Course Notes for EECS 451
Digital Signal Processing
Dept. of EECS
College of Engineering
The University of Michigan

Professor Andrew E. Yagle
Dept. of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122

Contents

0.1 What Is DSP? . . . 8
0.2 Why Use DSP? . . . 8
0.3 Chapter Summaries . . . 8

1 Discrete-Time Signals **10**
1.1 Notation . . . 10
1.1.1 Continuous-Time vs. Discrete-Time Notation . . . 10
1.1.2 Brackets Notation . . . 10
1.1.3 Stem Plots . . . 11
1.2 Duration of Discrete-Time Signals . . . 11
1.3 Impulses, Steps and Geometric Signals . . . 11
1.4 Discrete-Time Sinusoids . . . 12
1.4.1 Period of a Discrete-Time Sinusoid . . . 12
1.4.2 Frequency of a Discrete-Time Sinusoid . . . 13
1.5 Real Sampled Signals . . . 14

2 Discrete-Time Systems **16**
2.1 Overview of Discrete-Time Systems . . . 16
2.2 Difference Equations . . . 16
2.2.1 Nomenclature . . . 16
2.2.2 Realizations of ARMA Difference Equations . . . 17
2.2.3 Matlab's Filter Function . . . 18
2.3 Discrete-Time LTI Systems . . . 18
2.3.1 Scaling Property . . . 19
2.3.2 Additivity Property . . . 19
2.3.3 Time-Invariant Systems . . . 19
2.3.4 ARMA Difference Equations . . . 20
2.3.5 Impulse Response $h[n]$ . . . 20
2.4 Discrete-Time Convolution . . . 20
2.4.1 Convolution: Derivation . . . 21
2.4.2 Convolution: Properties . . . 21
2.4.3 Convolution: Computation . . . 21
2.4.4 Flip-and-Slide Method . . . 22
2.4.5 Linear-Combination-of-Delayed-Versions Method . . . 22
2.4.6 Using Matlab . . . 23

2.5 LTI in Series and Parallel . . . 23
2.6 Causality & BIBO Stability . . . 23
2.6.1 Definitions . . . 24
2.6.2 Causality . . . 24
2.6.3 BIBO Stability . . . 24
2.7 APPLICATION: Continuous Convolution by Discrete Convolution . . . 26

3 One-Sided z-Transforms 27
3.1 Simple z-Transforms . . . 27
3.1.1 Finite-Duration Signals . . . 27
3.1.2 Geometric Signals . . . 27
3.1.3 Causal Sinusoidal Signals . . . 28
3.2 Properties of z-Transforms . . . 28
3.2.1 Convolution . . . 28
3.2.2 Linearity . . . 29
3.2.3 Time Delay . . . 29
3.2.4 Scaling . . . 30
3.2.5 Multiplication by Time . . . 30
3.2.6 Initial Value Theorem . . . 30
3.3 Inverse z-transform . . . 31
3.3.1 Polynomials $X(z)$ . . . 31
3.3.2 Rational Functions $X(z)$ . . . 31
3.3.3 Rational Functions: Variation . . . 32
3.3.4 Multiple Poles at the Origin . . . 33
3.4 APPLICATION: Solving Difference Equations with Initial Conditions . . . 33
3.5 Transfer Functions . . . 34
3.5.1 Definition . . . 34
3.5.2 Relating Different Descriptions of LTI Systems . . . 34
3.6 APPLICATION: LCCDEs to Difference Equations . . . 36
3.6.1 Backward Difference . . . 36
3.6.2 Frequency Response of Backward Difference Approximation . . . 37
3.6.3 Sampling . . . 38

4 Two-Sided Z-Transforms 40
4.1 Definitions . . . 40
4.2 Geometric Signals . . . 40
4.2.1 Causal Geometric Signals . . . 40
4.2.2 Anticausal Geometric Signals . . . 41
4.2.3 Sums of Geometric Signals . . . 41
4.2.4 ROCs for Finite Duration, Left-sided and Right-sided Signals . . . 42
4.3 Inverse Z-transforms . . . 42
4.3.1 Multiple Inverse Z-transforms . . . 42
4.3.2 Stable Inverse Z-transform . . . 43
4.4 ROCs, Stability, Causality . . . 43
4.5 Deconvolution: Overview . . . 44
4.5.1 Deconvolution by Inverses . . . 44

4.5.2 Deconvolution by ARMA Difference Equations . . . 45
4.6 APPLICATION: Dereverberation . . . 45
4.6.1 Dereverberation: Problem . . . 45
4.6.2 Dereverberation: Solution . . . 45
4.7 APPLICATION: Deconvolving Multipath Systems . . . 46
4.7.1 Deconvolution of Minimum-Phase Multipath Systems. . . . 47
4.7.2 Deconvolving Non-Minimum-Phase Multipath Systems . . . 47

5 Frequency Response of LTI Systems 49
5.1 LTI System Response to a Complex Exponential . . . 49
5.2 LTI System Response to a Sinusoidal Input . . . 49
5.3 Computing and Using Frequency Response Functions . . . 50
5.4 Frequency Response Function of an ARMA Difference Equation . . . 51
5.5 From Frequency Response to Other Descriptions . . . 52
5.6 Effect of Poles and Zeros on Frequency Response Function Gain $|H(e^{j\omega})|$ . . . 53
5.7 APPLICATION: Notch Filters: Removing Sinusoidal Interference . . . 55
5.7.1 Derivation of Notch Filter . . . 55
5.7.2 Notch Filter Examples . . . 56
5.8 APPLICATION: Comb Filters: Removing Periodic Interference from Signals . . . 59
5.8.1 Derivation of Comb Filter . . . 60
5.8.2 Comb Filter Examples . . . 60
5.9 APPLICATION: Low-Pass Filters using Poles & Zeros . . . 63

6 Discrete-Time Fourier Series and Transforms 65
6.1 The Discrete-Time Fourier Series (DTFS) . . . 65
6.1.1 Forms of the DTFS . . . 65
6.1.2 Computing DTFS Coefficients . . . 66
6.1.3 DTFS and Frequency Response . . . 68
6.2 APPLICATION: Compute Spectra of Periodic Signals . . . 69
6.2.1 Computation of Fourier Series Coefficients from Samples . . . 69
6.2.2 APPLICATION: Interpreting Output of Matlab's "fft" . . . 70
6.3 The Discrete-Time Fourier Transform (DTFT) . . . 72
6.3.1 DTFT Basics and Properties . . . 72
6.3.2 Simple DTFT Computations . . . 72
6.3.3 Inverse DTFT . . . 73
6.3.4 Discrete Sinc Functions . . . 75

7 Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) 76
7.1 Definition of DFT . . . 76
7.2 Properties of DFT . . . 77
7.2.1 DFT of Periodic Sinusoids . . . 77
7.2.2 DFT of Convolution . . . 77
7.2.3 DFT Examples . . . 77
7.3 Fast Fourier Transform (FFT): Radix 2 . . . 78
7.3.1 Terminology . . . 78
7.3.2 Dividing a 16-Point DFT . . . 79

