\[\begin{array}{c}
\text{300 realization; no. of iterations}
\end{array}\]

FIGURE 9.3
9.2 POWER NORMALIZED LMS

Observe that the NLMS algorithm can be written in the form (we have substituted $2\mu = \mu$ for convenience):

$$w(n+1) = w(n) + \frac{\mu/K}{\varepsilon/K + \|x(n)\|^2/K} \epsilon(n)x(n) \tag{9.8}$$

The form of the above equation suggests a method for approximating the norm square of the data. The method replaces $\|x(n)\|^2/K$ by a variable $p(i)$ which is updated as follows:

$$p(n+1) = bp(n) + (1-b)|x(n)|^2 \quad p(0) = 0 \tag{9.9}$$

with a positive scalar $b$ and values in the range $0 \leq b < 1$. Hence, (9.8) takes the form:

$$w(n+1) = w(n) + \frac{\mu}{\varepsilon + p(n+1)} \epsilon(n)x(n)$$

$$= w(n) + \frac{\mu}{\varepsilon + p(n+1)} \left[ d(n) - w(n)x^T(n) \right] x(n) \tag{9.10}$$

From the above equation, it is understood that the step size of power NLMS (PNLMS) in (9.10) is approximately $M$ times smaller than the step size of the NLMS.

**Example 9.2.1 (Channel Equalization)**

It is desired to estimate a communication channel, shown in Figure 9.4, so that the noisy output signal from the channel space system is denoised by using a PNLMS.

**Solution:** At each instant time $n$, the measured output of the channel, $d(n)$, is compared with the output of the adaptive filter, $y(n)$, and an error signal $e(n) = d(n) - y(n)$, is generated. The error is then used to adjust the filter coefficients according to (9.10). In this case, the following constants and functions were used:

![Figure 9.4](image-url)
Variants of Least Mean-Square Algorithm

$245$

Variants of Least Mean-Square Algorithm

The output of the adaptive filter will assume values close to the input to the communication channel. Therefore, from an input/output point of view, the adaptive filter behaves as the channel. The top of Figure 9.5 shows the output of the channel with additive white noise having zero mean value, $v(n) = \text{rand} - 0.5$. The middle figure presents the output of the adaptive filter and the third presents the MSE versus iteration. The Book m-function that was used to create the figure as follows:

**Book m-Function for PNLMS**

```matlab
function[w,y,yc,x,J]=lms_power_norm_lms(mu,M,b,c)
%mu=step-size factor;
%M=length of adaptive filter;c=very small number;
%yc=conv(x,[filter coefficients])=channel
%output;d=yc+v;v=noise;0<b<1;

n=1:1500;p(M)=0;
N=length(x);y=zeros(1,N);w=zeros(1,M);
yc=conv(x,[0.9 0.5 -0.2]);%conv(x,y)=MATLAB function;
for n=M:N
    x1=x(n:-1:n-M+1);
    p(n+1)=b*p(n)+(1-b)*x1*x1';
    y(n)=w*x1';
    d=yc+v(n);
    w=w+mu*d*x1;
end

%compute the mean square error
J=0;
for n=1:N
    J=J+1/2*(yc(n)-y(n))^2;
end
```

$\text{FIGURE 9.5}$

$$x(n) = 2 \cdot 0.995^n \cdot \sin(0.1\pi n), \mu = 0.004, M = 30,$$

$$b = 0.5, c = 0.0001, \text{Channel} = \text{FIR \ [0.9 \ 0.5 \ -0.2]}$$
\[ e(n) = (y_c(n) + \text{rand} - 0.5) - y(n); \]
\[ w = w + \frac{(2 \mu e(n) x_1)}{(c+p(n+1))}; \]
\[ J = \left(\frac{e}{\max(e)}\right)^2; \]

We also observe that if \( b \) in (9.9) is zero, then the Book m-function above becomes the NLMS m-function.

---

**Example 9.2.2 (Channel Equalization)**

Figure 9.6a shows a baseband data transmission system equipped with an adaptive channel equalizer and a training system. The signal \( \{s(n)\} \) transmitted through the communication channel is amplitude- or phase-modulated pulses. The communication channel distorts the signal, the most important one is the pulse spreading, and results in overlapping of pulses, thus creating the **intersymbol interference** phenomenon. The noise \( v(n) \) further deteriorates the fidelity of the signal. It is ideally required that the output of the equalizer is the signal \( s(n) \). Therefore, an initialization period is used during which the transmitter sends a sequence of training symbols that are known to the receiver (**training mode**). This approach is satisfactory if the channel does not change the characteristics rapidly in time. However, for slow changes the output from the channel can be treated as the desired signal for further adaptation of the equalizer so that its variations can be followed (**decision-directed mode**).
If the equalization filter is the inverse of the channel filter, $W(z) = 1/H(z)$, the output will be that of the input to the channel, assuming, of course, that noise is small. To avoid singularities from the zero of the channel transfer function inside the unit circle, we select an equalizer such that $W(z)H(z) \approx z^{-L}$. This indicates that the output of the filter $W(z)$ is that of the input to the channel shifted by $L$ units of time. Sometimes, more general filters of the form $Y(z)/H(z)$ are used, where $Y(z) \neq z^{-L}$. These systems are known as the partial-response signaling systems. In these cases, $Y(z)$ is selected such that the amplitude spectra are about equal over the range of frequencies of interest. The result of this choice is that $W(z)$ has a magnitude response of about one, thereby minimizing the noise enhancement.

Figure 9.6b shows a channel equalization problem at the training stage. The channel noise $v(n)$ is assumed to be white Gaussian with variance $\sigma_v^2$. The equalizer is an $M$-tap finite impulse response (FIR) filter and the desired output is assumed to be delayed replica of the signal $s(n), s(n-L)$. The signal $s(n)$ is white, has variance $\sigma_s^2 = 1$, has zero mean value, is made up of random +1 and −1 and is uncorrelated with $v(n)$. The channel transfer functions were assumed to be of FIR form. In this example, we used the following transfer functions:

$$H(z) = H_1(z) = 1$$
$$H(z) = H_2(z) = 0.9 + 0.4z^{-1} - 0.2z^{-2}$$

The following Book m-function produced Figure 9.7.
Book m-Function for Channel Equalization

```matlab
function [Jav,wav,dn,e,x] = lms_equalizer_ex9_2_2(av,M,L,h,N,mu,c)
    %function [Jav,wav,dn,e,x] = lms_equalizer_ex9_2_2
    % (av,M,L,h,N,mu,c)
    % this function solves the example depicted
    % in Fig 9.7; av=number of times to average e(error or
    % learning curve); w(filter coefficient); N=length
    % of signal s; L=shift of the signal s to become dn;
    % h=assumed impulse response of the channel system;
    % mu=step-size factor; M=number of adaptive filter coefficients;
    % c=constant multiplier;
    wn=zeros(1,M);
    J=zeros(1,N);
    for i=1:av
        for n=1:N
            v(n)=c*randn;
            s(n)=(rand-0.5);
            if s(n)<=0
                s(n)=-1;
            else
                s(n)=1;
            end;
        end;
        dn=[zeros(1,L)s(:,1:N-L)];
        ych=filter(h,1,s);
        x=ych(1,1:N)+v;
        [w,y,e,J,w1,Js]=lms1(x,dn,mu,M);
        wn=wn+w;
        J=J+e.^2;
    end;
    Jav=J/av;
    wav=wn/av;
```

The constants used to create Figure 9.7 were as follows: \( av = 300, \) \( M = 10, \) \( L = 1, \) \( h = [0.9 \quad 0.4 \quad -0.2], \) \( N = 2000, \) \( c = 0.05. \)

9.3 SELF-CORRECTING LMS FILTER

We can arrange the standard LMS filter in a series form as shown in Figure 9.8. This book proposed configuration LMS filtering permits us to process the signal using filters with fewer coefficients, thus saving in computation time. Figure 9.9a shows the desired signal with noise. Figure 9.9b shows the output of the first stage. Figure 9.9c shows the output of the third stage of the self-correcting LMS filter (SCLMSF), \( \{ x_1(n) \}. \) Each stage LMS filter has four coefficients. We also used the following signals and constants: \( s = \sin(0.1n\pi), v = [\text{rand}(1,2000) - 0.5], x(n) = dn(n+2) \) (delay by 2 units), \( I = 3, \) \( mu = 0.005, \) \( M = 4. \) Problem 9.3.1 solves for the SCNLMSF.

The output of the \( i^{th} \) stage is related to the previous one as follows:

\[
x_{i+1}(n) = x_i(n) * w_{i+1}(n)
\]
Variants of Least Mean-Square Algorithm

Book m-Function for SCLMS Algorithm

\begin{verbatim}
function [w,y,e,J] = lms_self_correcting_lms(x,dn,mu,M,I)
    %function [w,y,e,J] = lms_self_correcting_lms(x,dn,mu,M,I);
    [w(1,:),y(1,:),e(1,:)] = lms1(x,dn,mu,M);
    for i = 2 : I %I=number of iterations, I<8-10 is sufficient;
        v(n) = x(n) + w(1,i-1) * x(n);
        x1(n) = x(n) + e(1,i-1);
        e1(n) = d(n) - x1(n);
        e1(n) = e1(n) + v(n);
        w(1,i) = w(1,i-1) + mu * e1(n) * x(n);
        [w(2,:),y(2,:),e(2,:)] = lms2(x2,n,mu,M);
        x2(n) = x2(n) + e(2,i-1);
        e2(n) = d(n) - x2(n);
        e2(n) = e2(n) + v(n);
        w(2,i) = w(2,i-1) + mu * e2(n) * x2(n);
        [w(3,:),y(3,:),e(3,:)] = lms3(x3,n,mu,M);
        x3(n) = x3(n) + e(3,i-1);
        e3(n) = d(n) - x3(n);
        e3(n) = e3(n) + v(n);
        w(3,i) = w(3,i-1) + mu * e3(n) * x3(n);
    end
end
\end{verbatim}

\textbf{FIGURE 9.8}

\textbf{FIGURE 9.9}
Adaptive Filtering

\[ [w(i,:), y(i,:), e(i,:)]=\text{lms1}(y(i-1,:), dn, mu, M); \]
end;
\[ J=e.^{\cdot2}; \]

9.4 THE SIGN-ERROR LMS ALGORITHM

The sign-error algorithm is defined by

\[ w(n+1) = w(n) + \mu \text{sign}[e(n)]x(n) \]  \hspace{1cm} (9.12)

where:

\[
\text{sign}(a) = \begin{cases} 
1 & a > 0 \\
0 & a = 0 \\
-1 & a < 0
\end{cases}
\]  \hspace{1cm} (9.13)

is the signum function. By introducing the signum function and setting \( \mu \) to a value of power of 2, the hardware implementation is highly simplified (shift and add/subtract operations only).

**Book m-Function for Sign-Error Algorithm**

```matlab
function [w, y, e, J, w1] = lms_sign_error(x, dn, mu, M)
%function [w, y, e, J, w1] = lms_sign_error(x, dn, mu, M);
%all quantities are real valued;
%x=input data to the filter;dn=desired signal;
%M=order of the filter;
%mu=step size parameter;
%x and dn must be of the same length;
N = length(x);
y = zeros(1,N);
w = zeros(1,M); %initialized filter coefficient vector;
for n = M:N
    x1 = x(n:-1:n-M+1); %for each n the vector x1 is produced
        %of length M with elements from x in reverse order;
    y(n) = w*x1';
    e(n) = dn(n) - y(n);
    w = w + mu*sign(e(n))*x1;
    w1(n-M+1,:) = w(1,:);
end;
J = e.^2;
%the columns of w1 depict the history of the filter
%coefficients;
```

9.5 THE NLMS SIGN-ERROR ALGORITHM

The NLMS sign-error algorithm is

\[ w(n+1) = w(n) + \mu \frac{\text{sign}[e(n)]x(n)}{\varepsilon + \|x(n)\|^2} \]
\[ \|x(n)\|^2 = x(n)^T x(n) \]  \hspace{1cm} (9.14)
The Book m-function for the NLMS sign-error algorithm is given below. The m-file `sign()` is a MATLAB function (see MATLAB functions).

**Book m-Function for NLMS Sign-Error Algorithm**

```matlab
function [w,y,e,J,w1]=lms_normalized_sign_error(x,dn,mu,M,ep)
%function [w,y,e,J,w1]=lms_normalized_sign_error
%(x,dn,mu,M,ep);
%all quantities are real valued;
%x=input data to the filter;dn=desired signal;
%M=order of the filter;
%mu=step size parameter;x and dn must be of the same
%length;
%ep=sm
N=length(x);
y=zeros(1,N);
w=zeros(1,M); %initialized filter coefficient vector
for n=M:N
    x1=x(n:-1:n-M+1); %for each n the vector x1 is produced
        %of length M with elements from x in reverse order;
    y(n)=w*x1';
    e(n)=dn(n)-y(n);
    w=w+2*mu*sign(e(n))*x1/(ep+x1*x1');
    w1(n-M+1,:)=w(1,:);
end;
J=e.^2;
%the columns of w1 depict the history of the filter
%coefficients;
```

**Example 9.5.1**

Compare the sign-error and NLMS sign-error algorithms for denoising a sinusoidal signal as shown in Figure 9.10.

**Solution:** The results are shown in Figure 9.11 using the following constants and functions: \( \mu = 0.005, \quad M = 15, \quad \varepsilon = 0.0001, \quad s = \sin(0.1\pi n), \quad x = \sin(0.1\pi n) + \text{rand}(n) - 0.5, \quad x(n) = dn(n+1), \quad n = 1:1500. \)

We observe that for the same inputs the NLMS sign-error algorithm is superior to unnormalized one.

![Figure 9.10](image-url)
### 9.6 THE SIGN-REGRESSOR LMS ALGORITHM

The sign-regressor or data-sign algorithm is given by

\[
\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n) \text{sign}[\mathbf{x}(n)]
\]  

(9.15)

where the sign function is applied to \( \mathbf{x}(n) \) on an element-by-element basis.

#### Book m-Function for Sign-Regressor Algorithm

```matlab
function [w,y,e,J,w1]=lms_sign_regressor(x,dn,mu,M)
    %function [w,y,e,J,w1]=lms_sign_error(x,dn,mu,M);
    % all quantities are real valued;
    % x=input data to the filter; dn=desired signal;
    % M=order of the filter;
    % mu=step size parameter;
    % x and dn must be of the same length;
    N=length(x);
    y=zeros(1,N);
    w=zeros(1,M); % initialized filter coefficient vector;
    for n=M:N
        x1=x(n:-1:n-M+1); % for each n the vector x1 is produced
        % of length M with elements from x in reverse order;
        y(n)=w*x1';
```

![Figure 9.11](image-url)
\[ e(n) = d(n) - y(n) \]
\[ w = w + \mu e(n) \cdot \text{sign}(x_1) \]
\[ w_1(n-M+1,:) = w(1,:) \]

\[ J = e^2 \]

%the columns of w1 depict the history of the filter coefficients;

9.7 SELF-CORRECTING SIGN-REGRESSOR LMS ALGORITHM

Book m-Function for Self-Correcting Sign-Regressor LMS Algorithm

function \[ w, y, e, J \] = lms_self_correcting_sign_regressor_ 
% lms(x,dn,mu,M,I)
% function [w,y,e,J]=lms_self_correcting_sign_regressor_lms(x,dn,mu,M,I);
% x=input data to the filter;dn=desired signal;length(x)=length(dn);
% y=output of the filter, an Ixlength(x) matrix;J=error function, an Ixlength(x) matrix;I=number of stages;
[ w(1,:), y(1,:), e(1,:), J(1,:) ] = lms_sign_regressor_lms(x, dn, mu, M);
for i = 2 : I
[ w(i,:), y(i,:), e(i,:), J(i,:) ] = lms1(y(i-1,:), dn, mu, M);
end;
J = e .^ 2;

Figure 9.12 was obtained using the following signals and constants: \( s = \sin(0.2\pi n) \), \( v = 1.2[\text{rand}(1,2000)-0.5] \), \( x = s + v \), delay by 2 units, \( \mu = 0.01 \), \( M = 2 \), \( I = 4 \). The reader should observe the power of this Book proposed technique. The adaptive filters have only two coefficients, and by repeating this process only three times, it gives excellent results.

9.8 THE NORMALIZED SIGN-REGRESSOR LMS ALGORITHM

The normalized sign-regressor is given by
\[ w(n+1) = w(n) + \mu \frac{e(n) \cdot \text{sign}(x(n))}{1 + \| x(n) \|^2}, \quad \| x(n) \|^2 = x(n)^T x(n) \] (9.16)

Book m-Function for Normalized Sign-Regressor LMS Algorithm

function \[ w, y, e, J, w1 \] = lms_normalized_sign_regressor(x, dn, mu, M)
% function [w,y,e,J,w1]=lms_sign_error(x,dn,mu,M);
% all quantities are real valued;
% x=input data to the filter;dn=desired signal;
% M=order of the filter;
% mu=step-size parameter;
% x and dn must be of the same length;
N = length(x);
y = zeros(1,N);
Adaptive Filtering

w=zeros(1,M);%initialized filter coefficient vector;
for n=M:N
    x1=x(n:-1:n-M+1);%for each n the vector x1 is produced
    %of length M with elements from x in reverse order;
    y(n)=w*x1';
    e(n)=dn(n)-y(n);
    w=w+mu*e(n)*sign(x1)./(0.0001+x1*x1');
    w1(n-M+1,:)=w(1,:);
end;
J=e.^2;
%the columns of w1 depict the history of the filter
%coefficients;

9.9 THE SIGN–SIGN LMS ALGORITHM

The sign–sign LMS algorithm is defined by

\[ w(n+1) = w(n) + \mu \text{sign}[e(n)]\text{sign}[x(n)] \]  

(9.17)

Book m-Function for Sign–Sign LMS Algorithm

function [w, y, e, J, w1]=lms_sign_sign(x, dn, mu, M)
    %function [w, y, e, J, w1]=lms_sign_sign(x, dn, mu, M)
    %all quantities are real valued;
    %x=input data to the adaptive filter;

FIGURE 9.12

First-stage output, y

Third-stage output, y

First-stage output, J

Third-stage output, J

No. of iterations

No. of iterations

No. of iterations

No. of iterations
Variants of Least Mean-Square Algorithm

\$dn=\text{desired signal;}\$
\$M=\text{order of the filter;}\$
\$\mu=\text{step-size parameter;}x \text{ and } dn \text{ must be of} \$
\$\text{the same length;}\$

\$N=\text{length}(x);\$
\$y=\text{zeros}(1,N);\$
\$w=\text{zeros}(1,M);\%\text{initialized filter coefficient vector}\$
\$\text{for } n=M:N\$
\$x_1=x(n:-1:n-M+1);\%\text{for each } n \text{ the vector } x_1 \text{ is produced}\$
\$\text{of length } M \text{ with elements from } x \text{ in reverse order;}\$
\$y(n)=w'x_1;\$
\$e(n)=dn(n)-y(n);\$
\$w=w+2*\mu*\text{sign}(e(n))*\text{sign}(x_1);\$
\$w(1:n-M+1,:)=w(1,:);\$
\$\text{end;}\$
\$J=e.^2;\$
\$\text{the columns of } w_1 \text{ depict the history of the filter}\$
\$\text{coefficients;}\$

**Book m-Function for Self-Correcting Sign–Sign LMS Algorithm**

\[w, y, e, J]=\text{lms\_self\_correcting\_sign\_sign\_lms}(x, dn, \mu, M, I)\]

\[w, y, e, J]=\text{lms\_self\_correcting\_sign\_sign\_lms}(x, dn, \mu, M, I)\]

\[x=\text{input data to the filter;} y=\text{output}\]
\[\text{data from the filter,}\]
\[y \text{ is an } I\times\text{length}(x) \text{ matrix;} J=\text{learning curves,}\]
\[\text{an } I\times\text{length}(x) \text{ matrix;} \mu=\text{step-size parameter;}\]
\[M=\text{number of coefficients;}\]
\[I=\text{number of stages;} w=\text{an } I\times\text{length}(x) \text{ matrix of filter}\]
\[\text{coefficients;} dn=\text{desired signal;}\]

\[\text{for } i=2:1\]
\[\text{[}w(i,:), y(i,:), e(i,:), J(i,:)] = \text{lms\_sign\_sign}(y(i-1,:), dn, \mu, M)\]
\[\text{end;}\]
\[J=e.^2;\]

**9.10 THE NORMALIZED SIGN–SIGN LMS ALGORITHM**

The sign–sign LMS algorithm is defined by

\[w(n+1)=w(n)+\mu \frac{\text{sign}(e(n))\text{sign}(x(n))}{\varepsilon + \|x(n)\|^2} x'(n)x(n) \quad (9.18)\]

**Book m-Function for Normalized Sign–Sign LMS Algorithm**

\[w, y, e, J, w_1]=\text{lms\_normalized\_sign\_sign}(x, dn, \mu, M)\]

\[w, y, e, J, w_1]=\text{lms\_normalized\_sign\_sign}(x, dn, \mu, M)\]
\[\text{all quantities are real valued;}\]
\[x=\text{input data to the adaptive filter;}\]
Adaptive Filtering

Adaptive Filtering

% $dn=$ desired signal;
% $M=$ order of the filter;
% $\mu =$ step-size parameter; $x$ and $dn$ must be of
% the same length;

$N = \text{length}(x);$  
$y = \text{zeros}(1,N);$  
$w = \text{zeros}(1,M);$ % initialized filter coefficient vector;
for $n = M:N$

$x_1 = x(n:-1:n-M+1);$ % for each $n$ the vector $x_1$ is produced
% of length $M$ with elements from $x$ in reverse order;

$y(n) = w'x_1;$
$e(n) = dn(n) - y(n);$  
$w = w + 2\mu \text{sign}(e(n)) \cdot \text{sign}(x_1) / (0.0001 + x_1 \cdot x_1');$
$w_1(n-M+1,:) = w(1,:);$  
end;

$J = e.^2;$  
% the columns of $w_1$ depict the history of the filter
% coefficients;

Figure 9.13 compares different types of sign-type LMS algorithm. The signal to be
detected at the output of the LMS filter is a sine wave and it is shown in Figure 9.13a.
The noisy signal is shown in Figure 9.13b. To recapture the sine wave from the noisy,
\{ $x(n)$ \}, signal, we use the setup shown in Figure 9.1. The constants and functions that
were used in this example are as follows:

![Figure 9.13](image-url)
Variants of Least Mean-Square Algorithm

\[ n = 1 : 2000; \quad s = \sin(0.2n\pi); \quad x = s + v = s + 0.5\text{randn}(1,2000) \]

\[ \sigma_r^2 = 0.24; \quad \mu = 0.002; \quad M = 20; \quad \text{delay} = 2 (= z^{-2}) \]

Figure 9.13c shows the output of the sign-regressor LMS filter. Figure 9.13d shows the output of a normalized sign-regressor LMS filter. Figure 9.13e shows the output of a sign–sign LMS filter. Figure 9.13f shows the output of a normalized sign–sign LMS filter. It is apparent that the normalized ones perform better.