7.3.3 Dividing up a 2N-point DFT . . . 80
7.3.4 Dividing and Conquering . . . 80
7.4 FFT: Cooley-Tukey . . . 80
7.4.1 Coarse and Vernier Indices . . . 81
7.4.2 Procedure . . . 81
7.4.3 2-D Visualization . . . 81
7.4.4 Radix-2 Cooley-Tukey FFTs . . . 82
7.5 Real-Time vs. Batch Signal Processing . . . 82
7.6 APPLICATION: Deconvolution using DFT . . . 83
7.6.1 Deconvolution using DFT . . . 83
7.6.2 Deconvolution: Z-Transforms . . . 83
7.6.3 FFT Implementation Issues . . . 84
7.7 APPLICATIONS: Filtering Signals using the DFT . . . 84
7.7.1 Brick-Wall Low-Pass Filtering . . . 84
7.7.2 APPLICATION: Noise Filtering of Signals with Line Spectra . . . 84
7.7.3 APPLICATION: Removal of Periodic Interference . . . 87
7.8 APPLICATION: Spectra of Non-Periodic Signals using the DFT . . . 88
7.8.1 Problem Statement . . . 88
7.8.2 Sampling in Time and Frequency . . . 89
7.8.3 Exchange the Time and Frequency Domains . . . 89
7.8.4 Ranges of Indices . . . 90
7.8.5 Summary . . . 90
7.8.6 Comparison with Discretization . . . 90

8 Data Windows and Spectrograms 92
8.1 Spectral Leakage . . . 92
8.1.1 Examples of Spectral Leakage . . . 92
8.1.2 Periodic Extension of a Signal Segment . . . 93
8.1.3 Discrete Sinc Function . . . 94
8.1.4 Spectrum of Sinusoidal Segment: Leakage . . . 94
8.1.5 Spectrum of Sinusoidal Segment: No Leakage . . . 95
8.2 Data Windows . . . 95
8.2.1 Need for Data Windows . . . 96
8.2.2 Choices of Data Windows . . . 97
8.3 Spectrograms . . . 98
8.3.1 Problem Statement . . . 98
8.3.2 Spectrogram: Motivation . . . 99
8.3.3 Spectrogram: Computation . . . 99
8.3.4 Time vs. Frequency Resolution Tradeoff . . . 100
8.3.5 Chirp Signal . . . 101
8.3.6 Interpretation of Chirp . . . 101
8.3.7 Removing Interference . . . 101

9 Discrete-Time Filter Design 103
9.1 FIR Filter Design . . . 103
9.1.1 Overview . . . 103

9.1.2 Forms of Desired Frequency Response Functions . . . 103
9.1.3 Linear Phase . . . 104
9.1.4 Forms of Linear Phase Filters . . . 104
9.1.5 Design by Windowing . . . 104
9.1.6 Design by Frequency Sampling . . . 105
9.1.7 Design using Equiripple . . . 106
9.2 IIR Filter Design . . . 107
9.2.1 Overview . . . 107
9.2.2 Impulse Invariance . . . 107
9.2.3 Bilinear Transformation . . . 108
9.3 Results of Different Approaches to Filter Design . . . 109

10 Multirate Signal Processing **111**
10.1 Downsampling . . . 111
10.1.1 Downsampling: Time Domain . . . 111
10.1.2 Downsampling: Spectrum . . . 112
10.2 Upsampling . . . 112
10.2.1 Upsampling: Time Domain . . . 112
10.2.2 Upsampling: Spectrum . . . 113
10.3 Interpolation . . . 113
10.3.1 Interpolating a Signal . . . 113
10.3.2 Upsampling and Interpolation . . . 114
10.4 Multirate . . . 114
10.5 APPLICATION: Discrete to Continuous Time . . . 115
10.5.1 Motivation for Oversampling . . . 115
10.5.2 Implementing Oversampling . . . 115
10.6 APPLICATION: Audio Signal Processing . . . 116
10.6.1 Music Notation & Frequencies . . . 116
10.6.2 Musical Circle of Fifths . . . 116
10.6.3 Application of Multirate . . . 117
10.7 Filter Design . . . 117

11 Correlation **118**
11.1 Autocorrelation . . . 118
11.2 Cross-correlation . . . 119
11.3 Correlation . . . 119
11.3.1 Signal Similarity . . . 119
11.3.2 White Gaussian Noise . . . 120
11.4 APPLICATION: Period . . . 120
11.5 APPLICATION: Delay . . . 121
11.6 APPLICATION: Classification . . . 121
11.7 APPLICATIONS: Biomedical . . . 123
11.7.1 Electrocardiograms (EKG's) . . . 123
11.7.2 Heart Rate by Autocorrelation . . . 123
11.7.3 Heart Rate from Spectrum . . . 124
11.7.4 Classification . . . 125

11.7.5 Ultrasound: Time Delay . . . 125

12 Image Processing **127**
12.1 Image Processing Basics . . . 127
12.1.1 Extensions from 1-D to 2-D . . . 127
12.1.2 2-D Signals and Sampling . . . 127
12.1.3 LSI Systems and Convolution . . . 127
12.2 Fourier Transforms . . . 128
12.2.1 DSFT . . . 128
12.2.2 2-D DFT . . . 129
12.3 Wavenumber Response . . . 130
12.4 2-D Noise Filtering . . . 131
12.4.1 Implementation of Filtering . . . 131
12.4.2 Discussion of Results . . . 132
12.5 Image Deconvolution . . . 132
12.5.1 Problem Formulation . . . 132
12.5.2 Tikhonov Regularization: I . . . 133
12.5.3 Tikhonov Regularization: II . . . 133
12.6 Median Filtering . . . 135

13 Matlab: A Short Introduction **136**
13.1 Introduction . . . 136
13.1.1 Getting Started . . . 136
13.1.2 Scripts (m-Files) . . . 137
13.2 Basic Computation . . . 137
13.2.1 Basic Arithmetic . . . 137
13.2.2 Entering Vectors and Arrays . . . 137
13.2.3 Array Operations . . . 137
13.2.4 Solving Systems of Equations . . . 138
13.3 Plotting . . . 138
13.3.1 Basic Plotting . . . 138
13.3.2 Plotting Problems . . . 139
13.3.3 More Advanced Plotting . . . 139
13.3.4 Plotting Examples . . . 140
13.4 Partial Fractions . . . 141
13.4.1 Rectangular-to-Polar Complex Conversion: . . . 141
13.4.2 Polynomial Zeros: . . . 141
13.4.3 Partial Fraction Expansions: . . . 141
13.4.4 Frequency Response: . . . 142
13.4.5 Discrete-Time Commands . . . 142
13.4.6 Continuous-time Comb Filter . . . 142
13.5 Image Processing Toolbox . . . 143
13.6 Symbolic Toolbox . . . 143

Overview

This is a set of lecture notes for EECS 451, Digital Signal Processing and Analysis, Dept. of EECS, The University of Michigan, Ann Arbor, MI.

0.1 What Is DSP?

The term "Digital Signal Processing" can refer to either of the following two types of signals:

Discrete-time signals are:

- Defined only at integer-valued times;
- Obtained by sampling continuous-time signals;
- Can be processed using computer programs;
- In finite precision, but roundoff is not an issue.

Digital signals are:

- Sampled–only defined at integer-valued times;
- Quantized–only take on a finite number of values;
- Processed using DSP chips, which store and process binary bits (0 or 1);
- In coarser precision, which can cause problems.

These notes will treat only discrete-time signals; digital signals should be studied in a hardware-oriented computer engineering course.

0.2 Why Use DSP?

The advantage of DSP over analog signal processing is the ability to use computers, instead of electronic circuits, to process and store the signals. Analog filters must be implemented using capacitors and resistors, whose tolerances can be 20% or more (this can be viewed as very coarse precision). Gains must be implemented using amplifiers, which generate heat, which in turn affects electronic components. And signals are stored on tape. In the early days of synthetic aperture radar (SAR), time delays were implemented by actually connecting and disconnecting by hand long cables, which implemented time delays due to the speed of electricity through the cables!

Digital signals are processed entirely on a computer or chip. Gains, sums, and delays are all much easier to implement. We will see in these notes that discrete-time signal processing enables filtering with much sharper frequency selectivity than is possible with analog signal processing, as well as operations such as deconvolution and batch signal processing that are difficult or impossible using analog signal processing. Since it is digital, DSP also enables storage on optical media (CDs and DVDs) and computer memory (based on storage of bits).

0.3 Chapter Summaries

To stress the parallels between discrete time and continuous time, we often present discrete time results alongside (sometimes quite literally) the analogous continuous-time results. This allows what you have learned for continuous time to be applied directly to discrete time. This will also implicitly serve as a quick review of continuous-time material.

Chapter 1 introduces discrete-time signals, including bracket and stem plot notations, discrete-time impulses, step functions, geometric, and sinusoidal signals. It also introduces periods and frequencies of discrete-time sinusoids, which are more complicated topics in discrete time than in continuous time.

Chapter 2 introduces difference equations, the discrete-time counterpart to differential equations, and generalizes the results of continuous-time Linear Time-Invariant (LTI) systems, causality, Bounded-

Input-Bounded-Output (BIBO) stability, and convolution, to discrete time. All of these generalize directly, except for replacement of integrals with summations. Convolution is (fortunately!) much simpler in discrete time than in continuous time. We apply discrete-time convolution to compute continuous-time convolution by discretization of the latter.

Chapter 3 presents the one-sided z-transform, which is the discrete-time counterpart to the Laplace transform, and plays the same role for discrete-time LTI systems as the Laplace transform. Inverse z-transforms of rational functions are computed using partial fraction expansions, as in continuous time. We also discuss the use of Matlab in computing partial fraction expansions. We then present transfer functions, which relate different descriptions of LTI systems to each other; the ability to do this easily will be important in later chapters. We also give two applications: (1) solving difference equations with initial conditions (again analogous to the Laplace transform); and (2) proper discretization of differential equations into difference equations.

Chapter 4 presents the two-sided z-transform, which is the discrete-time counterpart to the two-sided Laplace transform, which is generally not covered in continuous-time signals and systems. Computing inverse two-sided z-transforms is more complicated than computing inverse one-sided z-transforms, due to *Region of Convergence*. But we can almost always find a stable, though likely non-causal, inverse two-sided z-transform. We give three applications: (1) dereverberation; (2) deconvolution of minimum-phase MA systems; and (3) deconvolution of non-minimum-phase MA systems, used for multipath.

Chapter 5 generalizes to discrete time the important concept of the response of LTI systems to complex exponential signals and sinusoids. Frequency response functions are computed from impulse responses, or read off directly from ARMA difference equations; effects of transfer function poles and zeros on it are also examined. Using the latter, we give two applications: (1) notch filters for filtering single sinusoids from a signal; and (2) comb filters for filtering periodic interference from a signal.

Chapter 6 presents the Discrete-Time Fourier Series (DTFS) and Discrete-Time Fourier Transform (DTFT). These are discrete-time counterparts to continuous-time Fourier series and transforms, respectively. DTFS allows the response of an LTI system to any periodic input to be computed as the sum of individual responses to complex exponential or sinusoidal functions appearing in the DTFS of the input. DTFT computes the frequency response of an LTI system from its impulse response. We give one application: computation of continuous-time Fourier series coefficients (i.e., line spectrum) from samples of the periodic signal. We defer other applications requiring numerical computation of the DTFT.

Chapter 7 presents the Discrete Fourier Transform (DFT), which does not have a direct continuous-time counterpart, and the Fast Fourier Transform (FFT), which is a very fast algorithm for computing the DFT. The purposes of the DFT are: (1) numerical computation of the DTFT at a discrete set of frequencies; and (2) numerical computation of the *spectrum* of a discrete-time signal. The latter topic in particular is useful in many areas of engineering. We give too many applications to list them all here.

Chapter 8 introduces data windows as a way of reducing sidelobes in spectra of sinusoids computed using the DFT. It then introduces the spectrogram as a stack of spectra computed using a window whose location is moving in time. We provide spectrograms of "The Victors" and of a chirp signal.

Chapter 9 introduces Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) discrete-time filter design techniques. FIR techniques include: windowing the ideal impulse response; frequency sampling; and equiripple. IIR techniques include: impulse invariance; and bilinear transform.

Chapter 10 introduces multirate filtering, and applies it to: discrete-to-continuous-time filter design; selective discrete-time filter design; and generation of a complete library of musical tones from a single tone.

Chapter 11 introduces autocorrelation and its application to computing the period of a signal; cross-correlation and its application to computing time delay; and correlation and its application to detection.

Chapter 12 is an introduction to image processing, which covers filtering, deconvolution, Tikhonov regularization, and median filtering. Chapter 13 is an Appendix on use of Matlab in signals and systems.

Chapter 1

Discrete-Time Signals

1.1 Notation

1.1.1 Continuous-Time vs. Discrete-Time Notation

Recall that continuous-time signals, symbolized by $x(t)$, are functions of t, where t is a member of the set of real numbers. t is usually called "time," although it need not actually be time. A continuous-time signal assigns a number $x(t_o)$ to every real number t_o. The only restriction is that two different numbers must not be assigned to the same t_o.

Discrete-time signals, symbolized by $x[n]$, are functions of n, where n is a member of the set of *integers*. n is usually called "time," although it need not actually be time. A discrete-time signal assigns a number $x[n_o]$ to every integer n_o. The only restriction is that different numbers not be assigned to the same n_o.

It is important to note that discrete-time signals are *undefined* at non-integer values of time. For example, $x[3.5]$ is undefined; it is *not* zero. Summary:

- $x(t)$ (round brackets) is a continuous-time signal, defined for all real numbers t.
- $x[n]$ (square brackets) is a discrete-time signal, defined for all integers n.

Usually, discrete-time signals come from sampling continuous-time signals. The act of sampling a continuous-time signal $x(t)$ at S samples per second to obtain a discrete-time signal $x[n]$ can be depicted by writing $x[n] = x(n/S)$ for integers n For example, a 100 Hertz sinusoid sampled at S=1000 samples per second (that is, every $\frac{1}{S}$=1 ms) becomes $\cos(0.2\pi n)$:

$$\begin{aligned} x(t) &= \cos(200\pi t). \quad \text{Set} \quad t = n/1000: \\ x[n] &= \cos(200\pi n/1000) = \cos(0.2\pi n). \end{aligned} \tag{1.1}$$

This is illustrated in the next figure.

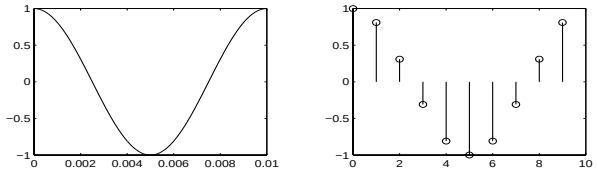


Figure 1.1: Sampling $x(t)$ at t=0.001n to obtain $x[n]$.