**Note:** The self-correcting approach can be used with any one of the LMS algorithms and their normalized form.

### 9.11 VARIABLE STEP-SIZE LMS

The variable step-size LMS (VSLMS) algorithm was introduced to facilitate the conflicting requirements. A large step-size parameter is needed for fast convergence, whereas a small step-size parameter is needed to reduce the misadjustment factor. When the adaptation begins and \( w(n) \) is far from its optimum value, the step-size parameter should be large in order for the convergence to be rapid. However, as the filter value \( w(n) \) approaches the steady-state solution, the step-size parameter should decrease in order to reduce the excess MSE.

To accomplish the variation of the step-size parameter, each filter coefficient is given a separate time-varying step-size parameter such that the LMS recursion algorithm takes the form:

\[
w_i(n+1) = w_i(n) + \mu_i(n)e(n)x(n-i) \quad i = 0, 1, \ldots, M-1 \quad (9.19)
\]

where:

- \( w_i(n) \) is the \( i \)th coefficient of \( w(n) \) at iteration \( n \)
- \( \mu_i(n) \) is its associated step size

The step sizes are determined in an *ad hoc* manner, based on monitoring sign changes in the instantaneous gradient estimate \( e(n)x(n-i) \). It was argued that successive changes in the sign of the gradient estimate indicates that the algorithm is close to its optimal solution, and hence, the step-size value must be decreased. The reverse is also true. The decision of decreasing the value of the step size by some factor \( c_1 \) is based on some number \( m_1 \) successive changes of \( e(n)x(n-i) \). Increasing the step-size parameter by some factor \( c_2 \) is based on \( m_2 \) successive sign changes. The parameters \( m_1 \) and \( m_2 \) can be adjusted to optimize performance, as can the factors \( c_1 \) and \( c_2 \).

The set of update (9.19) may be written in the matrix form:

\[
\begin{bmatrix}
\mu_0(n) \\
\vdots \\
\mu_{M-1}(n)
\end{bmatrix} + \begin{bmatrix}
0 \\
\vdots \\
0
\end{bmatrix} = \begin{bmatrix}
\mu_0(n) \\
\vdots \\
\mu_{M-1}(n)
\end{bmatrix}
\]

(9.20)
The VSLMS algorithm is given in Table 9.3.

The following proposed Book m-function can be used to take care, for example, of a noisy function whose added noise is varying. Figure 9.14 shows the results using the constant value of the step-size value and a varying one. The constants and functions used were as follows:

\[ n = 2000, \quad n = 1: 2000, \quad dn = \sin(0.2\pi n) + 1.50.988^n \text{randn}(1,2000), \]
\[ x(n) = dn(n+2), \text{delay by 2 units, } mu = 0.05, \text{ for the constant step-size case} \]

Book m-Function for Varying Variance of the Signal That Adjusts Appropriately the Value of \( \mu \)

```matlab
function [w, y, e, J, w1] = lms_varying_mu(x, dn, M, N)
w = zeros(1, M); \%N=length of x and dn signals;
for n=M:N
    x1=x(n:-1:n-M+1);
    if var(x1) >= 0.6
        mu = 0.0005;
    end
```

TABLE 9.3
The VSLMS Algorithm

<table>
<thead>
<tr>
<th>Input</th>
<th>Initial coefficient vector: ( w(0) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input data vector: ( x(n) = [x(n)x(n-1)\cdots x(n-M+1)]^T )</td>
</tr>
<tr>
<td></td>
<td>Gradient term: ( g_0(n-1) = e(n-1)x(n-1), g_1(n-1) = e(n-1)x(n-1), \ldots, )</td>
</tr>
<tr>
<td></td>
<td>( g_{M-1}(n-1) = e(n-1)x(n-M) )</td>
</tr>
<tr>
<td></td>
<td>Step-size parameter: ( \mu_0(n-1), \mu_1(n-1), \ldots, \mu_{M-1}(n-1) )</td>
</tr>
<tr>
<td></td>
<td>( a ) = small positive constant</td>
</tr>
<tr>
<td></td>
<td>( \mu_{\text{max}} ) = positive constant</td>
</tr>
<tr>
<td>Outputs</td>
<td>Desired output: ( d(n) )</td>
</tr>
<tr>
<td></td>
<td>Filter output: ( y(n) )</td>
</tr>
<tr>
<td></td>
<td>Filter update: ( w(n+1) )</td>
</tr>
<tr>
<td></td>
<td>Gradient term: ( g_0(n), g_1(n), \ldots, g_{M-1}(n) )</td>
</tr>
<tr>
<td></td>
<td>Update step-size parameter: ( \mu_0(n), \mu_1(n), \ldots, \mu_{M-1}(n) )</td>
</tr>
<tr>
<td>Execution</td>
<td>1. ( y(n) = w^T(n)x(n) )</td>
</tr>
<tr>
<td></td>
<td>2. ( e(n) = d(n) - y(n) )</td>
</tr>
<tr>
<td></td>
<td>3. Weights and step-size parameter adaptation</td>
</tr>
<tr>
<td></td>
<td>For ( i = 0, 1, 2, \ldots, M-1 )</td>
</tr>
<tr>
<td></td>
<td>( g_i(n) = e(n) \times (n-i) )</td>
</tr>
<tr>
<td></td>
<td>( \mu_i(n) = \mu_i(n-1) + \sigma \text{sign}(g_i(n)) \text{sign}(g_i(n)) % \sigma = \text{small positive step-size parameter}; )</td>
</tr>
<tr>
<td></td>
<td>if ( \mu_i(n) &gt; \mu_{\text{max}}, \mu_i(n) = \mu_{\text{max}} )</td>
</tr>
<tr>
<td></td>
<td>if ( \mu_i(n) &lt; \mu_{\text{min}}, \mu_i = \mu_{\text{min}} )</td>
</tr>
<tr>
<td></td>
<td>( w_i(n+1) = w_i(n) + \mu_i(n)g_i(n) )</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
</tbody>
</table>
Variants of Least Mean-Square Algorithm

```matlab
elseif 0.6<var(x1)<0.2
    mu=0.005;
elseif var(x1)<=0.2
    mu=0.05;
end
y(n)=w*x1';
e(n)=dn(n)-y(n);
w=w+mu*e(n)*x1;
w1(n-M+1,:)=w(1,:);
end;
J=e.^2;
```

9.12 THE LEAKY LMS ALGORITHM

Let us assume that \( \{ x(n) \} \) and \( \{ d(n) \} \) are jointly wide-sense stationary processes that determine when the coefficients \( w(n) \) converge in the mean to \( w^o = R_x^{-1}p_{dx} \).

That is,

\[
\lim_{n \to \infty} E[w(n)] = w^o = R_x^{-1}p_{dx} \tag{9.21}
\]

We start by taking the expectation of both sides of the LMS recursion as follows:

\[
E[w(n+1)] = E[w(n)] + \mu E[d(n)x(n)] - \mu E[x(n)x^T(n)w(n)] \tag{9.22}
\]
where:
\[ y(n) = x^T(n)w(n) \]

Assuming that \( x(n) \) and \( w(n) \) are statistically independent (independence assumption), (9.22) becomes

\[
E \{ w(n+1) \} = E \{ w(n) \} + \mu E \{ d(n)x(n) \} - \mu E \{ x(n)x^T(n) \} E \{ w(n) \} \\
= \left( I - \mu R \right) E \{ w(n) \} + \mu p_{dx}(n)
\]

which is similar to the equations of the steepest-descent method [see (7.18)], where we have set \( \mu' = \mu \) for simplicity. The difference in the above equation is the presence of the ensemble average symbol. This suggests that the steepest descent method is applicable to ensemble average \( E\{w(n+1)\} \). Rewriting (7.23) in matrix form, we obtain

\[
\xi'(n) = (I - \mu A)^n \xi'(0) \quad k = 0, 1, \ldots, M - 1 \quad n = 1, 2, 3, \ldots \tag{9.24}
\]

We observe that \( w(n) \to w' \) if \( \xi(n) = w(n) - w' \to 0 \) as \( n \to \infty \) or when \( \xi(n) = Q^T \xi \) converges to zero. The \( k \)th row of (9.24) is

\[
\xi'_k(n) = (1 - \mu \lambda_k)^n \xi'_k(0)
\]

which indicates that \( \xi'_k(n) \to 0 \) if

\[
|1 - \mu \lambda_k| < 1 \quad \text{or} \quad -1 < 1 - \mu \lambda_k < 1 \quad \text{or} \quad 0 < \mu < \frac{2}{\lambda_k}
\]

To have more restrictive condition, we can use the inequality

\[
0 < \mu < \frac{1}{\lambda_{\max}} \tag{9.27}
\]

If \( \lambda_k = 0 \), (9.25) indicates that no convergence takes place as \( n \) approaches infinity. Since it is possible for these undamped modes to become unstable, it is important for the stabilization of the LMS algorithm to force these modes to zero. One way to avoid this difficulty is to introduce a leakage coefficient \( \gamma \) into the LMS algorithm as follows:

\[
w(n+1) = (1 - \mu \gamma)w(n) + \mu e(n)x(n) \tag{9.28}
\]

where:

\[ 0 < \gamma \ll 1 \]

The effect of introducing the leakage coefficient \( \gamma \) is to force any undamped modes to become zero and to force the filter coefficients to zero if either \( e(n) \) or \( x(n) \) is zero. [The homogeneous equation \( w_i(n+1) = (1 - 2 \mu \gamma)w_i(n) \) has the solution \( w_i(n) = A(1 - \mu \gamma)^n \), where \( A \) is a constant.]

We write (9.28) in the form: \( e(n) = d(n) - x^T(n)w(n) \).
Variants of Least Mean-Square Algorithm

\[ w(n+1) = (1 - \mu \gamma)w(n) + \mu[d(n) - x^T(n)w(n)]x(n) \]
\[ = w(n) - \mu \gamma w(n) + \mu d(n)x(n) - x^T(n)w(n)x(n) \]
\[ = [I - \mu(x(n)x^T(n) + \gamma I)]w(n) + \mu d(n)x(n) \]

(9.29)

We set \( x^T(n)w(n) = x(n)x^T(n)w(n) \) because \( x^T(n)w(n) \) is a number. By taking the expected value of both sides of the above equation and using the independence assumption, we obtain

\[ E[w(n+1)] = [I - \mu E((x(n)x^T(n)) + \gamma I)]E[w(n)] + \mu E[d(n)x(n)] \]
\[ = [I - \mu(R_x + \gamma I)]E[w(n)] + \mu p_{ds}(n) \]

(9.30)

Comparing the above equation with (9.23), we observe that the autocorrelation matrix \( R_x \) of the LMS algorithm has been replaced with \( R_x + \gamma I \). Since the eigenvalues of \( R_x + \gamma I \) are \( \lambda_k + \gamma \) and \( \lambda_k \geq 0 \), all the modes of the leaky LMS algorithm will be decayed to zero. Furthermore, the constraint for the step-size parameter becomes

\[ 0 < \mu < \frac{1}{\lambda_{\text{max}} + \gamma} \]

(9.31)

As \( n \to \infty, w(n+1) \cong w(n) \), and hence, (9.30) becomes

\[ \lim_{n \to \infty} E[w(n)] = \left[ R_x + \gamma I \right]^{-1} p_{ds} \]

(9.32)

which indicates that the leakage coefficient produces a bias into the steady-state solution \( R_x^{-1} p_{ds} \). For another way to produce the leaky LMS algorithm, see Problem 9.11.1.

**Book m-Function for Leaky LMS Algorithm**

```matlab
function [w, y, e, J, w1] = lms_leaky_lms(x, dn, mu, gama, M)

%function [w, y, e, J, w1] = lms_leaky_lms(x, dn, mu, gama, M);
%all signals are real valued;x=input to filter;
%y=output from the filter;dn=desired signal;
%mu=step-size factor;gama=gamma factor<<1;
%M=number of filter coefficients;w1=matrix whose M
%rows give the history of each filter coefficient;

N = length(x);
y = zeros(1, N);
w = zeros(1, M);
for n = M:N
    x1 = x(n:-1:n-M+1);
y(n) = w*x1';
e(n) = dn(n) - y(n);
w = (1 - mu*gama)*w + mu*e(n)*x1;
w1(n-M+1,:) = w(1,:);
end;
J = e.^2;
end
```
Figure 9.15 was produced with the following constants and functions: $s(n) = \sin(0.1\pi n)$, $v(n) = 1.2 \left\{ \text{rand}(1,2000) - 0.5 \right\}$, $d_n = s + v$, delay for $x$ was 2, $d_n(n+2) = x(n)$, $\mu = 0.005$, $M = 15$, $\gamma = 0.01$.

For the normalized leaky LMS case, see Problem 9.11.2.

9.13 THE LINEARLY CONSTRAINED LMS ALGORITHM

In all previous analyses of the Wiener filtering problem, the steepest descent method, the Newton’s method, and the LMS algorithm, no constraint was imposed on the solution of minimizing the MSE. However, in some applications, there might be some mandatory constraints that must be taken into consideration in solving optimization problems. For example, the problem of minimizing the average output power of a filter, while the frequency response must remain constant at specific frequencies. In this section, we discuss the filtering problem of minimizing the MSE subject to a general constraint.

The error between the desired signal and the output of the filter is

$$e(n) = d(n) - w^T(n)x(n)$$

We then minimize this error in the mean-square sense subject to the constraint:

$$c^T w = a$$
where:
- \( a \) is a constant
- \( c \) is a constant vector

Using the Lagrange multiplier method (see also Appendix 4), we write

\[
J_c = E\{e^2(n)\} + \lambda (c^T w - a)
\]

where:
- \( \lambda \) is the Lagrange multiplier

Therefore, the following relations must be satisfied simultaneously:

\[
\nabla_w J_c = 0 \quad \frac{\partial J_c}{\partial \lambda} = 0
\]

The second term produces the constraint (9.34). Next, we substitute the error \( e(n) \) in (9.35) to obtain (see Problem 9.12.1)

\[
J_c = J_{\text{min}} + \xi^T R_c \xi + \lambda (c^T \xi - d')
\]

where:
- \( \xi(n) = w(n) - w^o \), \( w^o = R_c^{-1} p_{dx}\), \( R_c = E\{x(n)x^T (n)\} \)

and

\[
p_{dx} = E\{d(n)x^T (n)\} \quad d' = a - c^T w^o
\]

The solution has now changed to \( \nabla_\xi J_c = 0 \) and \( \partial J_c / \partial \lambda = 0 \). Hence, from (9.37), we obtain (see also Appendix 2)

\[
\nabla_\xi J_c = \begin{bmatrix}
\frac{\partial J_c}{\partial \xi_1} \\
\vdots \\
\frac{\partial J_c}{\partial \xi_M}
\end{bmatrix} = \begin{bmatrix}
2\xi_1 r_1 + 2\xi_2 r_2 + \cdots + 2\xi_M r_M \\
\vdots \\
2\xi_1 r_M + 2\xi_2 r_{M-1} + \cdots + 2\xi_M r_1
\end{bmatrix} + \lambda \begin{bmatrix}
c_1 \\
\vdots \\
c_M
\end{bmatrix} = 0
\]

or in a matrix form

\[
2R_c \xi^o_c + \lambda c = 0
\]

where:
- \( \xi^o_c \) is the constraint optimum of the vector \( \xi \)

In addition, the constraint gives the relation:

\[
\frac{\partial J_c}{\partial \lambda} = c^T \xi^o_c - a' = 0
\]

Solving the system of the last two equations for \( \lambda \) and \( \xi^o_c \), we obtain

\[
\lambda = \frac{2a'}{c^T R_c^{-1} c} \quad \xi^o_c = \frac{a' R_c^{-1} c}{c^T R_c^{-1} c}
\]
Substituting the value of $\lambda$ in (9.37), we obtain the minimum value of $J_c$ as

$$J_c = J_{\min} + \frac{\alpha^2}{c^TR_c^{-1}c}$$  \hspace{1cm} (9.44)

But $w(n) = \xi(n) + w^o$, and hence, using (9.43) we obtain the equation:

$$w^o = w^o + \frac{\alpha R_c^{-1}c}{c^TR_c^{-1}c}$$  \hspace{1cm} (9.45)

**Note:** The second term of (9.44) is the excess MSE produced by the constraint.

To obtain the recursion relation subject to constraint $c^Tw = a$, we must proceed in two steps:

- **Step 1:** $w'(n) = w(n) + \mu e(n)x(n)$  \hspace{1cm} (9.46)
- **Step 2:** $w(n+1) = w'(n) + \eta(n)$  \hspace{1cm} (9.47)

where $\eta(n)$ is chosen so that $c^Tw(n+1) = a$ while $\eta^T(n)\eta(n)$ is minimized. In other words, we choose the vector $\eta(n)$ so that the constraint $c^Tw = a$ after Step 2 while the perturbation introduced by $\eta(n)$ is minimized. The problem can be solved using the Lagrange multiplier method that gives

$$\eta(n) = \frac{a - c^Tw'(n)}{c^Tc}$$  \hspace{1cm} (9.48)

Thus, we obtain the final form of (9.47) as

$$w(n+1) = w'(n) + \frac{a - c^Tw'(n)}{c^Tc}$$  \hspace{1cm} (9.49)

The constraint algorithm is given in Table 9.4. Figure 9.16 shows the results of a constrained LMS filter with the following data: $s = \sin(0.1\pi n)$, $v = \text{noise} = 2(\text{rand}-0.5)$, $dn = s + v$, $x$ delayed by 2 from $dn$, $c = \text{ones}(1,32)$, $a = 0.8$, $\mu = 0.005$, $M = 32$.

**Note:** Solving the constrained optimization problem using the Lagrange multiplier method, the NLMS can be obtained (see Problem 9.12.2).

### 9.14 THE LEAST MEAN FOURTH ALGORITHM

The optimal weight vector $w^o$ that solves $\min E[|d(n) - x^T(n)w(n)|^4]$ (see Sayed 2003) can be approximated iteratively by the recursion:

$$w(n+1) = w(n) + \mu x(n)e(n)e(n)^2$$  \hspace{1cm} (9.50)

$$w(n+1) = w(n) + \mu x^*(n)e(n)|e(n)|^2$$ (complex case)

For the application of the least mean fourth (LMF) algorithm, see Problem 9.13.1.
Variants of Least Mean-Square Algorithm

9.15 THE LEAST MEAN MIXED NORM LMS ALGORITHM

The optimal weight vector \( \mathbf{w}^* \) that solves \( \min \{ \delta |e|^2 + 1/2(1 - \delta)|e|^4 \} \), \( e = d - \mathbf{x}^T \mathbf{w} \) for some constant \( 0 \leq \delta \leq 1 \), can be approximated iteratively using the recursion (see Sayed 2003):

\[
\begin{align*}
\mathbf{w}(n+1) &= \mathbf{w}(n) + \frac{\alpha - \mathbf{c}^T \mathbf{w}(n)}{\mathbf{c}^T \mathbf{c}} \mathbf{c} \\
\end{align*}
\]

### TABLE 9.4

The Linearly Constrained LMS Algorithm

<table>
<thead>
<tr>
<th>Input</th>
<th>Initial coefficient vector, ( \mathbf{w}(0) = \mathbf{0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input data, ( \mathbf{x}(n) )</td>
<td>Desired output, ( d(n) )</td>
</tr>
<tr>
<td>Desired output, ( d(n) )</td>
<td>Constant vector, ( \mathbf{c} )</td>
</tr>
<tr>
<td>Constant vector, ( \mathbf{c} )</td>
<td>Constraint constant, ( a )</td>
</tr>
<tr>
<td>Constraint constant, ( a )</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>Filter output, ( y(n) )</td>
</tr>
<tr>
<td>Procedure</td>
<td>( y(n) = \mathbf{w}^T(n) \mathbf{x}(n) )</td>
</tr>
<tr>
<td></td>
<td>( e(n) = d(n) - y(n) )</td>
</tr>
<tr>
<td></td>
<td>( \mathbf{w}'(n) = \mathbf{w}(n) + \mu e(n) \mathbf{x}(n) )</td>
</tr>
<tr>
<td></td>
<td>( \mathbf{w}(n+1) = \mathbf{w}'(n) + \frac{\alpha - \mathbf{c}^T \mathbf{w}(n)}{\mathbf{c}^T \mathbf{c}} \mathbf{c} )</td>
</tr>
</tbody>
</table>

### FIGURE 9.16

Delay by 2, \( x \)

\( y \)

\( J \)
\[ w(n+1) = w(n) + \mu x(n)e(n)[\delta + (1-\delta)|e(n)|^2] \]  \hspace{1cm} (9.51)

For the application of the least mean mixed norm (LMMN) algorithm and its comparison with the normalized one, see Problem 9.14.1.