1.1.2 Brackets Notation

A discrete-time signal $x[n]$ can be specified by listing its values for each integer n. For example,

$$x[n] = \begin{cases} 3, & \text{n=-1} \\ 2, & \text{n=0} \\ 4, & \text{n=2} \\ 0, & \text{for other } n \end{cases} \tag{1.2}$$

This is rather cumbersome. So instead of listing values for each n, two other methods are usually used to specify discrete-time signals.

The first method is the *brackets* method, which is very effective for signals of short duration and for periodic signals. The nonzero values of $x[n]$ are listed,

separated by commas, between two curly brackets. If $x[n] = 0$ outside the interval $-N \leq n \leq N$, for some integer N, then $x[n]$ can be specified as

$$x[n] = \{x[-N], \ldots x[-1], \underline{x[0]}, x[1], \ldots x[N]\}. \quad (1.3)$$

The underline designates which value of $x[n]$ is at time $n = 0$. This immediately assigns values of time n to the other other listed values. Some books use an upward-pointing arrow $\uparrow$ below the value designated to be at time n=0. Values not listed explicitly are 0. The signal specified by (1.2) in brackets notation is

$$x[n] = \{3, \underline{2}, 0, 4\}. \quad (1.4)$$

This is much more compact than (1.2).

A *periodic* signal is depicted by specifying two of its periods and adding ellipses "..." at either end. For example, the periodic signal $y[n] = \cos(\frac{\pi}{2}n)$ is

$$y[n] = \{\ldots -1, 0, \underline{1}, 0, -1, 0, 1, 0 \ldots\} \quad (1.5)$$

This starts the two periods at $n = -2$. Any starting time can be used, but n=0 should be included in a period so the underline can be used to fix time n=0.

1.1.3 Stem Plots

The second method for specifying discrete-time signals is *stem plots*. Stem plots look like a line of dandelions, where the location and height of each dandelion specify n and $x[n]$, respectively. Stem plots for $x[n]$ in (1.2) and $y[n]$ in (1.5) are shown in the next figure.

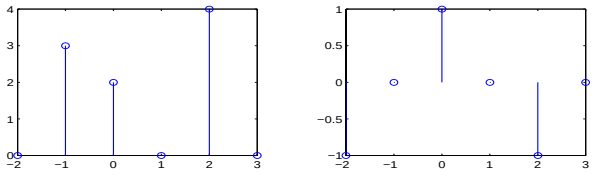


Figure 1.2: Stem plots of $x[n]$ and $y[n]$.

The left plot was generated in Matlab using

```
N=[-2:3];X=[0 3 2 0 4 0];stem(N,X)
```

Stem plots emphasize that $x[n]$ is *undefined* at non-integer times. Replacing `stem` with `plot` is incorrect, since it connects the values of $x[n]$ with straight lines, suggesting that $x[n]$ is piecewise linear between integer times, when it is undefined between integer times.

1.2 Duration of Discrete-Time Signals

We now encounter one of three situations in which discrete time is more complicated than continuous time. Consider a continuous-time signal $x(t)$ and a discrete-time signal $x[n]$. Let

- $x(t) = 0$ outside the interval $a \leq t \leq b$;
- $x[n] = 0$ outside the interval $a \leq n \leq b$.

where a, b are both integers and $x(a), x(b), x[a], x[b]$ are all nonzero values. The length or duration of $x(t)$ is clearly b–a. However, the length or duration of $x[n]$ is b–a+1, not b–a, since the number of numbers between a and b *inclusive* is b–a+1. For example, the duration of the signal specified by (1.2) is 2–(–1)+1=4, not 2–(–1)=3. Counting confirms that the duration of the signal specified by (1.2) is indeed four, not three, even though one of the values of the signal within its duration is zero. Careful counting is needed in discrete time!

Example: Duration.
Compute the duration of the signal $\{3, \underline{1}, 4, 6\}$.

Solution:
$a = -1$, $b = 2$, so b–a+1=4. Or just count to 4.

1.3 Impulses, Steps and Geometric Signals

The discrete-time *impulse* $\delta[n]$ is defined as

$$\delta[n] = \{\underline{1}\} = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \quad (1.6)$$

Unlike the continuous-time impulse, there is no requirement of zero width, infinite height, and unit area, and no singularity issue. Indeed, the very notion of continuity has no place in discrete time.

The stem plot depiction of $x[n]$ demonstrates the *sampling property of discrete-time impulses*, which is (the sampling property of continuous-time impulses is listed alongside for comparison)

$$x[n] = \sum_{i=-\infty}^{\infty} x[i]\delta[n-i] \text{ vs. } x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t-\tau)d\tau. \tag{1.7}$$

A stem plot depicts $x[n]$ as a series of impulses, each weighted by the value of $x[n]$ at the time at which the impulse is nonzero. This is precisely the sampling property of impulses.

A discrete-time *step* $u[n]$ is defined as

$$u[n] = \{\underline{1}, 1, 1 \ldots\} = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{1.8}$$

Both discrete-time and continuous-time steps are zero at negative times and one at positive times. But at time zero, the continuous-time step is undefined, while the discrete-time step is one: $u[0]=1$. The reason for defining $u[0]=1$ is so that we have (the corresponding continuous-time result is listed alongside)

$$u[n] = \sum_{i=-\infty}^{n} \delta[i] \quad \text{vs.} \quad u(t) = \int_{-\infty}^{t} \delta(\tau)\, d\tau. \tag{1.9}$$

Stem plots of an impulse and a step are shown next.

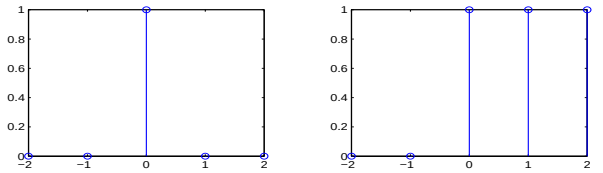


Figure 1.3: Discrete-time impulse and step signals.

A *geometric* signal is defined as $Ca^n u[n]$ for constants C, a, which may be complex numbers. Geometric signals are the discrete-time counterparts to exponential signals; a geometric signal can be regarded as a exponential signal that has been sampled at integer times. The geometric signal $(\frac{1}{2})^n u[n]$ and exponential signal $(\frac{1}{2})^t u(t)$ are plotted side-by-side in the next figure, for times between 0 and 4.

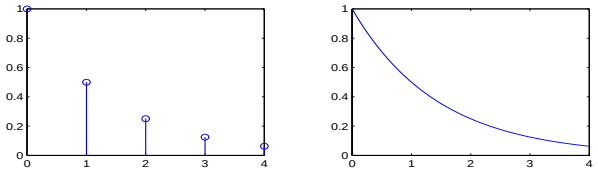


Figure 1.4: Geometric and exponential signals.

An exception to this nomenclature is signals of the form $x[n] = e^{j\omega n}$. Such signals should be called complex geometric signals, but are usually called complex exponential signals, to match continuous time use.

1.4 Discrete-Time Sinusoids

1.4.1 Period of a Discrete-Time Sinusoid

A discrete-time *sinusoid* has the form

$$x[n] = A\cos(\omega n + \theta), \quad n \in \{\text{all integers}\}. \tag{1.10}$$

This is exactly the same as in continuous time, except n has replaced t (this statement will appear frequently in this chapter). But there is a complication in discrete time sinusoids that does not arise in continuous time. This is the second of the three situations in which discrete time is more difficult than continuous time (the first was duration).

A discrete-time signal $x[n]$ is *periodic* with period N if $x[n]=x[n+N]$ for all integer times n. But in discrete time, the *period N must be an integer*, unlike the continuous-time case, since $x[n+N]$ is undefined if N is not an integer.

The period of the continuous-time sinusoid $A\cos(\omega t+\theta)$ is $T=\frac{2\pi}{\omega}$ for $\omega \neq 0$. But the period of the discrete-time sinusoid $A\cos(\omega n+\theta)$ is *not* $\frac{2\pi}{\omega}$, unless this is a nonzero integer, which it seldom is!

We assume that the frequency ω is the *fundamental frequency*, so that $0 \leq \omega \leq \pi$ (this is discussed next). Recall that $\cos(x)=\cos(x+2\pi D)$ *for all* x if and only if D is an integer. So the discrete-time sinusoid $A\cos(\omega n+\theta)$ is periodic with period N if and only if there exists an integer D such that

$$\omega(n+N)+\theta \quad = \quad (\omega n+\theta)+2\pi D$$

$$\rightarrow \quad T = \frac{2\pi}{\omega} = \frac{N}{D}. \qquad (1.11)$$

To compute the period of a discrete-time sinusoid $x[n]$ with fundamental frequency ω, proceed as follows:

1. Compute the continuous-time period $T=\frac{2\pi}{\omega}$.
2. If T is irrational, $x[n]$ is *not periodic*.
3. If T is rational, reduce T to lowest terms $T=\frac{N}{D}$ by dividing out common factors of N and D.
4. The period of $x[n]$ is then the *numerator* N.

Example: Discrete-time sinusoid period.
Compute the period of $\cos(0.3\pi n)$.

Solution:

$$T = \frac{2\pi}{\omega} = \frac{2\pi}{0.3\pi} = \frac{20}{3}. \qquad (1.12)$$

This fraction has been reduced to lowest terms, so the period is the numerator N=20.

The significance of the denominator D=3 is

$$x[n+20] = \cos(0.3\pi[n+20]) = \cos(0.3\pi n + [2\pi n]3). \qquad (1.13)$$

So D=3 periods of $\cos(0.3\pi t)$ of length $\frac{20}{3}$ each are required before it repeats for integer values n of t.

Example: Discrete-time sinusoid period.
Compute the period of $3\cos(0.56\pi n+1)$.

Solution:

$$T = \frac{2\pi}{\omega} = \frac{2\pi}{0.56\pi} = \frac{200}{56} = \frac{25}{7}. \qquad (1.14)$$

The period is the numerator N=25.

1.4.2 Frequency of a Discrete-Time Sinusoid

We now come to the third and final situation where discrete time is more difficult than continuous time. In discrete time, frequency ω is *itself* periodic with period 2π, since for any integer k we have

$$A\cos(\omega n + \theta) = A\cos([\omega + 2\pi k]n + \theta). \qquad (1.15)$$

Consider a discrete-time sine-wave generator. If ω=0, the output is constant. As ω increases, the output oscillates faster and faster until ω=π, when the output alternates between +1 and –1. As ω increases beyond π, the oscillation *slows down*, until at ω=2π the output is again constant! This phenomenon is shown in the next figure.

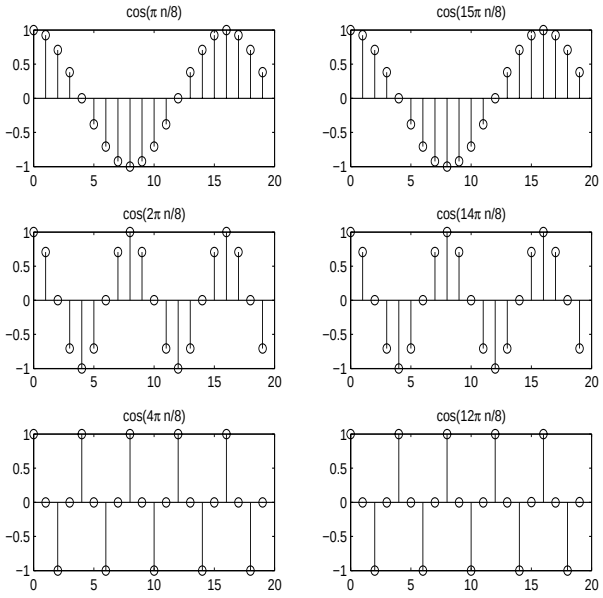


Figure 1.5: Discrete-time sinusoids of various frequencies. As the frequency increases, the oscillation speeds up then slows down.

Mathematically, this follows from

$$A\cos([\pi + \delta]n + \theta) = A\cos([\pi - \delta]n - \theta). \qquad (1.16)$$

The contrast between continuous-time frequency and discrete-time frequency is illustrated in the next figure, which depicts continuous-time sinusoids at frequencies $\omega=\frac{7\pi}{8}$ and $\omega=\frac{9\pi}{8}$, and in the next figure after, which depicts discrete-time sinusoids at the same two frequencies. The continuous-time sinusoids differ, but the discrete-time sinusoids are identical.

Aliasing in the continuous-time sampling theorem is actually a result of discrete-time frequencies ω such that $\pi < \omega < 2\pi$ being equivalent to *smaller* discrete-time frequencies $0 < \omega < \pi$ with the sign of the

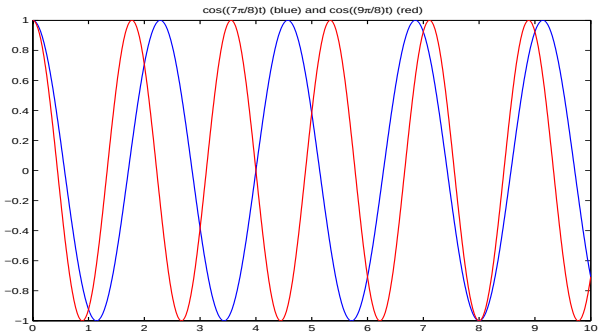


Figure 1.6: Continuous-time sinusoids at frequencies $\omega = 7\pi/8$ and $\omega = 9\pi/8$.

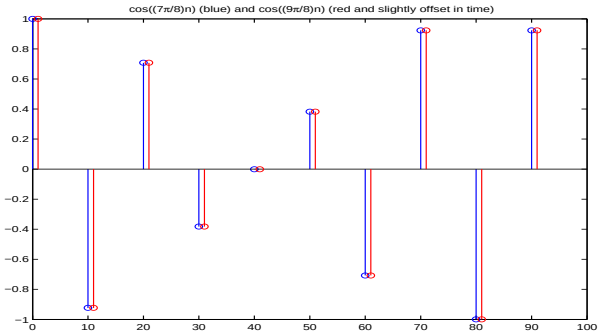


Figure 1.7: Discrete-time sinusoids at frequencies $\omega = 7\pi/8$ and $\omega = 9\pi/8$. The stem plots actually coincide, but are depicted as slightly offset in time.

phase changed. This is illustrated in the next figure, which shows why these sinusoids have identical rates of oscillation when sampled at integer times.

The *fundamental* frequency ω_o of the discrete-time sinusoid $A\cos(\omega n+\theta)$ is computed from ω using (1.15) and (1.16) to obtain $0 \leq \omega_o \leq \pi$, as the following example illustrates:

Example: Fundamental frequency.
Find the fundamental frequency of $3\cos(7.2\pi n+1)$.