### 9.16 SHORT-LENGTH SIGNAL OF THE LMS ALGORITHM

The proposed Book program that alleviates the problem of using a short-length signal present is given below. To test the proposed approach, we proceed to identify an unknown system as shown in Figure 9.17.

**Book Program for Short-Time Signals**

```matlab
>>M=8;mu=0.03;
>>v=rand(1,200)-0.5;%length (v)=200;
>>x=conv(v,[0.90.2-0.2]);
>>d=conv(x,[42-0.40.2-0.1]);
>>x=x(1,1:200);
>>d=d(1,1:200);
>>ve=[vvvvvvvv];%extended signal;
>>xe=conv(ve,[0.90.2-0.2]);
>>dn=conv(xe,[42-0.40.2-0.1]);
>>xe=xe(1,1:1600);
>>de=dn(1,1:1600);
>>[w,y,e,J,w1]=lms_short_signal(x,d,mu,M);
>>[we,ye,ee,Je,w1e]=lms_short_signal(xe,de,mu,M);
```

**Book m-Function for Short-Time Signals**

```matlab
function[w,y,e,J,w1]=lms_short_signal(x,dn,mu,M)
N=length(x);w=0.02*(rand(1,M)-0.5);
for n=M:N
    x1=x(n:-1:n-M+1);
    y(n)=w*x1';
    e(n)=dn(n)-y(n);
    w=w+mu*e(n)*x1;
```

---

**FIGURE 9.17**
The implementation of the LMS filter in the frequency domain can be accomplished simply by taking the discrete Fourier transform (DFT) of both the input data, \( \{x(n)\} \), and the desired signal, \( \{d(n)\} \). The advantage of doing this is due to the fast processing of the signal using the fast Fourier transform (FFT) algorithm. However, this procedure requires a block-processing strategy, which results in storing a number of incoming data in buffers, and thus, some delay is unavoidable.

![Figure 9.18](image-url)
Adaptive Filtering

The block diagram approach for the transform domain LMS (TDLMS) algorithm is shown in Figure 9.19. The signals are processed by a block-by-block format, that is, \( \{ x(n) \} \) and \( \{ d(n) \} \) are sequenced into blocks of length \( M \) so that

\[
x_i(n) = x(iM + n), \quad d_i(n) = d(iM + n) \quad n = 0, 1, \ldots, M - 1, \quad i = 0, 1, 2, \ldots
\]

The values of the \( i \)th block of the signals \( \{ x_i(n) \} \) and \( \{ d_i(n) \} \) are Fourier transforms using the DFT to find \( X_i(k) \) and \( D_i(k) \), respectively. Because of the DFT properties, the sequences \( X_i(k) \) and \( D_i(k) \) have \( M \) complex elements corresponding to frequency indices (“bins”) \( k = 0, 1, 2, \ldots, M - 1 \). Hence,

\[
X_i(k) = \text{DFT} \{ x_i(n) \} = \sum_{n=0}^{M-1} x_i(n) e^{-j \frac{2\pi nk}{M}} \quad k = 0, 1, \ldots, M - 1
\]

\[
D_i(k) = \text{DFT} \{ d_i(n) \} = \sum_{n=0}^{M-1} d_i(n) e^{-j \frac{2\pi nk}{M}} \quad k = 0, 1, \ldots, M - 1
\]

During the \( i \)th block processing, the output of each frequency bin of the adaptive filter is computed as follows:

\[
Y_i(k) = W_{i,k} X_i(k) \quad k = 0, 1, 2, \ldots, M - 1
\]

where:

\( W_{i,k} \) is the \( k \)th frequency bin corresponding to the \( i \)th update (corresponding to the \( i \)th block data)

The error in the frequency domain is

\[
E_i(k) = D_i(k) - Y_i(k) \quad k = 0, 1, 2, \ldots, M - 1
\]
Variants of Least Mean-Square Algorithm

The system output is given by

$$y_i(n) = y(iM + n) = IDFT \{Y_i(k)\} = \frac{1}{M} \sum_{k=0}^{M-1} Y_i(k)e^{-\frac{2\pi nk}{M}} \quad n = 0, 1, \ldots, M - 1$$

(9.57)

To update the filter coefficients we use, by analogy of the LMS recursion, the following recursion is used:

$$W_{i+1} = W_i + \mu E_i \cdot X_i^*$$

(9.58)

where:

$$W_i = [W_{i,0} \quad W_{i,1} \quad \cdots \quad W_{i,M-1}]^T$$

$$E_i = [E_i(0) \quad E_i(1) \quad \cdots \quad E_i(M - 1)]^T$$

$$X_i^* = [X_i^*(0) \quad X_i^*(1) \quad \cdots \quad X_i^*(M - 1)]^T$$

The dot (•) in (9.58) implies element-by-element multiplication and the asterisk (*) stands for complex conjugate. If we set $X_i^*$ in the form:

$$X_i = \text{diag} \{X_i(0) \quad X_i(1) \quad \cdots \quad X_i(M-1)\}$$

$$= \begin{bmatrix}
X_i(0) & 0 & \cdots & 0 \\
0 & X_i(1) & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & X_i(M-1)
\end{bmatrix}$$

(9.59)

then (9.58) becomes

$$W_{i+1} = W_i + \mu X_i^* E_i$$

(9.60)

Therefore, Equations 9.52 through 9.58 constitute the frequency domain of the LMS algorithm. The Book m-function gives the adaptive filter coefficients after $I$ blocks (or iterations) is given below.

**Book m-Function for the Frequency Domain of the LMS**

```matlab
function [A] = lms_FT_lms(x, d, M, I, mu)
    %function [A] = lms_FT_lms(x, d, M, I, mu);
    wk = zeros(1, M); %M=number of filter coefficients;
    for i = 0: I
        if I*M>length(x)-1
            ('error:I*M<length(x)-1')
        end;
        x1 = x(M*(i+1):-1:i*M+1);
        d1 = d(M*(i+1):-1:i*M+1);
        xk = fft(x1);
    end
end
```

```matlab
xl=x(M*(i+1):-1:i*M+1);
dl=d(M*(i+1):-1:i*M+1);
xk=fft(xl);
```
Adaptive Filtering

dk=fft(d1);
yk=wk.*xk;
ek=dk-yk;
wk=wk+mu*ek.*conj(xk);
A(i+1,:)=wk;
end;

%all the rows of A are the wk's at an increase order
%of iterations(blocks); to filter the data, wk must be
%inverted in the time domain, convolve with the data x
%and then plot the real part of the output y, e.g.,
%wn4=the forth iteration =ifft(A(4,:)),
%yn4=filter(wn4/4,1,x)for even M;

Example 9.16.1

The following Book m-file (or script file) produces the output of Figure 9.20. To produce the figure in the Command window, we just write: ex9_16_1.

Book m-File (Script File)

%ex9_16_1
M=40;I=12;mu=0.01;
n=0:999;

FIGURE 9.20
\[
d=d=\sin(0.1\pi n); v=1.5\times\text{rand}(1,1000);
\]
\[
x=d+v;
\]
\[
[A]=\text{lms\_FT\_lms}(x,d,M,I,\mu);
\]
\[
\text{wn10=ifft}(A(10,:))/M; % \text{inverse FT of row 10 of matrix A;}
\]
\[
\text{yn10=filter}(\text{wn10}/I,1,x)/M; % \text{inverse FT of row 10 of matrix A;}
\]
\[
\text{subplot}(2,1,1); \text{plot}(x(1,1:200),'k'); \quad xlabel('n'); ylabel('x(n)');
\]
\[
\text{subplot}(2,1,2); \text{plot}(\text{real}(\text{yn10}(1,1:200)),'k'); \quad xlabel('n'); ylabel('y(n)');
\]

### 9.17.1 Convergence

Let the signals \(\{x(n)\}\) and \(\{y(n)\}\) be jointly WSS and the initial filter coefficient be zero, \(W_0 = 0\). Based on (9.60) and (9.56) (see Problem 9.16.1), we obtain

\[
W_{i+1,k} = \left(1 - \mu \|X_i\|^2\right)W_{i,k} + \mu D_i(k)X_i^*(k)
\]  
(9.61)

The expected value of (9.61), assuming that \(W_{i,k}\) and \(X_i(k)\) are statistically independent, is given by

\[
E\{W_{i+1,k}\} = \left(1 - \mu E\left\{\|X_i(k)\|^2\right\}\right)E\{W_{i,k}\} + \mu E\{D_i(k)X_i^*(k)\}
\]  
(9.62)

Because \(x(n)\) and \(x(n)\) are stationary, their statistical characteristics do not change from block to block, and therefore, the ensembles \(E\{X_i(k)\|^2\}\) and \(E\{D_i(k)X_i^*(k)\}\) are independent of \(i\) but depend on \(k\). Taking the z-transform of (9.62) with respect to \(i\) of the dependent variable \(W_{i,k}\), we find the relation (see Table 3.3.1 and Problem 9.16.2):

\[
W_k(z) = -\mu E\left\{\|X_i(k)\|^2\right\} \frac{W_k(z)}{z-1} + \frac{\mu E\{D_i(k)X_i^*(k)\}}{z-1}
\]  
(9.63)

Applying the final value theorem (see Table 3.2 and Problem 9.16.2), we obtain the steady-state value for the filter coefficients:

\[
E\{W_k^\infty\} = \frac{E\{D_i(k)X_i^*(k)\}}{E\{\|X_i(k)\|^2\}}
\]  
(9.64)

Let the mean filter coefficient error \(E_i(k)\) be defined by

\[
E_i(k) = E\{W_{i,k}\} - E\{W_k^\infty\}
\]  
(9.65)

Then, using (9.64) and (9.62), we find (see Problem 9.16.3)

\[
E_{i+1} = \left(1 - \mu E\left\{\|X_i\|^2\right\}\right)E_i(k) \quad k = 0, 1, 2, 3, \ldots, M - 1
\]  
(9.66)
9.18 THE ERROR NORMALIZED STEP-SIZE LMS ALGORITHM

The variable error normalized step size (ENSS) depends on the optimization problem as follows:

Define \textbf{a posteriori} error \( e_p(n) \) as

\[ e_p(n) = d(n) - \mathbf{w}^T(n+1)\mathbf{x}(n) \]  

(9.67)

Then, we can define the following constraint optimization problem (Sayed 2003):

\[
\min_{\mathbf{w}} \left\| \mathbf{w}(n) - \mathbf{w}(n-1) \right\|^2 \quad \text{subject to} \quad e_p(n) = \left[ 1 - \mu \frac{\mathbf{x}(n)^2}{1 + \mu \left\| e_L(n) \right\|^2} \right] e(n)
\]

(9.68)

where:

\[
\left\| e(n) \right\|^2 = \sum_{i=0}^{L-1} \left| e(n-i) \right|^2
\]

(9.69)

By defining the difference as

\[
\Delta \mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n)
\]

(9.70)

and multiplying both sides of (9.70) by \( \mathbf{x}^T(n) \) from left, we obtain

\[
\mathbf{x}^T(n)\Delta \mathbf{w}(n) = \mathbf{x}^T(n)\mathbf{w}(n+1) - \mathbf{x}^T(n)\mathbf{w}(n)
\]

(9.71)

By adding and subtracting \( d(n) \) from the right-hand side of (9.71), we have

\[
\mathbf{x}^T(n)\Delta \mathbf{w}(n) = e(n) - e_p(n)
\]

(9.72)

Substituting (9.68) in (9.72), we obtain

\[
\mathbf{x}^T(n)\Delta \mathbf{w}(n) = \mu e(n) \frac{\mathbf{x}(n)}{1 + \mu \left\| e_L(n) \right\|^2}
\]

(9.73)

There are infinite many solutions to (9.73). The solution that results in a minimum value of \( \Delta \mathbf{w}(n) \) in the Euclidean second norm sense is the one taken directly from (9.73) (Sayed 2003), that is,

\[
\Delta \mathbf{w}(n) = \mu e(n) \frac{\mathbf{x}(n)}{1 + \mu \left\| e_L(n) \right\|^2}
\]

(9.74)

Equation 9.74 can be proved by assuming that \([\Delta \mathbf{w}(n) + \mathbf{z}]\) is any other solution, where \( \mathbf{z} \) is a column vector. The Euclidian vector norm of this solution can be written as
\[
\|\Delta w(n) + z\|^2 = (\Delta w(n) + z)^T (\Delta w(n) + z) \\
= \|\Delta w(n)\|^2 + \|z\|^2 + \Delta w^T(n)z + z^T \Delta w(n)
\]  
(9.75)

Substituting (9.74) in (9.75) yields

\[
\|\Delta w(n) + z\|^2 = \|\Delta w(n)\|^2 + \|z\|^2 + \mu e(n) \frac{x^T(n)z}{1 + \mu \|e_L(n)\|^2} + \mu e(n) \frac{z^T x(n)}{1 + \mu \|e_L(n)\|^2}
\]
(9.76)

Since \(\Delta w(n)\) is a solution, we conclude from (9.74) that \(x^T(n)z = 0 = z^T x(n)\), and hence, (9.76) becomes

\[
\|\Delta w(n) + z\|^2 = \|\Delta w(n)\|^2 + \|z\|^2
\]
(9.77)

The right-hand side of (9.77) is larger than \(\|\Delta w(n)\|^2\) for any nontrivial vector \(z\). Thus, the minimum value of \(\|\Delta w(n) + z\|^2\) is \(\|\Delta w(n)\|^2\) (i.e., when \(z = 0\)), which proves (9.74).

Now, substituting (9.74) in (9.70), we obtain the ENSS LMS algorithm:

\[
w(n+1) = w(n) + \frac{\mu x(n)}{1 + \mu \|e_L(n)\|^2} x(n)e(n)
\]
(9.78)

The fractional quantity in (9.78) represents the variable step size of the algorithm, that is,

\[
\mu(n) = \frac{\mu}{1 + \mu \|e_L(n)\|^2} \text{ or } \mu(n) = \frac{1}{1 + \frac{1}{\mu} \|e_L(n)\|^2}
\]
(9.79)

The above equation shows that \(\mu(n)\) is an increasing function of the step size \(\mu\). In a stationary environment, the best choice of \(L = n\), which in turn makes \(\mu(n)\) a monotonic decreasing function of \(\mu\). Clearly, large values of \(\mu\) should be used in this case to increase \(\mu(n)\), thus obtaining the fast rate of convergence mainly at the early stages of adaptation, where the error value is large. In stationary environments, the length of the error vector should be constant \((L = \text{constant value})\) such that the algorithm will adapt to statistical input data. In this case, as \(n\) increases, \(\|e_L(n)\|^2\) decreases and \(\mu(n)\) increases to a maximum value \(\mu\), which is the step size of the LMS algorithm.

The proposed Book algorithm performs better than many other algorithms that have been examined in stationary environments for both small and large filter lengths.

**Example 9.17.1**

Compare the ENSS LMS algorithm with the LMS and NLMS algorithms. To check the algorithms, we used the identification of an unknown system as shown in Figure 9.21. The results are shown in Figure 9.22.
Solution: The following Book m-functions and program were used.

**Book m-Functions for ENSS Application**

```matlab
function [J,w]=lms_error_normalized_SS(h,avn,N,mu1,M)
    J=zeros(1,N);
    for i=1:avn
        x=filter([1 0.5],1,randn(1,N)); y=zeros(1,N);
        w=zeros(1,M); e=zeros(1,N); X=zeros(1,M);
        d=filter(h,1,x); v=0.3*randn(1,N);
        D=0;
        %x=input to the unknown filter h;
        %y=output from the adaptive filter;
        %avn=number of averaging;d=output from
        %the unknown filter plus system noise;
```
%M=number of adaptive filter coefficients;
for k=1:N
    X=[x(k) X(1:M-1)];
    den=X*X'+0.0001;
    y=w*X';
    e(k)=d(k)+v(k)-y;
    D=D+e(k)^2;
    den1=D*mu1+1;
    w=w+(mu1/den1)*e(k)*X;
    J(k)=J(k)+abs(e(k))^2;
end
J=J/avn;

function [J,w]=lms_average_normalized_lms(h,avn,N,mu1,M)
J=zeros(1,N);
for i=1:avn
    x=filter([1 0.5],1,randn(1,N));y=zeros(1,N);
    w=zeros(1,M);e=zeros(1,N);X=zeros(1,M);
    d=filter(h,1,x);v=0.3*randn(1,N);
    %J=learning curve(MSE);h=unknownfilter;
    %w=adaptive filter coefficients;v=internal
    %system noise;avn=number of times been
    %averaged;M=number of adaptive filter
    %coefficients;N=number of data;
    for k=1:N
        X=[x(k) X(1:M-1)];
        den=X*X'+0.0001;
        y=w*X';
        e(k)=d(k)+v(k)-y;
        w=w+(mu1/den1)*e(k)*X;
        J(k)=J(k)+abs(e(k))^2;
    end
    J=J/avn;
end

function [J,w]=lms_average_unnormalized_lms(h,avn,N,mu1,M)
J=zeros(1,N);
%for explanation see the above two functions;
for i=1:avn
    x=filter([1 0.5],1,randn(1,N));y=zeros(1,N);
    w=zeros(1,M);e=zeros(1,N);X=zeros(1,M);
    d=filter(h,1,x);v=0.3*randn(1,N);
    for k=1:N
        X=[x(k) X(1:M-1)];
        y=w*X';
        e(k)=d(k)+v(k)-y;
        w=w+mu1*e(k)*X;
        J(k)=J(k)+abs(e(k))^2;
    end
    J=J/avn;
end
Book MATLAB Program

```matlab
>>[J, w]=lms_error_normalized_SS([210.5-0.2],200,1000, ... 0.8, 6);
>>[J1, w1]=lms_average_normalized_lms([210.5-0.2], 200, ... 1000, 0.01,6);
>>[J2, w2]=lms_average_unnormalized_lms([210.5-0.2], 200, ... 1000, 0.01,6);
>>plot(10*log10(J(1,1:550)),'k'); hold on; >>plot(10*log10(J1(1,1:550)),'k');
>>hold on; plot(10*log10(J2(1,1:550)),'k');xlabel('No. of iterations');
>>ylabel('J (MSE) in dB');
```

9.19 THE ROBUST VARIABLE STEP-SIZE LMS ALGORITHM

Based on regularization Newton’s recursion (Sayed 2003), we write

\[ w(n + 1) = w(n) + \mu(n) [\varepsilon(n)I + R_x]^{-1} [p - R_xw(n)] \]  \hspace{1cm} (9.80)

where:

- \( n \) is the iteration number
- \( w \) is an \( M \times 1 \) vector of adaptive filter weights
- \( \varepsilon(n) \) is an iteration-dependent regularization parameter
- \( \mu(n) \) is an iteration-dependent step size
- \( I \) is an \( M \times M \) identity matrix
- \( p(n) = E[d(n)x(n)] \) is the cross-correlation vector between the desired signal \( d(n) \) and the input signal \( \{x(n)\} \)
- \( R_x(n) = E\{x(n)x^T(n)\} \) is the autocorrelation matrix of \( x(n) \)

Writing (9.80) in the LMS form by replacing \( p \) and \( R_x \) by their instantaneous approximations \( \{d(n)x(n)\} \) and \( \{x(n)x^T(n)\} \), respectively, with the appropriate proposed weights, we obtain

\[ w(n + 1) = w(n) + \mu \|e_L(n)\|^2 \times [\alpha \|e_L(n)\|^2 I + \gamma x(n)x^T(n)]^{-1} x(n)e(n) \] \hspace{1cm} (9.81)

where:

- \( \mu \) is a positive constant step size
- \( \alpha \) and \( \gamma \) are positive constants
- \( e(n) \) as the system output error is defined by

\[ e(n) = d(n) - x^T(n)w(n) \] \hspace{1cm} (9.82)

\[ \|e(n)\|^2 = \sum_{i=0}^{n-1} |e(n - i)|^2 \] \hspace{1cm} (9.83)
Equation 9.83 is the square norm of the error vector, $e(n)$, estimated over its entire updated length $n$, and (9.84) is the square norm of the error vector, $e(n)$, estimated over the last $L$ values.

Expanding (9.81) and applying the matrix inversion formula (see Appendix 2),

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B[C^{-1} + DA^{-1}B]^{-1}DA^{-1}$$  

(9.85)

with $A = \alpha \|e(n)\|^2 I$, $B = x(n)$, $C = \gamma$, and $D = x^T(n)$, we obtain

$$\left[ \alpha \|e(n)\|^2 I + \gamma x(n)x^T(n) \right]^{-1} = \alpha^{-1}\|e(n)\|^2 I$$

$$-\alpha^{-1}\|e(n)\|^2 Ix(n) \frac{x^T(n)\alpha^{-1}\|e(n)\|^2}{\gamma^{-1} + x^T(n)\alpha^{-1}\|e(n)\|^2 x(n)}$$

(9.86)

Multiplying both sides of (9.86) by $x(n)$ from the right and rearranging the equation, we have (see Problem 9.18.1)

$$\left[ \alpha \|e(n)\|^2 I + \gamma x(n)x^T(n) \right]^{-1} x(n) = \frac{x(n)}{\alpha\|e(n)\|^2 + \gamma\|x(n)\|^2}$$

(9.87)

Substituting (9.87) in (9.81), we obtain a new proposed robust variable step size (RVSS):

$$w(n+1) = w(n) + \frac{\mu\|e_L(n)\|^2}{\alpha\|e(n)\|^2 + (1 - \alpha)\|x(n)\|^2} x(n)e(n)$$

(9.88)

where we replaced $\gamma$ by $(1 - \alpha) \geq 0$ in (9.88) without loss of generality. The fractional quantity in (9.88) may be viewed as a time-varying step size, $\mu(n)$, of the proposed RVSS algorithm. Clearly, $\mu(n)$ is controlled by normalized of both error and input data.

The parameters $\alpha$, $L$, and $\mu$ are appropriately chosen to achieve the best trade-off between the rate of convergence and low final mean-square error. A small constant $\varepsilon$ could be added to the denominator of (9.88) to prevent instability of the algorithm if the denominator becomes too small. The quantity $\|e_L(n)\|^2$ is large at the beginning of adaptation, and it decreases as $n$ increases, while $\|x(n)\|^2$ fluctuates depending on the recent values of the input signal. However, $\|e(n)\|^2$ is an increasing function of $n$ since $e(n)$ is a vector of increasing length. To compute (9.83) with minimum computational complexity, the error value produced in the first iteration is squared and stored. The error value in the second iteration is squared and added to the previous stored value. Then, the result is stored in order to be used in the next iteration and so on.
Example 9.18.1

Use the unknown system shown in Figure 9.21, study the RVSS algorithm with respect to normalized and ENSS algorithms. Figure 9.23 shows the desired results.

Solution: The following Book m-function and program were used.

**Book m-Function for RVSS Algorithm**

```matlab
function [J,w]=lms_robust_variable_SS(avn,N,mu1,h,M)
    J=zeros(1,N);
    %M>length(h);J=MSE-learning curve;avn=number of averaging
    %the learning curve; N=number of inputs;M=number
    %of adaptive filter coefficients; h=unknown system;
    %mu1=constant step-size;v=internal system noise;
    for i=1:avn
        y=zeros(1,N);w=zeros(1,M);e=zeros(1,N);
        X=zeros(1,M);D=zeros(1,M);
        x=filter([1 0.5],1,0.3*randn(1,N));v=0.1*randn(1,N);
        for k=1:N
            d=filter(h,1,x)+v;
            X=[x(k) X(1,1:M-1)];
            den=X*X'+0.0001;
            y=w*X';
            e(k)=d(k)-y;
            D=[e(k) D(1,1:M-1)];
            denx=D*D';
        end
        J=J+10*log10(denx);
    end
end
```

**FIGURE 9.23**

![Graph showing the comparison between different algorithms](image_url)
Variants of Least Mean-Square Algorithm

\[ a = 0.5; \]
\[ \mu = \frac{(\mu_1 \times \text{den})}{((a \times \text{den}) + (1-a) \times e \times e')} \]
\[ w = w + \mu \times e(k) \times X; \]
\[ J(k) = J(k) + \text{abs}(e(k))^2; \]
\[ \text{end; end; J=} \]
\[ \text{J/avn;} \]

**Book MATLAB Program**

```matlab
>> [J, w] = lms_robust_variable_SS(300, 2000, 0.07, [2 1 0.5 -0.2], 6);  
>> [J1, w1] = lms_error_normalized_SS([2 1 0.5 -0.2], 300, 2000, 0.15, 6);  
>> [J2, w2] = lms_average_normalized_lms([2 1 0.5 -0.2], 300, 2000, 0.02, 6);  
>> plot(10*log10(J(1,1:1800)),'k');  
>> hold on; plot(10*log10(J1(1,1:1800)),'k');  
>> hold on; plot(10*log10(J2(1,1:1800)),'k'); xlabel('No. of ... iterations');  
>> ylabel(J, 'MSE in dB');
```

The reader will observe that RVSSs adjust faster than the other algorithms.

**Example 9.18.2**

Use NLMS to study the results for an abrupt change in the unknown system, from \( h \) becomes \(-h\), which must be identified. The unknown system is shown in Figure 9.21.

**Solution:** The following Book m-functions and program were used to produce Figures 9.24 and 9.25. The value of \( w \) that was found with the constants used was 1.9988 0.9973 0.5573 -0.2052 0.0055 0.0519.

The code is as follows:

```matlab
function [J, mukav] = lms_abrupt_change_normalized_ ...  
    %N must be even; muk=gives the variation of step-size;  
    %N=number of data; M=length(h); avn=number of averaging;  
    %v=internal system noise; mu1=step-size constant; J=MSE;  
    J = zeros(1,N); muk = zeros(1,N);  
    for i = 1:avn  
        h = [2 1 0.5 -0.2];  
        y = zeros(1,N); w = zeros(1,M); e = zeros(1,N);  
        X = zeros(1,M); D = zeros(1,M);  
        x = filter([1 0.5],1,0.3*randn(1,N));  
        v = 0.1*randn(1,N);  
        for k = 1:N  
            d = filter(h,1,x)+v;  
            X = [x(k) X(1:M-1)];  
            den = X*X' + 0.0001;  
            y = w*X';
```

---

279
FIGURE 9.24

FIGURE 9.25
Variants of Least Mean-Square Algorithm

\[
e(k) = d(k) - y; \\
\text{if } k == N/2; \% N must be even; \\
\quad h = -h; \\
\text{end}; \\
w = w + (mu1/den) \times e(k) \times X; \\
muk(k) = muk(k) + (mu1/den); \\
J(k) = J(k) + |e(k)|^2; \\
\text{end}; \\
mukav = muk/avn; \\
J = J/avn;
\]

function \[w1, w\] = lms_abrupt_change_normalized_lms_w(N, M, mu1)
\% N must be even; muk=gives the variation of step-size; 
\% N=number of data; M>=length(h); avn=number of averaging; 
\% v=internal system noise; mu1=step-size constant; J=MSE; 
h = [2 1 0.5 -0.2]; 
y = zeros(1, N); w = zeros(1, M); e = zeros(1, N); 
X = zeros(1, M); D = zeros(1, M); 
v = 0.1 * randn(1, N); 
for k=1:N 
\% d=filter(h,1,x)+v; 
X = [x(k) \times X(1:M-1)]; 
\% den=X*X'+0.0001; 
y = w \times X'; 
e(k) = d(k) - y; 
\text{if } k == N/2; \% N must be even; 
\quad h = -h; 
\text{end}; 
w = w + (mu1/den) \times e(k) \times X; 
w1(k,:)=w(1,:); 
end;

Book MATLAB Program

>>[J,mukav]=lms_abrupt_change_normalized_lms(200,3000,6,0.1); 
>>[w1,w]=lms_abrupt_change_normalized_lms_w(3000,6,0.1);

For Figure 9.24, we used the following commands:

>> subplot(2,1,1); plot(10*log10(J), 'k'); 
>> xlabel('No. of iterations'); 
>> ylabel('J, MSE in dB'); subplot(2,1,2); plot(mukav, 'k'); 
>> axis([0 3000 0 1]); 
>> ylabel('Step-size'); xlabel('No. of iterations');

For Figure 9.25, we used the following commands:

>> plot(w1(:,1), 'k'); hold on; plot(w1(:,2), 'k'); 
>> hold on; plot(w1(:,5), 'k'); 
>> xlabel('No. of iterations'); ylabel('Different hs');
From Figure 9.26, the desire signal is the output of the linear regression model system, and it is given by (Haykin 1996)

\[ d(n) = w_{un}^T x(n) + v(n) \]  

(9.89)

where:
- \( w_{un} \) is the coefficient vector of the unknown system
- \( x(n) \) is the input vector (regressor)
- \( v(n) \) is a nonmeasurable white noise with variance \( \sigma_v^2 \)

The LMS FIR filter is chosen to minimize the following index of performance:

\[ J(w,K) = E \left\{ e^{2K(n)} \right\} \quad K = 1, 2, 3 \ldots \]  

(9.90)

From the above equation and using the instantaneous gradient vector, we find the modified LMS filter (Haykin 1996):

\[ w(n+1) = w(n) + \mu K x(n) e^{2K-1}(n) \]  

(9.91)

The following Book m-function simulates the above equation:

**Book m-Function for Modified LMS Filter**

```matlab
function [w, e, Jav] = lms_modified_lms(mu, av, a, b, N, M, K)
    for m = 1:av
        %averaging; a=multiplier of the random noise,
        %a*randn(1,N);
        w = zeros(1, M);
```

FIGURE 9.26
Variants of Least Mean-Square Algorithm

\[
x = a*(\text{rand}(1,N)-0.5);
\]
\[
d = \text{filter}([0.9 \ 0.4-0.2],1,x)+b*(\text{rand}(1,N)-0.5);
\]
%multiplier of multilinear regressor model;
for n=M:N
\[
x1 = x(n:-1:n-M+1);
\]
\[
y(n) = w*x1';
\]
\[
e(n,m) = d(n) - y(n); %e(n,m) becomes a Nxav
\]
%matrix;
\[
w = w + \mu*K*e(n,m)^{(2*K-1)}*x1;
\]
end
\[
y = \text{zeros}(1,N);
\]
end
\[
J_{av} = \text{sum}(e.^{2},2)/av; %e.^2 squares each matrix element;
\]

Figure 9.27 shows the results with the following inputs: \( \mu = 0.04, av = 300 \) (ensemble number), \( a = 1.5, b = 0.2, N = 15,000, M = 6 \). It is observed that the modified LMS filter is not as efficient as the standard LMS filter.

\[\text{◼}\]

9.21 MOMENTUM LMS

Another LMS type filter has been proposed, known as momentum LMS, which is given by

\[
w(n+1) = w(n) + (1-g)(w(n) - w(n-1)) + ag\mu e(n)x(n)
\]

(9.92)
where:

\[ 0 < g < 1 \]

\[ a > 1 \]

The following Book m-file was used to produce Figure 9.28.

**Book m-Function for Momentum LMS**

```matlab
function [w, y, e, J, w1] = lms_momentum_lms(x, d, mu, g, a, M)
    %0<g<1,
    %function [w, y, e, J, w1] = lms_momentum_lms(x, d, mu, M);
    %all quantities are real valued;
    %x=input data to the filter; d=desired signal;
    %M=order of the filter;
    %mu=step-size factor; x and d must be
    %of the same length, in the program they are
    %reversed;
    %w1=a matrix of dimensions: length(x)xM,
    %each column represents the variation of
    %each filter coefficient;
    N = length(x); w(2, 1:M) = zeros(1, M); w1 = zeros(1, M);
    w(1, 1:M) = zeros(1, M);
    xr = fliplr(x); dr = fliplr(d);
```
Variants of Least Mean-Square Algorithm

for \( n=1: N-M \)
\[
x_1 = x_r(n:1:M+n-1); \quad \% \text{for each } n \text{ the vector } x_1 \text{ is of length } M;
\]
\[
y(n) = w(n+1,1:M) \cdot x_1';
\]
\[
e(n) = d_r(n) - y(n);
\]
\[
w(n+2,:) = w(n+1,:) + (1-g) \cdot (w(n+1,:) - w(n,:)) + g \cdot a \cdot mu \cdot e(n) \cdot x_1;
\]
\[
w1(n,:) = w(1,:);
\end
\]
\[
J = e.\cdot^2; \quad \% \text{J is the learning curve of the adaptation};
\]

Figure 9.28a is the result of the standard LMS filter with the constants: \( \mu = 0.06, M = 6. \) Figure 9.28b is the result of the momentum LMS with the following constants: \( \mu = 0.06, M = 6, g = 0.4, a = 4. \) For comparison, we used the system identification problem with an FIR system with coefficients \([2 \quad 4 \quad -1]. \) The coefficients after 1900 iterations were found to be \((1) 2.0020 \quad 3.9968 \quad -0.9960 \quad -0.0039 \quad 0.0031 \quad -0.0019 \) and \((2) 2.0000 \quad 4.0000 \quad -1.0000 \quad -0.0000 \quad 0.0000 \quad -0.0000. \)

### 9.22 THE BLOCK LMS ALGORITHM

We can subdivide a set of data in \( L \) sections and process the data sequentially. The following Book m-function processes the data in a block-by-block format.

**Book m-Function for Block LMS Processing**

```matlab
function [w,y,e,J,w1]=lms_block_lms(x,d,mu,M,L)
    \%all quantities are real valued;
    \%x=input data to the filter; d=desired signal;
    \%M=order of the filter;
    \%mu=step-size factor; x and d must be \%of the same length, in the program they are \%reversed;N/L must be an integer;
    \%w1=a matrix of dimensions: length(x)x(M+1), \%each column represents the variation of \%each filter coefficient;
    N=length(x);w(2,1:M+1)=zeros(1,M+1);w1=zeros(1,M+1);
    w(1,1:M+1)=zeros(1,M+1);
    xr=fliplr(x);dr=fliplr(d);
    for m=1:L-1
        if m*(N/L)+1>N
            end;
        for n=((m-1)*(N/L)+1):1:m*(N/L)
            x1=xr(n:1:M+n); \%for each n the vector x1 is of length M+1;
            y(n) = w(n+1,1:M+1) \cdot x1';
            e(n) = d_r(n) - y(n);
            w(n+2,:) = w(n+1,:) + (mu/L) \cdot e(n) \cdot x1;
            w1(n,:) = w(1,:);
        end;
    end;
    J = e.^2; \%J is the learning curve of the adaptation;
```

Using the identification of an FIR system with impulse response 
\[ h = [0.9 \ 0.4 \ -0.1] \], with a zero-mean white noise as input we obtain
the learning curve shown in Figure 9.29. The constants used were as follows: 
\[ N = 12000, \ L = 4, \ \mu = 0.08, \ d = \text{Filter}([0.9 \ 0.4 \ -0.1],1,x) \].
The values of the adaptive filter were as follows: at the 3000th iteration 
\[ w(3000,:) = \begin{bmatrix} 0.8945 & 0.3975 & -0.0991 & 0.0003 & -0.0001 & 0.0003 & 0.0002 \end{bmatrix} \]; at
the 6000th iteration \[ w(6000,:) = \begin{bmatrix} 0.9000 & 0.4000 & -0.1000 & -0.0000 & -0.0000 & -0.0000 & 0.0000 \end{bmatrix} \].

9.23 THE COMPLEX LMS ALGORITHM

The complex LMS algorithm is given in Table 9.5. Figure 9.30 shows the two learning
curves: one for unnormalized and the other for normalized complex LMS adapti-
ve filters. It shows that after the first about 100 iterations, the normalized filter
performs better. The input for both the unknown system and the adaptive filter was 
\[ x = \text{randn}(1,12000) + j0.2\text{filter}([0.9 \ 0.2],1,\text{rand}(1,12000) - 0.5) \]. The output from
the unknown filter was \[ d = \text{Filter}([0.9 \ 0.4 \ -0.1],1,x) \]. To obtain the unknown
filter, we write \[ \text{real}(w) = \begin{bmatrix} 0.9092 & 0.4052 & -0.1044 & -0.0053 & -0.0011 & -0.0069 \end{bmatrix} \].

**Book m-File for Normalized Complex LMS Filter**

```matlab
function [w,y,e,J,w1,Js]=lms_complex_normalized_lms(x,dn,mu,M)
    %x=input data to the filter; dn=desired signal;
    %M=order of the filter;
    %mu=step-size factor; x and dn must be
```
Variants of Least Mean-Square Algorithm

% of the same length;
% Js=smoothes the learning curve;
% w1=a matrix of dimensions: length(x)xM,
% each column represents the variation of
% each filter coefficient;
N=length(x); w=zeros(1,M); w1=zeros(1,M);
for n=M:N:

TABLE 9.5
The Complex LMS Algorithm

Unnormalized
\[ x(0) = w(0) = [0 \ 0 \ \ldots \ 0]^T \]
For \( n \geq 0 \)

\[ e(n) = d(k) - w^H(n)x(n) \]
\[ w(n + 1) = w(n) + \mu e^*(n)x(n) \]

Normalized
For \( n \geq 0 \)

\[ e(n) = d(k) - w^H(n)x(n) \]
\[ w(n + 1) = w(n) + \mu e^*(n)x(n) / [0.0001 + x(n)x^H(n)] \]

FIGURE 9.30
The algorithm is defined as follows (see Chapter 10):

First we create the signal matrix, which is of the form:

\[
X(n) = \begin{bmatrix}
    x(n) & x(n-1) & \cdots & x(n-N+1) & x(n-N) \\
    x(n-1) & x(n-2) & \cdots & x(n-N) & x(n-N-1) \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    x(n-M) & x(n-M-1) & \cdots & x(n-N-M+1) & x(n-N-M)
\end{bmatrix}
\]

(9.93)
The output vector from the adaptive filter is given by

$$y(n) = X^T(n)w(n) = \begin{bmatrix} y_0(n) \\ y_1(n) \\ \vdots \\ y_N(n) \end{bmatrix} \quad (9.94)$$

The desired and error signals are given by

$$d(n) = \begin{bmatrix} d(n) \\ d(n-1) \\ \vdots \\ d(n-N) \end{bmatrix}, \quad e(n) = \begin{bmatrix} e_0(n) \\ e_1(n) \\ \vdots \\ e_N(n) \end{bmatrix} = \begin{bmatrix} d(n) - y_0(n) \\ d(n-1) - y_1(n) \\ \vdots \\ d(n-N) - y_N(n) \end{bmatrix} = d(n) - y(n) \quad (9.95)$$

The affine algorithm is given in Table 9.6.

The $w$ has dimensions $M \times (N - M + 1)$. Therefore, we observe the evolution of $w$ toward its exact value. The following Book m-file was used to identify the unknown system: $h = [0.9 \ 0.4 \ -0.1]$. In this case, we used the following constants: $N = 200$, $M = 5$, $\mu = 0.05$, $g = 0.2$, $x = \text{filter}([0.9 \ 0.2], 1, \text{rand}(1,200)-0.5)$, $d = \text{filter}([0.9 \ 0.4 \ -0.1], 1, x)$.

**Book m-File for System Identification**

```matlab
function [w,y,e,J]=lms_affine_lms(x,d,mu,M,g)
    %all quantities are real valued; 0<g<1
    %x=input data to the filter; dn=desired signal;
    %M=order of the filter;
    %mu=step-size factor; x and dn must be
    %of the same length;
    %Js=smoothes the learning curve;
    %w1=a matrix of dimensions: length(x)xM,
    %each column represents the variation of
    %each filter coefficient;
```

---

**TABLE 9.6**

**The Affine Projection Algorithm**

<table>
<thead>
<tr>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x(0) = w(0) = [0 \ 0 \ \cdots \ 0]^T$</td>
</tr>
<tr>
<td>$\gamma =$ small constant</td>
</tr>
<tr>
<td>For $n \geq 0$</td>
</tr>
<tr>
<td>$e(n) = d(n) - X^T(n)w(n)$</td>
</tr>
<tr>
<td>$w(n+1) = w(n) + \mu X(n)X^T(n)x(n) + \gamma I^{-1}e(n)$</td>
</tr>
</tbody>
</table>
N=length(x); w=zeros(1,M)';
xr=fliplr(x)'; dr=fliplr(d)';
for k=1:M
    X(k,:)=xr(k:1:N-M+k); % M x (N-M+1)
end;
for n=1:N-M+1
    y=X'*w(:,n); % ((N-M+1)xM)x(Mx1)=(N-M+1)x1
    e=dr(1:(N-M+1))-y;
    w(:,n+1)=w(:,n)+mu*X*inv(X'*X+g*eye(N-M+1))*e;
end;
J=e.^2; % J is the learning curve of the adaptation;

The evolving of the first four out of five coefficients versus the iteration number is shown in Figure 9.31.

9.25 THE COMPLEX AFFINE LMS ALGORITHM

The complex affine LMS algorithm is presented in Table 9.7.