Solution:
Using (1.15) and (1.16) gives

$$3\cos(7.2\pi n+1) \quad = \quad 3\cos(7.2\pi n - 3(2\pi n)+1) =$$

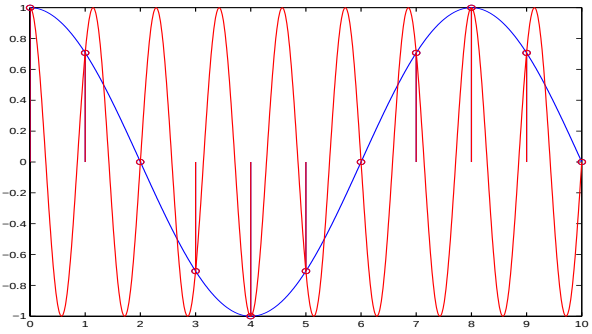


Figure 1.8: Sinusoidal signals at frequencies $\pi/4$ and $3\pi/4$ are identical at integer times.

$$3\cos(1.2\pi n+1) \quad = \quad 3\cos(0.8\pi n-1). \qquad (1.17)$$

So ω_o=0.8π. Note this is *not* 7.2π reduced mod 2π. Alternatively, we can use

$$\begin{aligned} 3\cos(7.2\pi n+1) \quad &= \quad 3\cos(7.2\pi n - 4(2\pi n)+1) = \\ 3\cos(-0.8\pi n+1) \quad &= \quad 3\cos(0.8\pi n-1). \qquad (1.18) \end{aligned}$$

since cosine is an even function.

To review, the three ways in which discrete time is more difficult than continuous time are as follows:

- If $x[n]$=0 outside the interval $a \leq n \leq b$, and $x[a] \neq 0$ and $x[b] \neq 0$, then the duration or length of $x[n]$ is b–a+1, not b–a;
- The period of $A\cos(\omega n+\theta)$ is N if $\frac{2\pi}{\omega}=\frac{N}{D}$ is reduced to lowest terms (no common factors). $A\cos(\omega n+\theta)$ is not periodic if $\frac{2\pi}{\omega}$ is irrational;
- Frequency ω is periodic with period 2π, since we have $A\cos(\omega n+\theta)$=$A\cos([\omega+2\pi k]n+\theta)$. Fundamental frequency ω_o has $0 \leq \omega_o \leq \pi$.

1.5 Real Sampled Signals

Real-world sampled signals are usually thousands of samples long. Stem plot notation would be cluttered for a signal this long, so real-world sampled signals are usually plotted to look like continuous-time signals, using continuous waveforms. It should

be kept in mind that even though these signals *look* continuous-time, they are actually discrete-time.

An example is shown in the figure below. This is the signal made by a trumpet playing musical note B, which has a fundamental frequency of 494 Hertz, and hence a period of $\frac{1}{494} \approx 2$ msec. The trumpet signal was sampled at the standard CD sampling rate of 44100 samples per sec. Several periods are shown. The actual plot has 200 points, but a stem plot with 200 stems would be very cluttered. We will encounter this signal several times in examples in these notes.

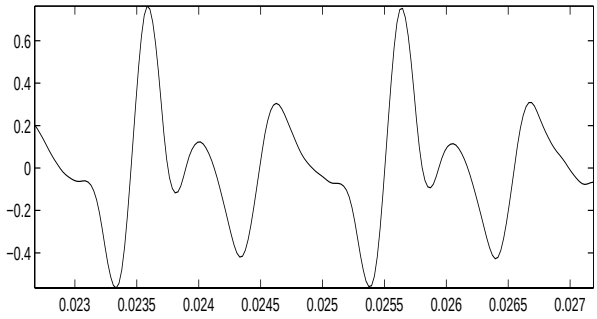


Figure 1.9: Time waveform of trumpet.

The next figure is the signal made by a train whistle. This signal is built into Matlab in `train.mat`. It was sampled at the (low) Matlab default sampling rate of 8192 samples per sec. Its duration is 12880 samples, which is $\frac{12880}{8192}$=1.5723 sec. It is hard to make much sense out of it at this time scale.

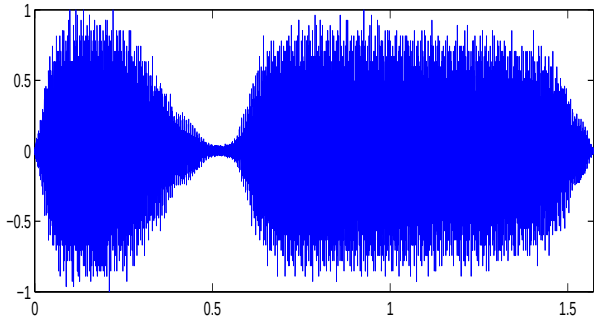


Figure 1.10: Time waveform of train whistle.

The next figure is a zoom (magnification) of the segment of the train whistle signal starting at 1.00 sec. More details can be seen, including the approximate periodicity of the signal.

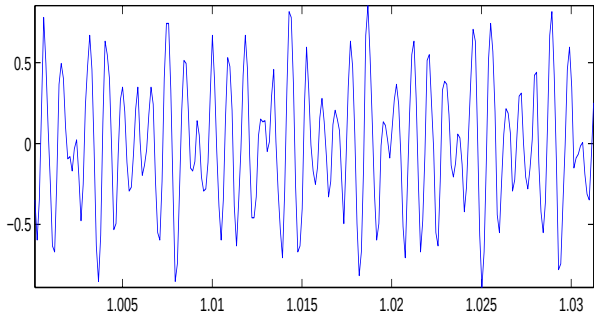


Figure 1.11: Time waveform of train whistle.

The final figure is a speech signal. It was sampled at 22050 samples per sec., which is half the CD sampling rate. Its duration is 72000 samples, which is $\frac{72000}{22050}$=3.265 sec. Note the different segments made by different phonemes. You will find out what is being said in problem set #1 of EECS 451.

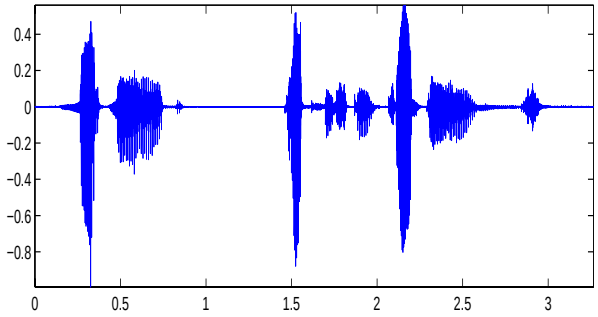


Figure 1.12: Time waveform of speech signal.

Chapter 2

Discrete-Time Systems

2.1 Overview of Discrete-Time Systems

Discrete-time systems accept as input a discrete-time signal $x[n]$ and output a discrete-time signal $y[n]$:

- Input=$x[n]$ $\rightarrow$ $\boxed{\text{SYSTEM}}$ $\rightarrow$ $y[n]$=output.

Discrete-time systems have a major advantage over continuous-time systems: they can be implemented easily on a digital computer, while continuous-time systems require physical implementation, e.g., using op-amp circuits.

Discrete-time systems are usually used to process *sampled* signals. But there are some situations in which explicitly discrete-time signals are processed using discrete-time systems. One example: $x[n]$ is the daily close, on day n, of a stock index. The L-day moving average $y[n]$ of stock index closes $x[n]$ is computed by averaging the L *most recent* values of input $x[n]$ using the *moving average* system

$$y[n] = \frac{1}{L} \sum_{i=n-L+1}^{n} x[i]. \tag{2.1}$$

L=120 (a 120-day moving average) is a common choice. This system illustrates one of the ways in which discrete time can be trickier than continuous time: Careful counting is required to ensure the lower limit in the summation is correct (e.g., try L=2).