We have tested the algorithm given in the table by using the following constants and vectors to identify an unknown system with \( h = [0.9 \ 0.2 \ -0.1] \). The input vectors and constants were as follows:

\[
x = \text{filter}([0.9 \ 0.2], 1, 0.5 \ \text{randn}(1,200)) + j[\text{rand}(1,200)-0.5],
\]

\[
d = \text{filter}([0.9 \ 0.4 \ -0.1], 1, x)\mu = 0.05, \gamma = 0.2, M = 5
\]
TABLE 9.7
The Complex Affine Algorithm

\[ x(0) = w(0) = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}^T \]

\( \gamma = \text{small constant} \)

For \( n \geq 0 \)

\[ e^*(n) = d^*(n) - X^H(n)w(n) \]

\[ w(n + 1) = w(n) + \mu X(n)X^H(n)X(n) + \gamma I)^{-1} e^*(n) \]

Based on the above constants and vectors, we obtained the following approximations:

real\( (w(:,20)) = 20\text{th iteration} = [0.5590 \quad 0.2487 \quad -0.0621 \quad 0.0000 \quad -0.0000]^T \)

real\( (w(:,100)) = 100\text{th iteration} = [0.8943 \quad 0.3975 \quad -0.0994 \quad 0.0000 \quad -0.0000]^T \)

real\( (w(:,150)) = 150\text{th iteration} = [0.8996 \quad 0.3998 \quad -0.1000 \quad 0.0000 \quad -0.0000]^T \)

To obtain the above results, we used the following Book m-function:

**Book m-Function for Complex Affine Algorithm**

```matlab
function [w,y,e,J]=lms_complex_affine_lms(x,d,mu,M,g)
    % all quantities are real valued; 0 < g < 1
    % x=input data to the filter; d=desired signal;
    % M=order of the filter;
    % mu=step-size factor; x and d must be
    % of the same length;
    N=length(x);w=zeros(1,M)';
xr=fliplr(x)';dr=conj(fliplr(d)');
for k=1:M
    X(k,:)=xr(k:1:N-M+k);%Mx(N-M+1)
end;
for n=1:N-M+1
    y=X'*w(:,n);%((N-M+1)xM)x(Mx1)=(N-M+1)x1
    e=dr(1:(N-M+1))-y;
    w(:,n+1)=w(:,n)+mu*X*inv(X'*X+g*eye(N-M+1))*e;
end;
J=e.*conj(e);%J is the learning curve of the adaptation;
```

**PROBLEMS**

9.1.1 Verify (9.3) and (9.4).

9.1.2 Use NLMS to identify the system shown in Figure 8.5. Plot the filter coefficients for two different step-size values.
9.1.3 Repeat Example 9.1.2 but with an NLMS algorithm.

9.1.4 Plot the average learning function for identifying the system shown in Figure P9.1.4.1. Plot also the average learning curve for an NLMS. Compare with the nonaverage situation.

9.3.1 Find the output of the first and second stages of an SCNLMSF that is shown in Figure 9.8.

9.11.1 Develop the leaky LMS algorithm by minimizing the modified mean-square error:

\[ J(n) = e(n)^2 + \gamma w^T(n)w(n) \]

9.11.2 Compare the normalized and unnormalized leaky LMS algorithms.


9.12.2 We can derive the NLMS subject to a constraint: Minimize \( \min_w \|w(n + 1) - w(n)\|^2 = \min\{ [w(n + 1) - w(n)]^T [w(n + 1) - w(n)] \} \) subject to the constraint \( d(n) = w^T(n + 1)x(n) \).

9.13.1 Study the application of the LMF algorithm by making changes to different constants and input signals with varying noise strengths.

9.14.1 Study the application of the LMMN LMS algorithm by varying constants and input signals and with varying noise strengths. Compare the results between the standard and the normalized algorithm.

9.16.1 Verify (9.61).

9.16.2 Verify (9.63).
9.16.3 Verify (9.66).
9.18.1 Verify (9.87).
9.18.2 Repeat Examples 9.18.1 and 9.18.2 using uncorrelated input signals.
9.18.3 Repeat Examples 9.18.1 and 9.18.2 using the simple LMS algorithm.

**HINTS–SOLUTIONS–SUGGESTIONS**

9.1.1

\[ w(n+1) = w(n) + \mu(n)e(n)x(n) \]  
\[ e_{ps}(n) = d(n) - w^T(n+1)x(n) \]  

Substituting (2) in (1), we get

\[ e_{ps}(n) = d(n) - [w^T(n) + \mu(n)e(n)x^T(n)]x(n) \]
\[ = d(n) - w^T(n)x(n) - \mu(n)e(n)x^T(n)x(n) \]
\[ = e(n) - \mu(n)e(n)\|x(n)\|^2 \]

\[ \frac{\partial e_{ps}^2(n)}{\partial \mu(n)} = 2 \left[ e(n) - \mu(n)e(n)\|x(n)\|^2 \right] = 0 \Rightarrow \mu(n) = \frac{1}{\|x(n)\|^2} \]

9.1.2 From Figure P9.1.2, we observe that with \( \mu = 0.1 \) the adaptive coefficients approach the desired ones within 50–100 iterations. However, for \( \mu = 0.02 \), they need more than 400 iterations to approach the right value. Also observe that with the smaller step-size value, the curves are smoother.

The Book m-function used to obtain Figure P9.1.2 is as follows:

```matlab
function [w,w1,y,x,J]=lms_power_norm_lms_pr9_1_2(mu,M,b,c)
    %mu=step-size factor;
    %M=length of adaptive filter;c=very small number;
    %yc=conv(x,[filter coefficients])=channel
```

**FIGURE P9.1.2**
The unknown FIR system had the following coefficients: 1, \(-2\), 4. Also the white noise was passed through an FIR filter with coefficients: 0.9, \(-1\), 0.2.

The Book m-function for averaging LMS and NLMS denoising algorithms is given as follows:

```matlab
function [w, y, e, Jav] = lms_denoising_norm_lms_Jav(N, mu, M, av, a, ep)
for m = 1:av%av=integer equal to the number of desired
    %averaging; a=multiplier of the random noise,
    %a*randn(1,N); ep=a very small number, e.g., ep=0.0001;
    w = zeros(1, M);
    dn = sin(0.2*pi*[1:N]) + a*randn(1, N);
    x = [0 0 dn(1, 1:N-2)]; %delay 2 units;
    for n = M:N
        x1 = x(n:-1:n-M+1);
        y(n) = w*x1';
        e(n, m) = dn(n) - y(n); %e(n,m) becomes a Nxav
        end;
        w = w + mu*e(n, m)*x1/(ep + x1*x1');
    end;
    y = zeros(1, N);
    end;
Jav = sum((e.^2, 2)/av; % e.^2 squares each matrix element;
```

9.1.4 The constants and functions used were as follows: \(\mu = 0.01\), \(M = 15\), \(av = 200\), \(v = 0.5*\text{randn}(1,4000)\). The plots of \(J\)'s were logarithmical of the form \(10*\log(\text{Jav})\) versus iteration numbers. A typical result of the adaptive filter coefficients was as follows:

Columns 1 through 10

\[
\begin{align*}
0.9724 & \quad 1.0513 & \quad 0.9279 & \quad 1.0897 & \quad 0.8992 & \quad 0.8913 & \quad 1.1115 \\
-0.8920 & \quad -1.1000 & \quad -0.9129 
\end{align*}
\]
Variants of Least Mean-Square Algorithm

Columns 11 through 15

\[-0.0718 \quad 0.0588 \quad -0.0440 \quad 0.0308 \quad -0.0166\]

The two Book m-functions are given below.

**Book m-Function Averaging \(J\) with NLMS**

```matlab
function [w,y,e,Jav]=lms_normalized_system_ident_Jav(mu,M,av)
    for m=1:av
        w=zeros(1,M);
        v=0.5*randn(1,4000);
        v1=conv(v,[1 0.9]);
        dn=conv(v1,[1 1 1 1 -1 -1 -1 -1 -1]);
        dn=dn(1,1:4000);
        x=v1(1,1:4000);
        N=length(x);
        for n=M:N
            x1=x(n:-1:n-M+1);
            y(n)=w*x1';
            e(n,m)=dn(n)-y(n);
            w=w+mu*e(n,m)*x1/(0.0001+x1'*x1);
        end;
        y=zeros(1,N);
    end;
    Jav=sum((e.^2),2)/av;
end;
```

**Book m-Function Averaging \(J\) with Unnormalized LMS**

```matlab
function [w,y,e,Jav]=lms_system_ident_Jav(mu,M,av)
    for m=1:av
        w=zeros(1,M);
        v=0.5*randn(1,4000);
        v1=conv(v,[1 0.9]);
        dn=conv(v1,[1 1 1 1 -1 -1 -1 -1 -1]);
        dn=dn(1,1:4000);
        x=v1(1,1:4000);
        N=length(x);
        for n=M:N
            x1=x(n:-1:n-M+1);
            y(n)=w*x1';
            e(n,m)=dn(n)-y(n);
            w=w+mu*e(n,m)*x1;
        end;
        y=zeros(1,N);
    end;
    Jav=sum((e.^2),2)/av;
end;
```

Figure P9.1.4.2 shows the resulting average learning curves.
The results are shown in Figure of P9.3.1. The following constants and signals were used: $\mu = 0.01$, $M = 6$, $dn(n) = 0.995^n \cos(0.1\pi n)$, $x(n) = 0.995^n \cos(0.1\pi n) + 0.2 \text{randn}(1,1500)$, $n = 1:1500$.

Taking the gradient of $J(n)$, we obtain

$$\nabla w J(n) = -e(n)x(n) + \gamma w(n) \Rightarrow w(n+1) = w(n) - \mu \nabla w J(n)$$

(steepest descent algorithm)

$$= w(n) - \mu [-e(n)x(n) + \gamma w(n)] = (1 - \mu \gamma)w(n) + \mu e(n)x(n)$$

The normalized leaky LMS algorithm is given as follows:

```matlab
function [w,y,e,J,w1]=lms_normalized_leaky_lms(x,dn,mu,gama,M)
    x1=x(n:-1:n-M+1);
y(n)=w'*x1;
end
```

FIGURE P9.1.4.2

9.3.1 The results are shown in Figure of P9.3.1. The following constants and signals were used: $\mu = 0.01$, $M = 6$, $dn(n) = 0.995^n \cos(0.1\pi n)$, $x(n) = 0.995^n \cos(0.1\pi n) + 0.2 \text{randn}(1,1500)$, $n = 1:1500$.

9.11.1 Taking the gradient of $J(n)$, we obtain

$$\nabla w J(n) = -e(n)x(n) + \gamma w(n) \Rightarrow w(n+1) = w(n) - \mu \nabla w J(n)$$

(steepest descent algorithm)

$$= w(n) - \mu [-e(n)x(n) + \gamma w(n)] = (1 - \mu \gamma)w(n) + \mu e(n)x(n)$$

9.11.2 The normalized leaky LMS algorithm is given as follows:

```matlab
function [w,y,e,J,w1]=lms_normalized_leaky_lms(x,dn,mu,gama,M)
    %function [w,y,e,J,w1]=lms_normalized_leaky_lms(x,dn,mu,gama,M);
    %all signals are real valued;x=input to filter;
    %y=output from the filter;dn=desired signal;
    %mu=step-size factor;gama=gamma factor<<1;
    %M=number of filter coefficients;w1=matrix whose M
    %rows give the history of each filter coefficient;

    N=length(x);
y=zeros(1,N);
w=zeros(1,M);
    for n=M:N
        x1=x(n:-1:n-M+1);
y(n)=w'*x1;
    end
```
Variants of Least Mean-Square Algorithm

\[ e(n) = d(n) - y(n); \]
\[ w = (1 - \mu \cdot \gamma) \cdot w + \mu \cdot e(n) \cdot x_1 / (0.0001 + x_1 \cdot x_1'); \]
\[ w_1(n-M+1,:) = w(1,:); \]
end;
\[ J = e^2; \]

The reader will find that the normalized leaky LMS algorithm performs better.

9.12.1

\[ J = E\{[d(n) - w^T(n)x(n)][d(n) - x^T(n)w(n)] + \lambda(c^Tw(n) - a)\} \]
\[ = E\{d^2(n) - d(n)w^T(n)x(n) - d(n)x^T(n)w(n) + w^T(n)x(n)x^T(n)w(n) \]
\[ + \lambda c^Tw(n) - \lambda a\} \]
\[ = \sigma_d^2 - w^T(n)E[d(n)x(n)] - wE[d(n)x(n)] + w^T(n)E[x(n)x'(n)]w(n) \]
\[ + \lambda c^Tw(n) - \lambda a \]
\[ = \sigma_d^2 - 2w^T(n)p_d + w^T(n)\sigma_d^2 w(n) + \lambda c^Tw(n) - \lambda a \]
\[ = \sigma_d^2 - 2w^T(n)p_d + [w(n) - w^o]^T \sigma_d^2 R_o[w(n) - w^o] + w^o^T R_o w^o \]
\[ + w^T(n)\sigma_o^2 w^o + w^o^T R_o w^o + \lambda[c^T[w(n) - w^o] - (a - c^T w^o)] \]
\[ = \sigma_d^2 - w^o^T p_d + \xi^T R_o \xi + \lambda(c^T \xi - a') \]

FIGURE P9.3.1

Input signal \( x(n) \)

First-stage output

Second-stage output

0
0
0

-2
-0.5
1

-2
-0.5
1

0
0
0

-2
-0.5
1

FIGURE P9.3.1

The reader will find that the normalized leaky LMS algorithm performs better.

9.12.1

\[ J = E\{[d(n) - w^T(n)x(n)][d(n) - x^T(n)w(n)] + \lambda(c^Tw(n) - a)\} \]
\[ = E\{d^2(n) - d(n)w^T(n)x(n) - d(n)x^T(n)w(n) + w^T(n)x(n)x^T(n)w(n) \]
\[ + \lambda c^Tw(n) - \lambda a\} \]
\[ = \sigma_d^2 - w^T(n)E[d(n)x(n)] - wE[d(n)x(n)] + w^T(n)E[x(n)x'(n)]w(n) \]
\[ + \lambda c^Tw(n) - \lambda a \]
\[ = \sigma_d^2 - 2w^T(n)p_d + w^T(n)\sigma_d^2 w(n) + \lambda c^Tw(n) - \lambda a \]
\[ = \sigma_d^2 - 2w^T(n)p_d + [w(n) - w^o]^T \sigma_d^2 R_o[w(n) - w^o] + w^o^T R_o w^o \]
\[ + w^T(n)\sigma_o^2 w^o + w^o^T R_o w^o + \lambda[c^T[w(n) - w^o] - (a - c^T w^o)] \]
\[ = \sigma_d^2 - w^o^T p_d + \xi^T R_o \xi + \lambda(c^T \xi - a') \]
where:

\[ R_w w^o = p_{dx}, \quad [w^{oT} R_w w(n)]^T = w^{T}(n) R_w w^o \quad (R_w = \text{symmetric}) \]

9.12.2 We write the cost function as follows:

\[
J(n) = [w(n+1) - w(n)]^T [w(n+1) - w(n)] + \lambda [d(n) - w(n+1) x(n)]
\]  
(1)

Differentiate (1) with respect to \( w(n+1) \) (see also Appendix 2) to obtain

\[
\frac{\partial J(n)}{\partial w(n+1)} = 2[w(n+1) - w(n)] - \lambda x(n) = 0 \Rightarrow w(n+1) = w(n) + \frac{1}{2} \lambda x(n)
\]  
(2)

Substitute (2) in the constraint \( d(n) = w^{T}(n+1) x(n) \) to obtain

\[
d(n) = \left[ w(n) + \frac{1}{2} \lambda x(n) \right] x(n) = w^{T}(n) x(n) + \frac{1}{2} \lambda \|x(n)\|^2
\]  
(3)

But \( e(n) = d(n) - w^{T}(n) x(n) \) and (3) becomes

\[
\lambda = \frac{2e(n)}{\|x(n)\|^2}
\]  
(4)

Substituting (4) in (2), we obtain

\[
w(n+1) = w(n) + \frac{1}{\|x(n)\|^2} e(n)x(n)
\]  
(5)

From (5), we obtain the final form by introducing the step-size factor \( \mu \) to control the change in weight vector.

9.13.1 The Book m-function FIR LMF algorithm is given as follows:

```matlab
function [w, y, e, J, w1] = lms_least_mean_fourth(x, dn, mu, M)
    N = length(x); w = zeros(1, M);
    for n = M:N
        x1 = x(n:-1:n-M+1);
        y(n) = w*x1';
        e(n) = dn(n) - y(n);
        w = w + mu*x1*e(n)^3;
        w1(n-M+1,:) = w(1,:);
    end;
    J = e.^2;
end;
```

9.14.1 The Book m-function for the LMMN LMS algorithm is given below:

```matlab
function [w, y, e, J, w1] = lms_least_mean_mixed_norm(x, dn, mu, M, delta)
    N = length(x); w = zeros(1, M); % 0 < delta < 1;
    ```
Variants of Least Mean-Square Algorithm

for n=M:N

\[ x_1 = x(n:-1:n-M+1); \]
\[ y(n) = w' \times x_1; \]
\[ e(n) = d(n) - y(n); \]
\[ w = w + \mu \times x_1 \times e(n) \times (\text{delta} + (1-\text{delta}) \times e(n)^2); \]
\[ w_{l(n-M+1,:)} = w(l,:); \]
end;
\[ J = e.^2; \]

To obtain the normalized form of the above algorithm, divide the last term of line 7 with the expression \((\varepsilon + x_1^t \times x_1)\).

9.16.1 The \(k\)th value is

\[ W_{i+1,k} = W_{i,k} + \mu X_i \times E_i = W_{i,k} + \mu X_i \times [D_i(k) - Y_i(k)] = W_{i,k} + \mu X_i \times [D_i(k) - W_{i,k} Y_i(k)] \]

\[ = W_{i,k} + \mu X_i \times D_i(k) - \mu W_{i,k} \times |X_i|^2 = \left(1 - \mu |X_i|^2\right) W_{i,k} + \mu X_i \times D_i(k) \]

9.16.2 The \(z\)-transform and the ensemble operator are linear operations and can be interchanged. Therefore, the \(z\)-transform is

\[ z W_{i,k}(z) = z W_{0,k} = \left(1 - \mu E\left[|X_i(k)|^2\right]\right) W_{i,k}(z) + \left[\frac{\mu E\left[D_i(k)X_i^*(k)\right]}{\left(1 - z^{-1}\right)}\right] \]

(1)

In the above equation, \(W_{0,k} = 0\) since it was assumed that the initial conditions have zero values and \(W_{i,k}(z) = \sum_{i=0}^{\infty} E\left[W_{i,k}\right] z^{-1}\). Multiplying (1) by \(z^{-1}\) and \((z - 1)\), and applying the final value theorem (see Table 3.3.1), we obtain

\[ E\left[W_{k}^\infty\right] = \lim_{z \to 1^-} (z - 1) W_{k}(z) \]

\[ = \lim_{z \to 1^-} \frac{\mu E\left[D_i(k)X_i^*(k)\right]}{\left(1 - z^{-1}\right)} = \frac{E\left[D_i(k)X_i^*(k)\right]}{E\left[|X_i(k)|^2\right]} \]

(2)

9.16.3

\[ E_{i+1}(k) = E\left[W_{i+1,k}\right] - E\left[W_{k}^\infty\right] \]

\[ = (1 - 2 \mu E\left[|X_i(k)|^2\right]) E\left[W_{i,k}\right] + 2 \mu E\left[D_i(k)X_i^*(k)\right] - \frac{E\left[D_i(k)X_i^*(k)\right]}{E\left[|X_i(k)|^2\right]} \]

\[ = (1 - 2 \mu E\left[|X_i(k)|^2\right]) E\left[W_{i,k}\right] - (1 - 2 \mu E\left[|X_i(k)|^2\right]) \frac{E\left[D_i(k)X_i^*(k)\right]}{E\left[|X_i(k)|^2\right]} \]

\[ = (1 - 2 \mu E\left[|X_i(k)|^2\right]) E_{i}(k) \]
9.18.1

\[
\left[ \alpha \|e(n)\|^2 \mathbf{I} + \gamma \mathbf{x}(n) \mathbf{x}^T(n) \right]^{-1} \mathbf{x}(n) = \alpha^{-1} \|e(n)\|^2 \mathbf{I} \mathbf{x}(n) - \alpha^{-1} \|e(n)\|^2 \mathbf{I} \mathbf{x}(n)
\]

\[
\frac{\mathbf{x}^T(n) \alpha^{-1} \|e(n)\|^2 \mathbf{x}(n)}{\gamma^{-1} + \mathbf{x}^T(n) \alpha^{-1} \|e(n)\|^2 \mathbf{x}(n)}
\]

\[
= \frac{\gamma^{-1} \alpha^{-1} \|e(n)\|^2 \mathbf{I} \mathbf{x}(n) + \alpha^{-1} \|e(n)\|^2 \mathbf{I} \mathbf{x}(n) \mathbf{x}^T(n) \alpha^{-1} \|e(n)\|^2 \mathbf{x}(n)}{\gamma^{-1} + \mathbf{x}^T(n) \alpha^{-1} \|e(n)\|^2 \mathbf{x}(n)}
\]

\[
-\alpha^{-1} \|e(n)\|^2 \mathbf{I} \mathbf{x}(n) \mathbf{x}^T(n) \alpha^{-1} \|e(n)\|^2 \mathbf{x}(n)
\]

\[
= \frac{\|e(n)\|^2 \mathbf{x}(n)}{\alpha + \mathbf{x}^T(n) \gamma \|e(n)\|^2 \mathbf{x}(n)} = \frac{\mathbf{x}(n)}{\alpha \|e(n)\|^2 \gamma \mathbf{x}^T(n) \mathbf{x}(n)}
\]
Appendix 1

Suggestions and Explanations for MATLAB Use

A1.1 SUGGESTIONS AND EXPLANATIONS FOR MATLAB USE

It is suggested that before start using the text, the reader, who does not have a lot of experience with MATLAB, goes over this appendix and tries to execute the presented material in MATLAB.

A1.1.1 CREATING A DIRECTORY

It was found by the author that it is less confusing if for a particular project we create our own directory where our own developed MATLAB m-files are stored. However, any time we need anyone of these files, we must include the directory in the MATLAB path. Let us assume that we have the following directory path: c:\ap\sig-syst\ssmatlab. The following two approaches can be used:

```matlab
>> cd 'c:\ap\sig-syst\ssmatlab'
```

or

```matlab
>> path(path,'c:\ap\sig-syst\ssmatlab')
```

%remember to introduce %the path any time you start new MATLAB operations;
%the % symbol is necessary for the MATLAB to ignore the %explanations;

The MATLAB files are included in the “ssmatlab” directory.

A1.1.2 HELP

In case we know the name of a MATLAB function and we would like to know how to use it, we write the following command in the Command window:

```matlab
>> help sin
```

or

```matlab
>> help exp
```

etc.

In case we want to look for a key word, we write

```matlab
>> look for filter
```
A1.1.3 Save and Load

When we have created many variables in the Command window and we would like to save two of them in a particular directory and in a particular file, we proceed as follows:

```matlab
>> cd 'c:\ap\matlabdata'
>> save data1 x dt % it saves in the matlabdata directory the file data1 having the two variables x and dt;
```

Let us assume now that we want to bring these two variables in the working space to use them. We first change the directory, as we did earlier, and then we write in the Command window

```matlab
>> load data1
```

Then, the two variables will appear in the working space ready to be used.

A1.1.4 MATLAB as Calculator

```matlab
>> pi^pi-10;
>> cos(pi/4);
>> ans*ans; % the result will be \((\sqrt{2}/2) \times (\sqrt{2}/2) = 1/2\) because the first output is eliminated, only the last output is kept in the form of ans;
```

A1.1.5 Variable Names

```matlab
>> x = [1 2 3 4 5];
>> dt = 0.1;
>> cos(pi*dt); % since no assignment takes place there is no variable;
```

A1.1.6 Complex Numbers

```matlab
>> z = 3+j*4; % note the multiplication sign;
>> z*s; % or \(z^2\) will give you the same results;
>> rz = real(z); iz = imag(z); % will give \(rz = 3\), and \(iz = 4\);
>> az = angle(z); abz = abs(z); % will give \(az = 0.9273\) rad, % and \(abz = 5\);
>> x = exp(-z)+4; % \(x = 3.9675 + j0.0377\);
```

A1.1.7 Array Indexing

```matlab
>> x = 2:1:6; % \(x\) is an array of the numbers \{2, 3, 4, 5, 6\};
>> y = 2:-1:-2; % \(y\) is an array of the numbers \{2, 1, 0, -1, -2\};
>> z = [1 3 y]; % \(z\) is an array of the numbers \{1, 3, 2, 1, 0, -1, -2\}; note the required space between array numbers;
```
Appendix 1

303

>>xt2=2*x;%xt2 is an array of numbers of x each one multiplied by 2;
>>xty=x.*y;%xty is an array of numbers which are the result of multiplication of corresponding elements, that is, \{4, 3, 0, -5, -12\};

A1.1.8 Extracting and Inserting Numbers in Arrays

>>x=2:1:6;
>>y=[x zeros(1,3)];%y is an array of the numbers \{2, 3, 4, 5, 6, 0, 0, 0\};
>>z=y(1,3:7);%1 stands for row 1 which y is and 3:7 instructs %to keep columns 3 through 7 the result is the array \{4, 5, 6, 0, 0\};
>>lx=length(x);%lx is the number equal to the number of columns %of the row vector x, that is lx = 5;
>>x(1,2:4)=4.5*[1 2 3];%this assignment substitutes the elements %of x at column positions 2, 3 and 4 with the numbers %\{4.5, 9, 13.5\},note the columns of %\{2:4 and 1:3\} are the same;
>>x(1,2:2:length(x))=pi;%substitutes the columns 2 and 4 of x %with the value of pi, hence the array is \{2, 3.1416, 4,3.1416 6\};

A1.1.9 Vectorization

>>n=0:0.2:1;
>>s=sin(0.2*pi*n);%the result of these two commands gives the %signal s (sine function) at times (values of n) %0, 0.2, 0.4, 0.6, 0.4, 1;

This approach is preferable since MATLAB executes the vectorization approach faster rather than the loop approach, which is given as follows:

>>s=[];%initializes all values of vector s to zero;
>>for n=0:0.5%note that the index must be integer;
>>s(n+1)=sin(0.2*pi*n*0.2);%since we want values of s %every 0.2 seconds we must multiply n by 0.2; %note also that for n = 0 the variable becomes %s(1) and this is because the array in MATLAB %always starts counting columns from 1;
>>end;

The results are identical with the previous one.
A1.1.10  Windowing

The following windows are used for correcting, to some extent, the effects of truncating a vector.

@bartlett—Bartlett window
@barthannwin—Modified Bartlett–Hanning window
@blackman—Blackman window
@blackmanharris—Minimum four-term Blackman–Harris window
@bohmanwin—Bohman window
@chebwin—Chebyshev window
@flattopwin—Flat-top window
@gausswin—Gaussian window
@hamming—Hamming window
@hann—Hann window
@kaiser—Kaiser window
@nuttallwin—Nuttall-defined minimum four-term Blackman–Harris window
@parzenwin—Parzen (de la Vallée-Poussin) window
@rectwin—Rectangular window
@tukeywin—Tukey window
@triang—Triangular window

An example for windowing is given as follows:

\[
\begin{align*}
N &= 65; \\
\text{w} &= \text{window}(@\text{blackmanharris},N); \\
\text{w1} &= \text{window}(@\text{hamming},N); \\
\text{w2} &= \text{window}(@\text{gausswin},N,2.5); \\
\text{plot}(1:N,[w,w1,w2]) \text{; axis([1 N 0 1])}; \\
\text{legend('Blackman-Harris','Hamming','Gaussian');}
\end{align*}
\]

A1.1.11  Matrices

If \(a\) and \(b\) are matrices such that \(a\) is a \(2 \times 3\) matrix and \(b\) is a \(3 \times 3\) matrix, then \(c = a \cdot b\) is a \(2 \times 3\) matrix.

\[
\begin{align*}
\text{a} &= \begin{bmatrix} 1 & 2 \\ 4 & 6 \end{bmatrix}; \text{a is a 2x2 matrix} \\
\text{b} &= \text{a}'; \text{b is a transposed 2x2 matrix of a and is} \begin{bmatrix} 1 & 4 \\ 2 & 6 \end{bmatrix}; \\
\text{da} &= \text{det}(a); \text{da is a number equal to the determinant of a, da} = -2; \\
\text{c} &= \text{a(:)}; \text{c is a vector which is made up of the columns of a, } c = [1 4 2 6]; \\
\text{ia} &= \text{inv}(a); \text{ia is a matrix which is the inverse of a;}
\end{align*}
\]
>>sal=sum(a,1); % sal is a row vector made up of the sum of the % rows, sal = [5 8];
>>sa2=sum(a,2); % sa2 is a column vector made up by the sum of % the columns, sa2 = [3 10]';

A1.1.12 Producing a Periodic Function

>>x=[1 2 3 4];
>>xm=x'*ones(1,5); % xm is 4x5 matrix and each of its column is x';
>>xp=xm(:)'; % xp is a row vector, xp = [x x x x x];

A1.1.13 Script Files

Script files are m-files in which we introduce their names in the Command window and receive the results. We must, however, have the directory that includes the file in the MATLAB search directories. You can modify the file in any desired way and get new results. Suppose that any time we ask for the file pexp.m, the magnitude and the angle of the exponential function $e^{j\omega}$ are plotted. To accomplish this, we first go to the Command window and open a new m-file. In the window, we type the file as shown below. As soon as we finish typing, we click on Save as and save the file in, say, c:\ap\ssmatlab. If we want to see the results, in the Command window we just write pexp and hit the Enter key.

Script File pexp.m

```matlab
>>w=0:pi/500:pi-pi/500; % they are 500 at pi/500 apart;
>>x=exp(j*w); ax = abs(x); anx = angle(x);
>>subplot(2,1,1); plot(w,ax,'k'); '%k' means plot line in black;
>>xlabel('\omega rad/s'); ylabel('Magnitude');
>>subplot(2,1,2); plot(w,anx,'k');
>>xlabel('\omega rad/s'); ylabel('Angle');
```

If we have the function $2e^{j\omega}/(e^{j\omega} - 0.5)$ and want to plot the results as earlier, we substitute in the script file the function $x$ with the function: $x=2*exp(j*w)./(exp(j*w)-0.5);$.

In the above MATLAB expression, note the dot before the slash. This instructs MATLAB to operate at each value of $w$ separately and, thus, gives results at each frequency point.

A1.1.14 Functions

We present here an example of how to write functions. The reader should also study the functions that are presented throughout the book. In the Fourier series, for example, we plot the functions of the form:

$$s(t) = \sum_{n=0}^{N} A_n \cos(n\omega_0 t)$$
and plot this sum of cosines, each one having a different amplitude and frequency. Let \( A = [1 \ 0.6 \ 0.4 \ 0.1] \), \( \omega_0 = 2 \), and \( 0 \leq t \leq 4 \). We approach this solution by vectorizing the summation. The MATLAB function is of the form:

\[
\text{function}[s]=\text{sumofcos}(A,N,\omega_0,\text{rangeoft})
\]

\[
n=0:N-1;
\]

\[
s=A*\cos(\omega_0*n'*\text{rangeoft});
\]

% when we want to use this function at the command window to % find s we write for example:

\[
\%>>A = [1 0.6 0.4 0.1];N = 4;\omega_0 = 2;\text{rangeoft} = 0:0.05:6;
\]

\[
\%>>[s] = \text{sumofcos}(A,N,\omega_0,\text{rangeoft});
\]

% at the Enter key click the vector s is one of the % variables in the command window and it can be plotted % at the wishes of the reader; we % must secure that the directory in which sumofcos function % exists is in the MATLAB % path; After you type the function in the Editing window, % you save as . in the directory, for example, % c:\ap\ssmatlab and filename: sumofcos.m.

It is recommended that the reader set small numbers for \( N(N=4) \) and the range of \( t(0:0.2:1) \) and produce first the matrix \( \cos(\omega_0*n'*t) \) and then see the result \( A*\cos(\omega_0*n'*t) \).

### A1.1.15 Complex Expressions

We can produce the results by writing, for example,

\[
>>\text{x}=[1 \ 3 \ 1 \ 5 \ 3 \ 4 \ 5 \ 8];
\]

\[
>>\text{plot(}\text{abs}(\text{fft(}\text{x},256)),\text{'r'}\%\text{will plot in red color the}
\text{spectrum of the vector x of 256 points;}
\]

### A1.1.16 Axes

\[
>>\text{axis([xmin xmax ymin ymax]);%sets the max and min values of}
\text{the axes;}
\]

\[
>>\text{grid on;%turns on grid lines in the graph;}
\]

### A1.1.17 2D Graphics

*To plot a sine and a cosine signal*

\[
>>\text{x=}\text{linspace}(0,2*\text{pi},40);\%\text{produces 40 equal spaced points}
\text{between 0 and 2}\pi;
\]

\[
>>\text{y=}\text{sin(}\text{x};\text{plot(}\text{x},\text{y,}'r'\%\text{will plot the sine signal with red}
\text{color;}
\]
\% will plot the cosine signal with green color;

\textit{For other color lines}

'y'=yellow, 'c'=cyan, 'b'=blue, 'w'=white, 'k'=black

\textit{Type of lines}

'g:'=green dotted line, 'r-'=red dashed line, 'k-- x'=black dashed line with x's, 'k--.'=black dash-dot line, '+'=plus sign, 'ko'=black circles

\textit{Add Greek letters}

\textbackslash omega will produce Greek lowercase omega, \textbackslash Omega will produce Greek uppercase omega. The same is true for the rest of the Greek letters. For example, if we want to write the frequency in a figure under the x-axis, in the Command window we write:

\texttt{>>xlabel('\textbackslash omega rad/s');}

For an omega with a subscript 01, we write:

\texttt{>>xlabel('\omega_{01} rad/s');}

\textit{Add grid lines}

\texttt{>>grid on; \% this is done after the command plot;}

\textit{Adjusting axes}

\texttt{>>axis square; \% sets the current plot to be square rather than the default rectangle;}
\texttt{>>axis off; \% turn off all axis labeling, grid, and tick marks}
\texttt{\% marks, leave the title and any labels placed \% by the 'text' and 'gtext' commands;}
\texttt{>>axis on; \% turn on axis labeling, tick marks, and grid;}
\texttt{>>axis([xmin xmax ymin ymax]); \% set the maximum and \% minimum values of the axes using values given \% in the row vector;}

\textit{Subplots (Example)}

\texttt{>>n=0:100; x=sin(n*pi*n); y=cos(n*pi*n); z=x.*y; w=x+y;}
\texttt{ % subplot(2,2,1); plot(n,x); subplot(2,2,2); plot(n,y); \% subplot(2,2,3); plot(n,z); subplot(2,2,4); plot(n,w);}

\textit{Log plotting}

\texttt{>>semilogx(x); \% will plot the vector x in log scale in x-axis \% and linear scale in y-axis;
Appendix 1

>>semilogy(x); % will plot the vector x in log scale in y-direction and linear scale in the x-axis;
>>loglog(x); % will plot the vector x in log scale both axes;

Histogram

>>x=randn(1,1000); hist(x,40); colormap([0 0 0]); % will plot a Gaussian histogram of 40 bars in white; if instead we entered the vector [1 1 1] the bars would be black; the vector [1 0 0] will give red and the vector [0.5 0.5 0.5] will give gray;
>>x=-3.0:0.05:3; y=exp(-.*x); bar(x,y); colormap([.5 .5 .5]); % will produce bar figure of the bell curve with gray color;
>>stairs(x,y,'k'); % will produce a stair-like black curve;

Add words

>>gtext('the word');

At the return, a crosshair will appear in the figure. Move the center at the point in the figure where the word must start and click.

Add legend

>>plot(x1,y1,'+',x2,y2,'*'); % there will be two curves in the graph;
>>legend('Function 1','Function 2');

The following rectangle will appear in the figure:

+ Function 1
* Function 2

A1.1.18 3D Plots

A1.1.18.1 Mesh-Type Figures
If, for example, we desire to plot the function \( f(x) = e^{-(x^2+y^2)} \) in the ranges \(-2 \leq x \leq 2\) and \(-2 \leq y \leq 2\), we proceed as follows:

>>x=-2:0.1:2; y=-2:0.1:2; [X,Y]=meshgrid(x,y);
>>f=exp(-(X.*X+Y.*Y)); mesh(X,Y,f);colormap([0 0 0]);

The above commands will produce a mesh-type 3D figure with black lines.
A1.2 GENERAL PURPOSE COMMANDS

A1.2.1 MANAGING COMMANDS AND FUNCTIONS

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>help</code></td>
<td>Online help for MATLAB functions and m-files, for example, <code>&gt;&gt;help plot</code></td>
</tr>
<tr>
<td><code>path</code></td>
<td>Shows the path to MATLAB directories which are available at the command window</td>
</tr>
</tbody>
</table>

A1.2.2 MANAGING VARIABLES AND WORKPLACE

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clear</code></td>
<td>Removes all the variables and items in the memory. Let us assume that the memory contains the variables ( x, y, z ). Then, write <code>&gt;&gt;clear x z</code>; only ( y ) will remain in the memory.</td>
</tr>
<tr>
<td><code>length</code></td>
<td>A number that gives the length of a vector. <code>&gt;&gt;x=[1 3 2 5];</code> then <code>&gt;&gt;length(x)</code> will give the number 4. If we write <code>&gt;&gt;y=length(x);</code>, then we get the value of variable ( y ) equal to 4.</td>
</tr>
<tr>
<td><code>size</code></td>
<td>Shows the array dimensions. <code>&gt;&gt;x=[1 3 2 5];</code> then <code>size(x)</code> will give the numbers ( 1 \times 4 ), which means 1 row and 4 columns. Let us write <code>&gt;&gt;x=[1 2 3 5; 6 4];</code> then <code>size(x)</code> will give the numbers ( 2 \times 4 ), which means that ( x ) is a matrix of ( 2 \times 4 ) dimensions</td>
</tr>
<tr>
<td><code>who</code></td>
<td>Produces a list of the variables in the memory.</td>
</tr>
<tr>
<td><code>format</code></td>
<td>Used for display as follows: <code>&gt;&gt;format short,pi;</code> will produce the number 1.1416, <code>&gt;&gt;format long,pi;</code> will produce the number 3.14159265358979, <code>&gt;&gt;format long,single(pi);</code> will produce the number 3.1415927.</td>
</tr>
</tbody>
</table>

A1.2.3 OPERATORS AND SPECIAL CHARACTERS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code></td>
<td>Plus</td>
</tr>
<tr>
<td><code>-</code></td>
<td>Minus</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Number and matrix multiplications</td>
</tr>
<tr>
<td><code>.*</code></td>
<td>Array multiplication, for example, <code>&gt;&gt;x=[1 2 3];y=[2 3 4];z=x.*y;</code>. Hence, ( z=[2 6 12] )</td>
</tr>
<tr>
<td><code>^</code></td>
<td>Array power, for example, <code>&gt;&gt;x=[2 3 4];y=x.^3;</code>. Hence, ( y=[8 27 64] ). <code>&gt;&gt;x=[2 4;1 5]; y=x.^2;</code>. Hence, ( y=[4 16;1 25] )</td>
</tr>
<tr>
<td><code>/</code></td>
<td>Right division</td>
</tr>
<tr>
<td><code>./</code></td>
<td>Array division, for example, <code>&gt;&gt;x=[2 4 6];y=[4 4 12];z=x./y;</code>. Hence, ( z=[0.5 1 0.5] )</td>
</tr>
<tr>
<td><code>:</code></td>
<td>Colon, for example, <code>&gt;&gt;x=[1 4 6 7 8 9 ... 2 5 8 1];</code> The vector ( x ) is interpreted by MATLAB as a row vector having 10 elements</td>
</tr>
<tr>
<td><code>%</code></td>
<td>Comments, for example, <code>&gt;&gt;x=[1 4 2 6];%this is a vector. The MATLAB ignores “this is a vector”</code></td>
</tr>
</tbody>
</table>
| `'` | Transpose of a matrix or vector, for example, `>>x=[2 6 3];y=x';` will have \( y = \begin{bmatrix} 2 \\ 6 \\ 3 \end{bmatrix} \) (Continued)
& Logical AND
| Logical OR
~ Logical NOT
xor Logical exclusive (XOR)

A1.2.4 **CONTROL FLOW**

**for** Repeat operations a specific number of times.