Pollsters often use a three-day moving average (L=3) to smooth out variations in daily poll results $x[n]$ caused by statistical uncertaintly of polling data. Otherwise politicians and journalists might attach too much significance to statistical fluctuation. Later, we put this smoothing effect of most moving average systems on firmer mathematical ground.

2.2 Difference Equations

2.2.1 Nomenclature

The discrete-time counterpart to a continuous-time Linear Constant-Coefficient Differential Equation (LCCDE) is called a *difference equation*. The general forms of LCCDEs and difference equations are

$$\text{LCCDE}: \quad \sum_{i=0}^{N} a_{N-i} \frac{d^i y}{dt^i} = \sum_{j=0}^{M} b_{M-j} \frac{d^j x}{dt^j} \tag{2.2}$$

$$\text{Difference}: \quad \sum_{i=0}^{N} a_i y[n-i] = \sum_{j=0}^{M} b_j x[n-j].$$

Note that the order of the indices of the coefficients are reversed between continuous and discrete time. The reader should be cautioned that discretizing the LCCDE does not lead to the difference equation shown: the coefficients of the discretized LCCDE are different (see Chapter 3). In practice, we divide all of the coefficients of the ARMA difference equation by the coefficient of $y[n]$, so that a_0=1.

Difference equations have two parts. The left side is an *autoregression*, which means the present output is a linear combination of N previous outputs (regression). The right side is a *moving average* of the

inputs $x[n]$, since it computes a weighted average of the present input and M most recent values of the input. It is called a *moving* average since the input values being averaged change (move) with time. The *order* of the ARMA difference equation is (N, M).

The autoregressive part of the difference equation is the left side equated to the present input $x[n]$. The moving average part of the difference equation is the right side equated to the present output $y[n]$.

$$\underbrace{\sum_{i=0}^{N} a_i y[n-i]}_{\text{Autoregressive(AR)}} = \underbrace{\sum_{j=0}^{M} b_j x[n-j]}_{\text{Moving Average(MA)}} \tag{2.3}$$

This most general form of a difference equation is called an AutoRegressive Moving Average, or ARMA, difference equation. The special cases of autoregressive (AR) or moving average (MA) difference equations are then defined as

- AR: $\sum_{i=0}^{N} a_i y[n–i]=x[n]$.
- MA: $y[n]=\sum_{j=0}^{M} b_j x[n–j]$.

The 120-day moving average of stock index closes is an MA system of order 119 with coefficients $b_j=\frac{1}{120}$ for $j = 0, 1 \ldots 119$, since changing variables gives

$$y[n] = \frac{1}{120} \sum_{i=n-119}^{n} x[i] = \sum_{j=0}^{119} \frac{1}{120} x[n-j]. \tag{2.4}$$

MA systems are easier to deal with than AR or ARMA systems. Systems described by difference equations can be put into the form (2.3).

An ARMA difference equation can be implemented numerically by rewriting it as (we assume a_0=1)

$$y[n] = \sum_{j=0}^{M} b_j x[n-j] - \sum_{i=1}^{N} a_i y[n-i] \tag{2.5}$$

This allows $y[n]$, the present (at time n) output, to be computed recursively from the following:

- Present input $x[n]$;
- Past inputs $\{x[n–1],\ldots x[n–M]\}$;
- Past outputs $\{y[n–1],\ldots y[n–N]\}$.

using weighted sums, which any DSP chip can perform. It can be implemented physically as follows.

2.2.2 Realizations of ARMA Difference Equations

A realization of a system is a physical implementation of it using analog and/or digital electronic circuits. ARMA difference equations can be implemented physically using three types of elements:

- Gains, which multiply their input by a constant;
- Summers or adders, which add their inputs;
- Delays, which delay their input by one sample.

Gains and summers are implemented by op-amps, while delays are implemented by shift registers.

There are two architectures (connections of elements) that can be used: (1) Direct form I; and (2) Direct Form II. In practice, Direct Form II should always be used, since it requires only about half as many delay elements in its implementation.

We now derive the Direct Form I and II realizations for the general (2,2)-order ARMA difference equation

$$\begin{aligned} &y[n] + a_1 y[n-1] + a_2 y[n-2] \\ &= b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] \end{aligned} \tag{2.6}$$

The procedure to follow is easily generalized to larger-order ARMA difference equations.

Define the intermediate variable $z[n]$ of the ARMA difference equation as the moving average of inputs

$$z[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2]. \tag{2.7}$$

Then the ARMA difference equation is implemented by equating (2.7) to the autoregression

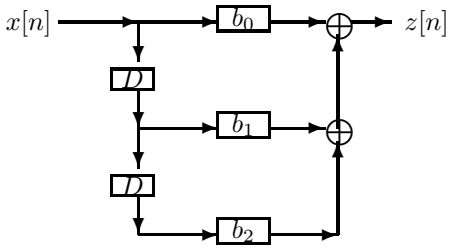
$$z[n] = y[n] + a_1 y[n-1] + a_2 y[n-2]. \tag{2.8}$$

Rewrite (2.8) so $z[n]$ is input and $y[n]$ is output:

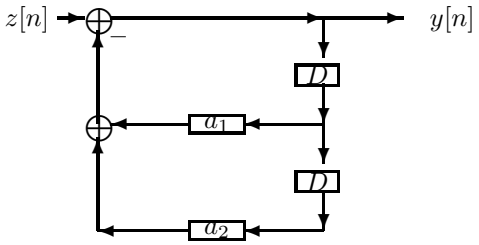
$$y[n] = z[n] - a_1 y[n-1] - a_2 y[n-2]. \tag{2.9}$$

We will realize (2.7) and (2.9) separately, and then connect them in series to obtain the realization of the complete ARMA difference equation. The order in which we connect them determines whether the realization is Direct Form I or Direct Form II.

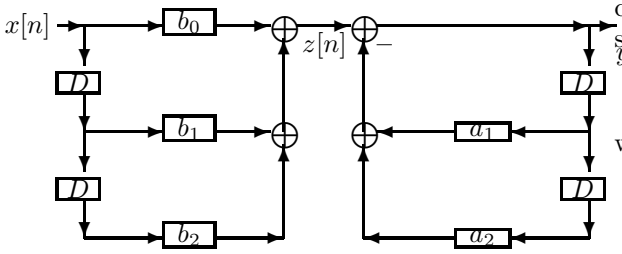
To realize (2.7), note that it expresses $z[n]$ as a linear combination of delayed versions of the input $x[n]$. Let D be a unit delay, i.e., a shift register that stores a number. Then (2.7) can be implemented as



Similarly, (2.9) can be implemented as

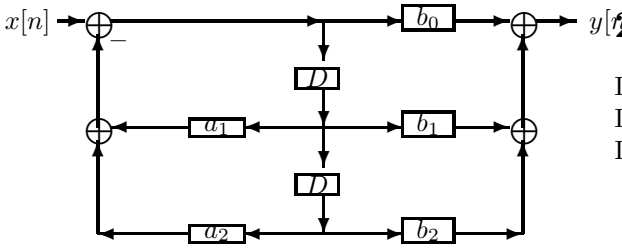


Now connect these two systems in series, so that the output $z[n]$ of (2.7) is fed into the input $z[n]$ of (2.9). This gives the complete system



This is the Direct Form I realization. Note that four delay elements are needed to implement it.