```
>>for n=0:3
    >>x(n+1)=sin(n*pi*0.1); % observe the n+1, if the
    +%1 was not there, x(0) is not defined by MATLAB
    >>end;
```

Then, \( x = [0 \ 0.3090 \ 0.5878 \ 0.8090] \)

```
>>for n=0:2
    >> for m=0:1
    >>     x(n+1,m+1)=n^2+m^2;
    >> end;
    >>end;
```

Then, \( x = \begin{bmatrix} 0 & 1 \\ 1 & 2 \\ 4 & 5 \end{bmatrix} \)

**while** Repeats statements an indefinite times of times

```
>>a=1;num=0;
>>while (1+a)<=2 & (1+a)>=1.0001
    a=s/2;
    num=num+1;
>>end;
```

We obtain \( a = 0.0001 \) and \( num = 14 \)

**if** Executes statements conditionally

```
if expression
    commands evaluated if true
else
    commands evaluated if false
end;
```

If there are more than one alternative, the `if-else-end` statement takes the form:
elseif
  if expression1
    commands evaluated if expression 1 is true
  elseif expression2
    commands evaluated if expression 2 is true
elseif...
  .
  .
else
  commands evaluated if no other expression is true
end;

A1.3 ELEMENTARY MATRICES AND MATRIX MANIPULATION

A1.3.1 ELEMENTARY MATRICES AND ARRAYS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eye(n,n)</code></td>
<td>Identity matrix (its diagonal elements are 1 and all the others are 0)</td>
</tr>
<tr>
<td><code>linspace</code></td>
<td>Generates 100 equally spaced points between <code>x1</code> and <code>x2</code></td>
</tr>
<tr>
<td><code>ones</code></td>
<td>Generates a row vector with its elements only ones</td>
</tr>
<tr>
<td><code>rand</code></td>
<td>Uniformly distributed random numbers. <code>&gt;&gt;x=rand(1,5);</code> where <code>x</code> is a row vector of 5 elements of random numbers.</td>
</tr>
<tr>
<td><code>zeros</code></td>
<td>Creates arrays and matrices of all zeros. <code>&gt;&gt;x=zeros(1,4);</code> where <code>x</code> is a row vector of 4 elements all with zero value.</td>
</tr>
<tr>
<td><code>:(colon)</code></td>
<td>Regularly spaced vector. <code>&gt;&gt;x = [1 4 2 5 8 3]; y = x(1:3:6);</code> Hence, <code>y</code> = [1 5 8 3];</td>
</tr>
<tr>
<td><code>eps</code></td>
<td>Floating-point relative accuracy. <code>&gt;&gt;n=-4:4;x=sin(n*pi*.1)./((n*pi+eps);</code></td>
</tr>
<tr>
<td><code>i,j</code></td>
<td>Imaginary unit</td>
</tr>
<tr>
<td><code>pi</code></td>
<td>Ratio of a circle’s circumference to its diameter</td>
</tr>
</tbody>
</table>

A1.3.2 MATRIX MANIPULATION

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>diag</code></td>
<td>Diagonal matrices and diagonals of a matrix. <code>&gt;&gt;x=[1 3 5;2 6 9;4 7 0]; y=diag(x);</code> will give a column vector <code>y</code> = [1 6 0]'</td>
</tr>
</tbody>
</table>
### A1.4 ELEMENTARY MATHEMATICAL FUNCTIONS

#### A1.4.1 Elementary Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Absolute value of a number and the magnitude of a complex number</td>
</tr>
<tr>
<td>acos, acosh</td>
<td>Inverse cosine and inverse hyperbolic cosine</td>
</tr>
<tr>
<td>acot, acoth</td>
<td>Inverse cotangent and inverse hyperbolic cotangent</td>
</tr>
<tr>
<td>acsc, acsch</td>
<td>Inverse cosecant and inverse hyperbolic cosecant</td>
</tr>
<tr>
<td>angle</td>
<td>Phase angle of a complex number, for example, angle(1 + j) = 0.7854</td>
</tr>
<tr>
<td>asec, asech</td>
<td>Inverse secant and inverse hyperbolic secant</td>
</tr>
<tr>
<td>asin, asinh</td>
<td>Inverse sine and inverse hyperbolic sine</td>
</tr>
<tr>
<td>atan, atanh</td>
<td>Inverse tangent and inverse hyperbolic tangent</td>
</tr>
<tr>
<td>ceil</td>
<td>Round toward infinity, for example, ceil(4.22) = 5</td>
</tr>
<tr>
<td>conj</td>
<td>Complex conjugate, for example, conj(2 + j * 3)</td>
</tr>
<tr>
<td>cos, cosh</td>
<td>Cosine and hyperbolic cosine</td>
</tr>
<tr>
<td>cot, coth</td>
<td>Cotangent and hyperbolic cotangent</td>
</tr>
<tr>
<td>csc, csch</td>
<td>Cosecant and hyperbolic cosecant</td>
</tr>
<tr>
<td>exp</td>
<td>Exponential, for example, exp(-1) = 1/e = 0.3679</td>
</tr>
<tr>
<td>fix</td>
<td>Rounds toward zero, for example, fix(-3.22) = -3</td>
</tr>
<tr>
<td>floor</td>
<td>Round toward minus infinity, for example, floor(-3.34) = -4, and floor(3.65) = 3</td>
</tr>
<tr>
<td>imag</td>
<td>Imaginary part of a complex number, for example, imag(2 + j * 5)</td>
</tr>
<tr>
<td>log</td>
<td>Natural logarithm, for example, log(10) = 2.3026</td>
</tr>
<tr>
<td>log2</td>
<td>Based 2 logarithm, for example, log2(10) = 3.3219</td>
</tr>
<tr>
<td>log10</td>
<td>Common (base 10) logarithm, for example, log10(10) = 1</td>
</tr>
<tr>
<td>mod</td>
<td>Modulus (signed remainder after division), for example, mod(10, 3) = 1, mod(10, 4) = 2.</td>
</tr>
<tr>
<td></td>
<td>In general, mod(x, y) = x - n * y</td>
</tr>
<tr>
<td>real</td>
<td>Real part of complex number</td>
</tr>
<tr>
<td>rem</td>
<td>Remainder after division, for example, rem(10, 3) = 1, rem(10, 5) = 0, rem(10, 4) = 2</td>
</tr>
<tr>
<td>round</td>
<td>Round to the nearest integer, for example, round(3.22) = 3, round(3.66) = 4</td>
</tr>
<tr>
<td>sec, sech</td>
<td>Secant and hyperbolic secant</td>
</tr>
<tr>
<td>sign</td>
<td>Signum function, for example, sign(x) = 0 for x = 0, sign(x) = 1 for x &gt; 0, and sign(x) = -1 for x &lt; 1</td>
</tr>
<tr>
<td>sin, sinh</td>
<td>Sine and hyperbolic sine</td>
</tr>
<tr>
<td>sqrt</td>
<td>Square root, for example, sqrt(4) = 2</td>
</tr>
<tr>
<td>tan, tanh</td>
<td>Tangent and hyperbolic tangent</td>
</tr>
<tr>
<td>erf, erfc</td>
<td>Error and co-error function</td>
</tr>
<tr>
<td>gamma</td>
<td>Gamma function, for example, gamma(6) = 120 or 1 * 2 * 3 * 4 * (6 – 1) = 120</td>
</tr>
</tbody>
</table>
A1.5 NUMERICAL LINEAR ALGEBRA

A1.5.1 Matrix Analysis

- **det**: Matrix determinant, for example, >>>a=[1 2; 3 4]; det(a); gives -2
- **norm**: The norm of a vector, for example, norm(v)=sum(abs(v).^2)^(1/2)
- **rank**: Rank of a matrix, rank(A) provides the number of independent columns or rows of matrix A
- **trace**: Sum of the diagonal elements, for example, trace([1 3; 4 12]) = 13
- **eig**: Eigenvalues and eigenvectors, for example, [v,d]=eig([1 3; 5 8]);. Therefore,
  \[
  v = \begin{bmatrix}
  -0.8675 & -0.3253 \\
  0.4974 & -0.9456 
  \end{bmatrix},
  d = \begin{bmatrix}
  -0.7202 & 0 \\
  0 & 9.7202 
  \end{bmatrix}
  \]
- **inv**: Matrix inversion, for example, >>>A=[1 3; 5 8]; B=inv(A);. Therefore,
  \[
  B = \begin{bmatrix}
  -1.1429 & 0.4286 \\
  0.7143 & -0.1429 
  \end{bmatrix},
  A*B = \begin{bmatrix}
  1.0000 & 0 \\
  0 & 1.0000 
  \end{bmatrix}
  \]

A1.6 DATA ANALYSIS

A1.6.1 Basic Operations

- **max**: Maximum element of an array, for example, >>>v=[1 3 5 2 1 7]; x=max(v); Therefore, x=7
- **mean**: Average or mean value of an array, for example, >>>v=mean([1 3 5 2 8]); Therefore v=3.8
- **median**: Median value of an array, for example, >>>a=median([1 3 5 2 8]); Therefore a=3.
- **min**: Minimum element of an array
- **sort**: Sorts elements in ascending order, for example, >>>v=sort([1 3 5 2 8]); v=[1 2 3 5 8]
- **std**: Standard deviation
- **sum**: Sum of an array elements, for example, >>>a=sum([1 3 5 2 8]); Therefore a=19

A1.6.2 Filtering and Convolution

- **conv**: Convolution and polynomial multiplication, for example, conv([1 1 1]) = [1 2 3 2 1]. If we have to multiply these two polynomials \((x^2 + 2x + 1)^*(x + 2)\), we convolve their coefficients \(\text{conv([1 2 1],[1 2])} = [1 4 5 2]\). Therefore, we write the polynomial: \(x^3 + 4x^2 + 5x + 2\)
- **conv2**: Two-dimensional convolution
- **filter**: Filter data with infinite impulse response (IIR) or finite impulse response (FIR) filter.
  Let the FIR filter be given by \(y(n) = 0.8x(n) + 0.2x(n-1) - 0.05x(n-2)\). Let the input data are \(x = [0.5 -0.2 0.6 0.1]\). Hence, \(a = [1], b = [0.5 0.2 -0.05]\), and the output is given by \(y = \text{filter}(a, b, x)\). The result is \(y=[0.6250 -0.4063 0.8906 -0.1230]\)
### A1.6.3 FOURIER TRANSFORMS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs</code></td>
<td>Absolute value and complex magnitude, for example, ( \text{abs}(4+j*3) = 5, \text{abs}([-0.2 \ 3.2]) = [0.2 \ 3.2] )</td>
</tr>
<tr>
<td><code>angle</code></td>
<td>Phase angle, e.g., ( \text{angle}(4+j*3) = 0.6435 ) in radians</td>
</tr>
<tr>
<td><code>fft</code></td>
<td>One-dimensional fast Fourier transform. For example, ( &gt;&gt;x = [1 \ 1 \ 1 \ 0]; y = \text{fft}(x); ) Hence, ( y = [3 \ 0-1.0000i \ 1.0000 \ 0+1.0000i] ). If we had written ( z = \text{fft}(x,8) ), we would have obtained ( z = [3 \ 1.7071-1.7071i \ 0-1.0000i \ 0.2929+0.2929i \ 1 \ 0.2929-0.2929i \ 0+1.0000i \ 1.7071+1.7071i] )</td>
</tr>
<tr>
<td><code>fft2</code></td>
<td>Two-dimensional fast Fourier transform</td>
</tr>
<tr>
<td><code>fftshift</code></td>
<td>Shift DC component of fast Fourier transform to the center of spectrum. For example, we write in the Command window: ( &gt;&gt;x = [1 \ 1 \ 1 \ 1 \ 1 \ 0]; y = \text{fft}(x,256); ). Then the command ( \text{plot}(\text{abs}(\text{fftshift}(y))) ) will center the spectrum. We can also write: ( \text{plot}(\text{abs}(\text{fftshift}(\text{fft}(x,256)))) )</td>
</tr>
<tr>
<td><code>ifft</code></td>
<td>Inverse one-dimensional fast Fourier transform</td>
</tr>
<tr>
<td><code>ifft2</code></td>
<td>Inverse two-dimensional fast Fourier transform</td>
</tr>
</tbody>
</table>

### A1.7 2D PLOTTING

#### A1.7.1 2D Plots

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>plot</code></td>
<td>Linear plot. If we have three vectors of equal length such as ( x ) with numbers of equal distance, ( y ) and ( z ), we can create the following simple plots: ( \text{plot}(y) ) will plot the values of ( y ) at numbers 1, 2, … in the ( x )-direction, ( \text{plot}(x,y) ) will plot the ( y ) values versus the equal-distance values of the vector ( x ) in the ( x )-direction, ( \text{plot}(x,y,x,z) ) will plot both vectors ( y ) and ( z ) on the same graph, we can plot the two vectors by writing ( &gt;&gt;\text{plot}(x,y); \text{hold on}; \text{plot}(x,z); ). If we would like the second graph to have different color, we write ( \text{plot}(x,z,'g') ) for getting green color.</td>
</tr>
<tr>
<td><code>loglog</code></td>
<td>Log–log scale plot. For example, ( \text{loglog}(y) ) will produce the plot.</td>
</tr>
<tr>
<td><code>semilogx</code></td>
<td>Semilog scale plot. The log scale will be on the ( x )-axis and the linear scale on the ( y )-axis. The plot is accomplished by writing ( \text{semilogx}(y) ).</td>
</tr>
<tr>
<td><code>semilogy</code></td>
<td>Semilog scale plot. The log scale will be on the ( y )-axis and the linear scale on the ( x )-axis. The plot is accomplished by writing ( \text{semilogy}(y) ).</td>
</tr>
<tr>
<td><code>axis</code></td>
<td>Controls axis scaling. For example, if we want the axes to have specific ranges, we write after we create a plot using the MATLAB default: ( \text{axis}([\text{min}x \ \text{max}x \ \text{min}y \ \text{max}y]) ).</td>
</tr>
<tr>
<td><code>grid</code></td>
<td>Grid lines. After we create the plot, we write ( \text{grid on} ).</td>
</tr>
<tr>
<td><code>subplot</code></td>
<td>Create axes in tiled positions. For example, when we write ( \text{subplot}(3,1,1) ), we expect 3×1 plots in one page starting from plot one. Next, we write ( \text{subplot}(3,1,2) ) and then proceed to plot the second plot and so on. If we write ( \text{subplot}(3,2,1) ), we expect 3×2 = 6 plots on the page. After we write ( \text{subplot}(3,2,1) ), we proceed to plot the first of the 3×2 matrix-format plots. For example, if we write ( \text{subplot}(3,2,2) ) and proceed to plot the figure, we create a plot at line two and the second plot.</td>
</tr>
</tbody>
</table>

(Continued)
<table>
<thead>
<tr>
<th><strong>legend</strong></th>
<th>Graph legend. For example, if we have two lines on the plot, one red and one green, and write <code>legend(&quot;one&quot;, &quot;two&quot;)</code>, then a rectangle frame will appear on the graph with a red line and the word <em>one</em> and with a green line and the word <em>two</em>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>title</strong></td>
<td>Graph title. For example, if we write <code>title(&quot;This is a graph&quot;)</code>, then the script in parentheses will appear on the top of the graph.</td>
</tr>
<tr>
<td><strong>xlabel</strong></td>
<td><em>x</em>-axis label. For example, if we write <code>xlabel(&quot;n time&quot;)</code>, the <em>n time</em> will appear under the <em>x</em>-axis.</td>
</tr>
<tr>
<td><strong>gtext</strong></td>
<td>Place the text with mouse. After we have created a plot, if we write in the Command window <code>gtext(&quot;this is the 1st graph&quot;)</code>, at the return a crosshair will appear on the graph and at the click the phrase in parentheses will appear on the graph.</td>
</tr>
</tbody>
</table>
Appendix 2
Matrix Analysis

A2.1  DEFINITIONS*

Let $A$ be an $m \times n$ matrix with elements $a_{ij}$, $i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$. A shorthand description of $A$ is

$$[A]_{ij} = a_{ij} \quad (A2.1)$$

The transpose of $A$, denoted by $A^T$, is defined as the $n \times m$ matrix with elements $a_{ji}$ or

$$[A^T]_{ij} = a_{ji} \quad (A2.2)$$

Example A2.1.1

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 9 \\ 3 & 1 \end{bmatrix}; \quad A^T = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 9 & 1 \end{bmatrix}$$

A square matrix is a matrix in which $m = n$. A square matrix is symmetric if $A^T = A$.

The rank of a matrix is the number of linearly independent rows or columns, whichever is less. The inverse of a square $n \times n$ matrix $A^{-1}$ in which

$$A^{-1}A = AA^{-1} = I \quad (A2.3)$$

where:

$I$ is the identity matrix

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad (A2.4)$$

* In this appendix, capital letters represent matrices and lowercase letters without subscripts, excluding identifiers, indicate vectors. If lowercase letters have subscripts and indicate vectors, they should be written in a boldfaced format.
A **matrix** \( A \) is singular if its inverse does not exist.

The determinant of a square \( n \times n \) matrix is denoted by \( \det(A) \), and it is computed as

\[
\det[A] = \sum_{j=1}^{n} a_{ij} C_{ij} \tag{A2.5}
\]

where:

\[
C_{ij} = (-1)^{i+j} M_{ij} \tag{A2.6}
\]

and \( M_{ij} \) is the determinant of the submatrix \( A \) obtained by deleting the \( i \)th row and \( j \)th column and is called the **minor** of \( a_{ij} \). \( C_{ij} \) is the **cofactor** of \( a_{ij} \).

**Example A2.1.2**

\[
A = \begin{bmatrix}
1 & 2 & 4 \\
4 & -3 & 9 \\
-1 & -1 & 6
\end{bmatrix}
\]

\[
\det[A] = (-1)^{1+1} \begin{vmatrix} -3 & 9 \\ -1 & 6 \end{vmatrix} + (-1)^{1+2} \begin{vmatrix} 4 & 9 \\ -1 & 6 \end{vmatrix} + (-1)^{1+3} \begin{vmatrix} 4 & -3 \\ -1 & 1 \end{vmatrix}
\]

\[
= (18 + 9) + [(-2)(24 + 9)] + [4(-4 - 3)]
\]

Any choice of \( i \) will yield the same value for \( \det(A) \).

A **quadratic form** \( Q \) is defined as

\[
Q = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j \tag{A2.7}
\]

In defining the quadratic form, it is assumed that \( a_{ji} = a_{ij} \). This entails no loss in generality since any quadratic functions may be expressed in this manner. \( Q \) may also be expressed as

\[
Q = x^T A x
\]

where:

\[
x = [x_1 \ x_2 \ x_n]^T
\]

\( A \) is a square \( n \times n \) matrix with \( a_{ji} = a_{ij} \) (symmetric matrix)
Example A2.1.3

\[
Q = \begin{bmatrix}
x_1 & x_2 \\
an_{11} & a_{12} & x_1 & x_2 \\
an_{21} & a_{22} & x_1 & x_2 \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
a_{11}x_1 + a_{21}x_2 & a_{12}x_1 + a_{22}x_2 \\
x_1 & x_2 \\
\end{bmatrix}
\]

\[
= a_{11}x_1^2 + a_{21}x_1x_2 + a_{12}x_1x_2 + a_{22}x_2^2
\]

A square \( n \times n \) matrix \( A \) is positive semidefinite if \( A \) is symmetric and

\[
x^T A x \geq 0 \tag{A2.9}
\]

for all \( x \neq 0 \). If the quadratic form is strictly positive, the matrix \( A \) is called positive definite. If a matrix is positive definite or positive semidefinite, it is automatically assumed that the matrix is symmetric.

The trace of a square matrix is the sum of the diagonal elements or

\[
\text{tr}(A) = \sum_{i=1}^{n} a_{ii} \tag{A2.10}
\]

A partitioned \( m \times n \) matrix \( A \) is a matrix that is expressed in terms of its submatrices. An example is the \( 2 \times 2 \) partitioned matrix:

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
k \times l & k \times (n-l) \\
(m-k) \times l & (m-k) \times (n-l)
\end{bmatrix}
\]

\[
\text{tr}(A) = \sum_{i=1}^{n} a_{ii} \tag{A2.11}
\]

The MATLAB functions are as follows:

\>
B=A';%B is the transpose of A
\>
B=inv(A);%B is the inverse of A
\>
a=det(A);% a is the determinant of A
\>
I=eye(n);%I is an nxn identity matrix
\>
a=trace(A);%a is the trace of A
\>
A2.2 SPECIAL MATRICES

A diagonal matrix is a square \( n \times n \) matrix with \( a_{ij} = 0 \) for \( i \neq j \). A diagonal matrix has all the elements off the principal diagonal equal to zero. Hence,
A generalization of the diagonal matrix is the square \( n \times n \) **block diagonal** matrix:

\[
A = \begin{bmatrix}
A_{11} & 0 & \cdots & 0 \\
0 & A_{22} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & A_{nn}
\end{bmatrix} = \begin{bmatrix}
A_{11}^{-1} & 0 & \cdots & 0 \\
0 & A_{22}^{-1} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & A_{nn}^{-1}
\end{bmatrix}
\]

(A2.14)

in which all \( A_{ii} \) matrices are square and the submatrices are identically zero. The submatrices may not have the same dimensions. For example, if \( k = 2 \), \( A_{11} \) may be a \( 2 \times 2 \) matrix and \( A_{22} \) might be a scalar. If all \( A_{ii} \) are nonsingular, then

\[
A^{-1} = \begin{bmatrix}
A_{11}^{-1} & 0 & \cdots & 0 \\
0 & A_{22}^{-1} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & A_{kk}^{-1}
\end{bmatrix}
\]

(A2.15)

and

\[
\det\{A\} = \prod_{i=1}^{n} \det\{A_{ii}\}
\]

(A2.16)

A square \( n \times n \) matrix is **orthogonal** if

\[
A^{-1} = A^{T}
\]

(A2.17)

**Example A2.2.1**

\[
A = \begin{bmatrix}
\frac{2}{\sqrt{5}} & 1 & 0 \\
1 & 2 & \frac{1}{\sqrt{5}} \\
\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}}
\end{bmatrix}, \\
A^{-1} = \frac{1}{\det\{A\}} \begin{bmatrix}
\frac{2}{\sqrt{5}} & 1 & 0 \\
1 & 2 & \frac{1}{\sqrt{5}} \\
\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}}
\end{bmatrix}^{T} = \begin{bmatrix}
\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\
1 & 2 \\
\frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}}
\end{bmatrix} = A^{T}
\]
A matrix is **orthogonal** if its columns (and rows) are orthonormal. Therefore, we must have

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}$$

$$a_i^T a_j = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases}$$  \hspace{1cm} (A2.18)

An idempotent matrix is a square $n \times n$ matrix that satisfies the following relations:

$$A^2 = A$$

$$A^m = A$$  \hspace{1cm} (A2.19)

**Example A2.2.2**

The **projection** matrix $A = H(H^T H)^{-1} H^T$ becomes $A^2 = H(H^T H)^{-1} H^T H$

$$(H^T H)^{-1} H^T = H(H^{-1} H^{-T} H^T H(H^T H)^{-1}) H^T$$

$$= H(H^{-1} I (H^T H)^{-1}) H^T = H(H^T H)^{-1} H^T$$

Hence, it is an idempotent matrix.

A **Toeplitz** square matrix is defined as

$$[A]_{ij} = a_{i-j}$$  \hspace{1cm} (A2.20)

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \cdots & a_{-(n-2)} \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0 \end{bmatrix}$$  \hspace{1cm} (A2.21)

Each element along the northwest-to-southeast diagonals is the same. If in addition $a_{-k} = a_k$, then $A$ is **symmetric Toeplitz**.

The MATLAB functions are as follows:

```matlab
>> A = diag(x); % creates a diagonal matrix A with its diagonal % the vector x;
>> A = toeplitz(x); % A is a symmetric Toeplitz matrix;
>> A = toeplitz(x, y); % x and y must be of the same length, the % main diagonal will be the first element of % x, the first element of y is not used;
```
A2.3 MATRIX OPERATION AND FORMULAS

Addition and Subtraction

\[
A + B = \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{bmatrix} + \begin{bmatrix}
b_{11} & b_{12} & \cdots & b_{1n} \\
b_{21} & b_{22} & \cdots & b_{2n} \\
\vdots \\
b_{m1} & b_{m2} & \cdots & b_{mn}
\end{bmatrix}
\]

(A2.22)

Both matrices must have the same dimension.

Multiplication

\[
AB \ (m \times n \times n \times k) = C \ (m \times k)
\]

\[
c_{ij} = \sum_{j=1}^{n} a_{ij}b_{ji}
\]

(A2.23)

Example A2.3.1

\[
AB = \begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
a_{31} & a_{32}
\end{bmatrix} \begin{bmatrix}
b_{11} & b_{12} \\
b_{21} & b_{22}
\end{bmatrix}
\]

(A2.24)

\[
= \begin{bmatrix}
a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\
a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\
a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22}
\end{bmatrix}; \ 3 \times 2 \times 2 \times 2 = 3 \times 2
\]

Transposition

\[
(AB)^T = B^T A^T
\]

(A2.25)

Inversion

\[
(A^T)^{-1} = (A^{-1})^T
\]

(A2.26)

\[
(AB)^{-1} = B^{-1} A^{-1}
\]

(A2.27)
\[
A^{-1} = \frac{C^T}{\text{det}[A]} \quad (A \equiv n \times n \text{ matrix}) \tag{A2.28}
\]

\[
c_{ij} = (-1)^{i+j}M_{ij} \tag{A2.29}
\]

where:

\(M_{ij}\) is the minor of \(a_{ij}\) obtained by deleting the \(i\)th row and \(j\)th column of \(A\)

**Example A2.3.2**

\[
A^T = \begin{bmatrix} 2 & 4 \\ -1 & 5 \end{bmatrix}^{-1} = \begin{bmatrix} 5 & 1 \\ -4 & 2 \end{bmatrix}^{-1} = \begin{bmatrix} 15 & -4 \\ -12 & 2 \end{bmatrix}
\]

\[
AA^{-1} = \begin{bmatrix} 2 & 4 \\ -1 & 5 \end{bmatrix} \begin{bmatrix} 15 & -4 \\ -12 & 2 \end{bmatrix} = \begin{bmatrix} 14 & -8 + 8 \\ -5 + 5 & 4 + 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I
\]

**Determinant [see (A2.5)]**

\[
A = n \times n \text{ matrix}; \quad B = n \times n \text{ matrix}
\]

\[
\text{det}[A^T] = \text{det}[A] \tag{A2.30}
\]

\[
\text{det}[cA] = c^n \text{det}[A] \tag{A2.31}
\]

\[
\text{det}[AB] = \text{det}[A]\text{det}[B] \tag{A2.32}
\]

\[
\text{det}[A^{-1}] = \frac{1}{\text{det}[A]} \tag{A2.33}
\]

**Trace [see (A2.10)]**

\[
A = n \times n \text{ matrix}; \quad B = n \times n \text{ matrix}
\]

\[
\text{tr}[AB] = \text{tr}[BA] \tag{A2.34}
\]

\[
\text{tr}[A^T B] = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}b_{ij} \tag{A2.35}
\]

\[
\text{tr}[xy^T] = y^T x; \quad x, y = \text{vectors} \tag{A2.36}
\]

**Matrix Inversion Formula**

\[
(A + BCD)^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1} \tag{A2.37}
\]

\[
(A + xx^T)^{-1} = A^{-1} - \frac{A^{-1}xx^TA^{-1}}{1 + x^TA^{-1}x}, \quad x = n \times 1 \text{ vector} \tag{A2.38}
\]
Appendix 2

Partition Matrices

Examples of $2 \times 2$ partition matrices are given as follows:

\[
AB = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix} = \begin{bmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}
\end{bmatrix}
\] (A2.39)

\[
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}^T = \begin{bmatrix}
A_{11}^T & A_{12}^T \\
A_{21}^T & A_{22}^T
\end{bmatrix}
\] (A2.40)

\[
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} = \begin{bmatrix}
k \times k & k \times (n-k) \\
(n-k) \times k & (n-k) \times (n-k)
\end{bmatrix}
\]

\[
A^{-1} = \begin{bmatrix}
(A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} & -(A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1}A_{12}A_{22}^{-1} \\
-(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}A_{21}A_{11}^{-1} & (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}
\end{bmatrix}
\] (A2.41)

\[
det\{A\} = \det\{A_{12}\}\det\{A_{11} - A_{12}A_{22}^{-1}A_{21}\} = \det\{A_{11}\}\det\{A_{22} - A_{21}A_{11}^{-1}A_{12}\} \quad (A2.42)
\]

Some important theorems are as follows:

1. A square matrix $A$ is singular (invertible) if and only if its columns (or rows) are linearly independent or, equivalently, if its $\det\{A\} \neq 0$. If this is true, $A$ is of full rank. Otherwise, it is singular.

2. A square matrix $A$ is positive definite, if and only if
   a. $A = CC^T$ (A2.43)

   where:
   - $C$ is a square matrix of the same dimension as $A$ and it is of full rank (invertible)

   b. The principal minors are all positive. (The $i$th principal minor is the determinant of the submatrix formed by deleting all rows and columns with an index greater than $i$.) If $A$ can be written as in (A2.43), but $C$ is not of full rank or the principal minors are only nonnegative, then $A$ is positive definite.

3. If $A$ is positive definite, then

   \[
   A^{-1} = (C^{-1})^T C^{-1}
   \] (A2.44)

4. If $A$ is positive definite and $B(m \times n)$ is of full rank ($m \leq n$), then $B, A, B^T$ is positive definite.

5. If $A$ is positive definite (or positive semidefinite), then the diagonal elements are positive (nonnegative).
Appendix 2

**A2.4 EIGENDECOMPOSITION OF MATRICES**

Let $\lambda$ denotes an eigenvalue of the matrix $A$ $(n \times n)$, then

$$A v = \lambda v \tag{A2.45}$$

where:
- $v$ is the eigenvector corresponding to the eigenvalue $\lambda$.

If $A$ is symmetric, then

$$A v_i = \lambda_i v_i, A v_j = \lambda_j v_j \ (\lambda_i \neq \lambda_j)$$

and

$$v_i^T A v_i = \lambda_i v_i^T v_i \ (a)$$

$$v_j^T A v_i = \lambda_j v_j^T v_i \ \text{or} \ v_j^T A v_i = \lambda_j v_j^T v_i \ (b)$$

Subtracting (a) from (b), we obtain $(\lambda_i - \lambda_j) v_i^T v_i = 0$. But $\lambda_i \neq \lambda_j$, and hence $v_i^T v_i = 0$, which implies that the eigenvectors of a symmetric matrix are orthogonal. We can proceed and normalize them producing orthonormal eigenvectors.

From (A2.45), we write

$$A = V \Lambda V^T$$

where:

$$\Lambda = \begin{bmatrix}
\lambda_1 & 0 & \cdots & 0 \\
0 & \lambda_2 & \cdots & 0 \\
\vdots & & \ddots & \vdots \\
0 & 0 & \cdots & \lambda_n
\end{bmatrix} \tag{A2.47}$$

Because $v_i$ are mutually orthogonal, $v_i^T v_j = \delta_{ij}$ makes $V$ a unitary matrix, $V^T V = I = V V^T$.

Postmultiplying (A2.46) by $V^T$, we obtain

$$A = V \Lambda V^T \sum_{i=1}^{n} \lambda_i v_i v_i^T$$

which is known as unitary decomposition of $A$. We also say that $A$ is unitary similar to the diagonal $\Lambda$, because a unitary matrix $V$ takes $A$ to diagonal form: $V^T A V = \Lambda$. 
If $\Lambda = I$, then from (A2.48) $A = VV^T = I$, and hence,

$$ I = VV^T = \sum_{i=1}^{n} v_i v_i^T $$  \hfill (A2.49)

Each of the terms in the summation is of rank 1 projection matrix:

$$ P_i^2 = v_i v_i^T v_i v_i^T = v_i v_i^T = P_i \quad (v_i^T v_i = 1) $$  \hfill (A2.50)

$$ P_i^T = v_i v_i^T = P_i $$  \hfill (A2.51)

Hence, we write [see (A2.48) and (A2.49)]

$$ A = \sum_{i=1}^{n} \lambda_i P_i $$  \hfill (A2.52)

$$ I = \sum_{i=1}^{n} P_i $$  \hfill (A2.53)

**Inverse**

Because $V$ is unitary matrix, $VV^T = I$ or $V^T = V^{-1}$ or $V = (V^T)^{-1}$, and therefore,

$$ A^{-1} = (V^T)^{-1} \Lambda^{-1} V^{-1} = V \Lambda^{-1} V^T = \sum_{i=1}^{n} \frac{1}{\lambda_i} v_i v_i^T $$  \hfill (A2.54)

**Determinant**

$$ \det(A) = \det(V) \det(\Lambda) \det(V^{-1}) = \det(\Lambda) = \prod_{i=1}^{n} \lambda_i $$  \hfill (A2.55)

### A2.5 MATRIX EXPECTATIONS

$$ E[x] = m_x $$  \hfill (A2.56)

$$ E[(x - m_x)(x - m_x)^T] = R_{xx} $$  \hfill (A2.57)

$$ E[\text{tr}(A)] = \text{tr}[E[A]] $$  \hfill (A2.58)

$$ E[Ax + b] = Am_x + b $$  \hfill (A2.59)

$$ E[xx^T] = R_{xx} + m_x m_x^T $$  \hfill (A2.60)

$$ E[xa^T x] = (R_{xx} + m_x m_x^T) a $$  \hfill (A2.61)

$$ E[(x + a)(x + a)^T] = R_{xx} + (m_x + a)(m_x + a)^T $$  \hfill (A2.62)
\[ E[x^T x] = \text{tr} \{ R_{xx} \} + m_s^x m_s = \text{tr} \{ R_{xx} + m_s m_s^T \} \]  
(A2.63)

\[ E[x^T ax^T] = a^T \{ R_{xx} + m_s m_s^T \} \]  
(A2.64)

\[ E[x^T Ax] = \text{tr} \{ AR_{xx} \} + m_s^T A m_s = \text{tr} \{ A(R_{xx} + m_s m_s^T) \} \]  
(A2.65)

### A2.6 DIFFERENTIATION OF A SCALAR FUNCTION WITH RESPECT TO A VECTOR

\[ x = [x_1 \ x_2 \ \cdots \ x_n]^T; \quad \frac{\partial}{\partial x} = \begin{bmatrix} \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} & \cdots & \frac{\partial}{\partial x_n} \end{bmatrix}^T \]

\[ \frac{\partial}{\partial x} (y^T x) = \frac{\partial}{\partial x} (x^T y) = y \]  
(A2.66)

\[ \frac{\partial}{\partial x} (x^T A) = A \]  
(A2.67)

\[ \frac{\partial}{\partial x} (x^T x) = 2x \]  
(A2.68)

\[ \frac{\partial}{\partial x} (x^T Ay) = Ay \]  
(A2.69)

\[ \frac{\partial}{\partial x} (y^T Ax) = A^T y \]  
(A2.70)

\[ \frac{\partial}{\partial x} (x^T Ax) = (A + A^T) x \]  
(A2.71)

\[ \frac{\partial}{\partial x} (x^T Axx^T) = (A + A^T) xx^T + x^T AxI \]  
(A2.72)

\[ \frac{\partial}{\partial x} (x^T Ax) = 2Ax \quad \text{if } A \text{ is symmetric} \]  
(A2.73)
This page intentionally left blank
Appendix 3

Mathematical Formulas

A3.1 TRIGONOMETRIC IDENTITIES

\[ \cos(-a) = \cos a \]
\[ \sin(-a) = -\sin a \]
\[ \cos\left( a \pm \frac{\pi}{2} \right) = \mp \sin a \]
\[ \sin\left( a \pm \frac{\pi}{2} \right) = \pm \cos a \]
\[ \cos(a \pm \pi) = -\cos a \]
\[ \sin(a \pm \pi) = -\sin a \]
\[ \cos^2 a + \sin^2 a = 1 \]
\[ \cos^2 a - \sin^2 a = \cos 2a \]
\[ \cos(a \pm b) = \cos a \cos b \mp \sin a \sin b \]
\[ \sin(a \pm b) = \sin a \cos b \pm \cos a \sin b \]
\[ \cos a \cos b = \frac{1}{2} [\cos(a - b) + \cos(a + b)] \]
\[ \sin a \sin b = \frac{1}{2} [\cos(a - b) - \cos(a + b)] \]
\[ \sin a \cos b = \frac{1}{2} [\sin(a - b) + \sin(a + b)] \]
\[ c \cos a + d \sin a = \sqrt{a^2 - b^2} \cos \left[ a - \tan^{-1}\left( \frac{d}{c} \right) \right] \]
\[ \cos^2 a = \frac{1}{2}(1 + \cos 2a) \]
\[ \cos^3 a = \frac{1}{4}(3 \cos a + \cos 3a) \]
\[
\cos^4 a = \frac{1}{8}(3 + 4 \cos 2a + \cos 4a)
\]
\[
\sin^2 a = \frac{1}{2}(1 - \cos 2a)
\]
\[
\sin^4 a = \frac{1}{8}(3 - 4 \cos 2a + \cos 4a)
\]
\[
e^{\pm ja} = \cos a \pm j \sin a
\]
\[
\cosh a = \cos ja
\]
\[
\sin a = \frac{j}{2}(e^{-ja} - e^{ja})
\]
\[
\sinh a = -j \sin ja
\]
\[
tanh a = -j \tan ja
\]

A3.2 ORTHOGONALITY

\[
\sum_{n=0}^{N-1} \cos \left( \frac{2\pi k}{N} \right) n \cos \left( \frac{2\pi l}{N} \right) n = 0 \quad 1 \leq k, l \leq N - 1, k \neq l
\]

\[
\sum_{n=0}^{N-1} \sin \left( \frac{2\pi k}{N} \right) n \sin \left( \frac{2\pi l}{N} \right) n = 0 \quad 1 \leq k, l \leq N - 1, k \neq l
\]

\[
\sum_{n=0}^{N-1} \sin \left( \frac{2\pi k}{N} \right) n \cos \left( \frac{2\pi l}{N} \right) n = \begin{cases} N/2 & k = 0, N/2 \\ N & k = 0, N/2 \\ 0 & k = 0, N / 2 \end{cases} \quad 1 \leq k, l \leq N - 1, k \neq l
\]

The above formulas are correct if all \( k, l \), and \( n \) are replaced by \( k \mod N \) and \( l \mod N \).
A3.3  SUMMATION OF TRIGONOMETRIC FORMS

\[ \sum_{n=0}^{N-1} \cos \left( \frac{2\pi k}{N} \right) n = \begin{cases} 0 & 1 \leq k \leq N - 1 \\ \frac{N}{k} & k = 0, N \end{cases} \]

\[ \sum_{n=0}^{N-1} \sin \left( \frac{2\pi k}{N} \right) n = \begin{cases} 0 & 1 \leq k \leq N - 1 \\ \frac{N}{k} & k = 0, N \end{cases} \]

where:

\( k, l, \) and \( n \) are integers

A3.4  SUMMATION FORMULAS

A3.4.1  Finite Summation Formulas

\[ \sum_{k=0}^{n} a^k = \frac{1 - a^{n+1}}{1 - a} \quad a \neq 1 \]

\[ \sum_{k=1}^{n} ka^k = \frac{[a1 - (n+1)2^n + n2a^{n+1}]}{(1-a)^2} \quad a \neq 1 \]

\[ \sum_{k=1}^{n} k^2 a^k = a[(1+a)-(n+1)^2a^n+(2n^2+2n-1)a^{n+1}-n^2a^{n+2}] \]

\[ \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \]

\[ \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \]

\[ \sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4} \]

\[ \sum_{k=0}^{2n-1} (2k+1) = n^2 \]

A3.4.2  Infinite Summation Formulas

\[ \sum_{k=0}^{\infty} a^k = \frac{1}{1 - a} \quad |a| < 1 \]
\[
\sum_{k=0}^{\infty} k a^k = \frac{a}{(1-a)^2} \quad |a| < 1
\]
\[
\sum_{k=0}^{\infty} k^2 a^k = \frac{a^2 + a}{(1-a)^3} \quad |a| < 1
\]

A3.5 SERIES EXPANSIONS

\[
e^a = 1 + a + \frac{a^2}{2!} + \frac{a^3}{3!} + \cdots
\]

\[
\ln(1 + a) = a - \frac{a^2}{2} + \frac{a^3}{3} - \frac{a^4}{4} + \cdots \quad |a| < 1
\]

\[
\sin a = a - \frac{a^3}{3!} + \frac{a^5}{5!} - \frac{a^7}{7!} + \cdots
\]

\[
\cos a = 1 - \frac{a^2}{2!} + \frac{a^4}{4!} - \frac{a^6}{6!} + \cdots
\]

\[
\tan a = a + \frac{a^3}{3} + \frac{2a^5}{15} + \frac{17a^7}{315} + \cdots \quad |a| < \frac{\pi}{2}
\]

\[
\sinh a = a + \frac{a^3}{3!} + \frac{a^5}{5!} + \frac{a^7}{7!} + \cdots
\]

\[
\cosh a = 1 + \frac{a^2}{2!} + \frac{a^4}{4!} + \frac{a^6}{6!} + \cdots
\]

\[
\tanh a = a - \frac{a^3}{3} - \frac{2a^5}{15} - \frac{17a^7}{315} - \cdots \quad |a| < \frac{\pi}{2}
\]

\[
(1 + a)^n = 1 + na + \frac{n(n-1)}{2!} a^2 + \frac{n(n-1)(n-2)}{3!} a^3 + \cdots \quad |a| < 1
\]

A3.6 LOGARITHMS

\[
\log_b N = \log_a N \log_b a = \frac{\log_a N}{\log_a b}
\]

A3.7 SOME DEFINITE INTEGRALS

\[
\int_0^\infty x^2 e^{-ax} \, dx = \frac{2}{a^3}
\]
\[ \int_0^\infty x^n e^{-ax} \, dx = \frac{n!}{a^{n+1}} \quad a > 0 \]

\[ \int_0^\infty e^{-a^2x^2} \, dx = \frac{\sqrt{\pi}}{2a} \quad a > 0 \]

\[ \int_0^\infty xe^{-a^2x^2} \, dx = \frac{1}{2a^2} \quad a > 0 \]

\[ \int_0^\infty \frac{e^{-ax}}{x} \sin(mx) \, dx = \tan^{-1}\left( \frac{m}{a} \right) \quad a > 0 \]

\[ \int_0^\infty \frac{\sin mx}{x} \, dx = \frac{\pi}{2} \]
Appendix 4
Lagrange Multiplier Method

To solve a constrained optimization problem, the Lagrange method is used. The normalized least mean-square (NLMS) recursion, for example, can be obtained as a solution to the following problem:

\[
\text{minimize } \min_w \|w(n) - w(n-1)\|^2 \text{ subject to constraint } d(n) = w^T(n+1)x(n)
\]

The first step in the solution is to write the cost function as follows:

\[
J(n) = [w(n+1) - w(n)]^T[w(n+1) - w(n)] + \lambda [d(n) - w^T(n+1)x(n)]
\]

(A4.1)

Differentiating the cost function above with respect to \(w(n+1)\), we obtain

\[
\frac{J(n)}{\partial w(n+1)} = 2[w(n+1) - w(n)] - \lambda x(n)
\]

(A4.2)

Setting the results to 0, we obtain

\[
w(n+1) = w(n) + \frac{1}{2} \lambda x(n)
\]

(A4.3)

Substituting this result into the constraint \(d(n) = w^T(n+1)x(n)\), we obtain

\[
d(n) = [w(n) + \frac{1}{2} \lambda x(n)]^T x(n) = w^T(n)x(n) + \frac{1}{2} \lambda \|x(n)\|^2
\]

(A4.4)

Since \(e(n) = d(n) - w^T(n)x(n)\), solving (A4.4) for \(\lambda\) leads to

\[
\lambda = \frac{2e(n)}{\|x(n)\|^2}
\]

(A4.5)

Substituting (A4.5) in (A4.3), we find

\[
w(n+1) = w(n) + \frac{1}{\|x(n)\|^2}e(n)x(n)
\]

(A4.6)
Finally, introducing a factor $\mu$ in (A4.6) to control the change in the weight vector, we obtain the conventional NLMS algorithm:

$$w(n+1) = w(n) + \frac{\mu}{\|x(n)\|^2} e(n) x(n)$$  \hspace{1cm} (A4.7)


This page intentionally left blank
Adaptive filters are used in many diverse applications, appearing in everything from military instruments to cellphones and home appliances. Adaptive Filtering: Fundamentals of Least Mean Squares with MATLAB® covers the core concepts of this important field, focusing on a vital part of the statistical signal processing area—the least mean square (LMS) adaptive filter. This largely self-contained text:

- Discusses random variables, stochastic processes, vectors, matrices, determinants, discrete random signals, and probability distributions
- Explains how to find the eigenvalues and eigenvectors of a matrix and the properties of the error surfaces
- Explores the Wiener filter and its practical uses, details the steepest descent method, and develops the Newton’s algorithm
- Addresses the basics of the LMS adaptive filter algorithm, considers LMS adaptive filter variants, and provides numerous examples
- Delivers a concise introduction to MATLAB®, supplying problems, computer experiments, and more than 110 functions and script files

Featuring robust appendices complete with mathematical tables and formulas, Adaptive Filtering: Fundamentals of Least Mean Squares with MATLAB® clearly describes the key principles of adaptive filtering and effectively demonstrates how to apply them to solve real-world problems.