Now exchange the order of the two systems connected in series. Then the two chains of delays lying side-by-side can be replaced with a single chain of delays, so only half as many delays are required.



This is the Direct Form II realization. Note that only two delay elements are needed, instead of four.

2.2.3 Matlab's Filter Function

Matlab's `filter` function has the form

- `Y=filter(B,A,X)` where:
- `B`=$\{b_0, b_1 \ldots b_M\}$=MA coefficients vector.
- `A`=$\{a_0, a_1 \ldots a_N\}$=AR coefficients vector.
- `X`=$\{x[0], x[1], x[2] \ldots x[N–1]\}$ vector.
- `Y`=$\{y[0], y[1], y[2] \ldots y[N–1]\}$ vector.
- `N`=#of output values of $y[n]$ to be computed.

`filter` is implemented using the Direct Form II realization. Its use is illustrated with an example.

Example: Fibonacci Series

The Fibonacci series is defined by

$$\begin{aligned} y[n+2] &= y[n+1] + y[n], \quad y[0] = y[1] = 1. \\ y[n] &= \{\underline{1}, 1, 2, 3, 5, 8, 13 \ldots\} \end{aligned} \tag{2.10}$$

Each term is the sum of the two previous terms.

Replacing n with n–2 and noting that the initial conditions can be replaced with an impulsive input shows Fibonacci series can also be computed using

$$y[n] - y[n-1] - y[n-2] = x[n] = \delta[n]. \tag{2.11}$$

with initial conditions $y[-1]$=$y[-2]$=0.

Use `filter` to compute the Fibonacci series.

Solution:

```
Y=filter([1],[1 -1 -1],[1 zeros(1,6)])
```

produces the output `Y=[1 1 2 3 5 8 13]`

More terms can be generated by replacing 6 with one less than the number of outputs required.

2.3 Discrete-Time LTI Systems

In practice, we are only interested in Linear Time-Invariant (LTI) Systems. The reason for this is that LTI systems have two crucial properties:

- Assume initial conditions are all zero. Then:
- $\boxed{x[n] \rightarrow \boxed{\text{LTI}} \rightarrow y[n] = \sum_{i=-\infty}^{\infty} h[i]x[n-i]}$
  where $\delta[n] \rightarrow \boxed{\text{LTI}} \rightarrow h[n]$=impulse response.

- $\boxed{A\cos(\omega n+\phi) \rightarrow \boxed{\text{LTI}} \rightarrow AM\cos(\omega n+\phi+\theta)}$ where $Me^{j\theta}=\sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}$.

These two properties are two of the most important concepts in signals and systems, and in DSP!

A system is Linear Time-Invariant (LTI) if it is:

- Linear, which means it has the
  - Scaling property; and
  - Additivity property
- Time-Invariant

2.3.1 Scaling Property

If the response of a system to input $x[n]$ is output $y[n]$, and this implies that the response to input $cx[n]$ is output $cy[n]$ for any *constant* c, then the system has the scaling property:

- If: $x[n] \rightarrow \boxed{\text{SYSTEM}} \rightarrow y[n]$
- Then: $cx[n] \rightarrow \boxed{\text{SYSTEM}} \rightarrow cy[n]$
- For: any *constant* c.

$\boxed{\text{Scaling the input scales the output.}}$

The scaling property is also known as homogeneity.

These three systems have the scaling property:

- $y[n]$=3$x[n]$; $y[n]$=sin(n)$x[n]$; $y[n]$–$y[n$–1$]$=$x[n]$.

These three systems do not have the scaling property:

- $y[n]$=$x[n]^2$; $y[n]=\frac{x[n+1]}{x[n-1]}$; $y[n]$=$x[n]$+1.

So the second set of systems are all nonlinear.

How can one tell quickly whether a system has the scaling property? A good rule of thumb that works often (but not always) is to test whether the system has the scaling property with c=2:

$\boxed{\begin{array}{c}\text{If doubling the input does not double the output,}\\ \text{the system is not linear. But if doubling the input}\\ \text{doubles the output, then the system is likely linear.}\end{array}}$

Note $y[n]$=$|x[n]|$ passes this rule of thumb, but it does not have the scaling property (try c=–2).

2.3.2 Additivity Property

Let the responses of the system to the N inputs $\{x_i[n], i$=1 $\ldots N\}$ be the outputs $\{y_i[n], i$=1 $\ldots N\}$. If the response to $\sum_{i=1}^{N} x_i[n]$ is $\sum_{i=1}^{N} y_i[n]$, then the system has the additivity property:

- If: $x_1[n] \rightarrow \boxed{\text{LINEAR}} \rightarrow y_1[n]$
- And: $x_2[n] \rightarrow \boxed{\text{LINEAR}} \rightarrow y_2[n]$. Then:
- $(x_1[n]$+$x_2[n])$ $\rightarrow \boxed{\text{LINEAR}} \rightarrow (y_1[n]$+$y_2[n])$
- And: This extends to any number of inputs.

$\boxed{\text{Response to sum is sum of responses.}}$

Additivity holds for infinite sums and integrals.

The combination of scaling and additivity properties is also known as the superposition property.

2.3.3 Time-Invariant Systems

If the response of a system to input $x[n]$ is output $y[n]$, and this implies the response to the delayed input $x[n$–$N]$ is the delayed output $y[n$–$N]$ for any *constant* N, then the system is time-invariant:

- If: $x[n] \rightarrow \boxed{\text{SYSTEM}} \rightarrow y[n]$
- Then: $x[n$–$N] \rightarrow \boxed{\text{SYSTEM}} \rightarrow y[n$–$N]$
- For: Any *constant* N.

$\boxed{\text{Delaying the input by N delays the output by N.}}$

Physically, a system is time-invariant if it has no internal clock. If the input signal is delayed, the system has no way of knowing it, so it accepts the delayed input and delivers a correspondingly delayed output.

How can one tell if a system is time-invariant? A good rule of thumb that almost always works is:

$\boxed{\begin{array}{c}\text{If in the equation describing the system,}\\ n \text{ appears explicitly only in: } x[n];\ y[n];\\ \text{delayed versions of these; limits of sums;}\\ \text{then the system is likely time-invariant.}\end{array}}$

These two systems are time-invariant:

- $y[n]$=sin($x[n]$) and $y[n]$=$\frac{x[n+2]}{x[n-1]}$.

These two systems are not time-invariant:

- $y[n]$=$nx[n]$ and $y[n]$=$x[2n]$.

2.3.4 ARMA Difference Equations

An important fact *for constant coefficients* is that:

ARMA difference equations are LTI systems.

Let the responses to inputs $x_1[n]$ and $x_2[n]$ be outputs $y_1[n]$ and $y_2[n]$, respectively. Then, for a system described by an ARMA difference equation with constant coefficients $\{a_i\}$ and $\{b_j\}$, we have

$$\begin{aligned} \sum_{i=0}^{N} a_i y_1[n-i] &= \sum_{j=0}^{M} b_j x_1[n-j] \\ \sum_{i=0}^{N} a_i y_2[n-i] &= \sum_{j=0}^{M} b_j x_2[n-j]. \end{aligned} \quad (2.12)$$

Adding these two equations gives

$$\sum_{i=0}^{N} a_i(y_1[n-i]+y_2[n-i]) = \sum_{j=0}^{M} b_j(x_1[n-j]+x_2[n-j]) \quad (2.13)$$

so the additivity property holds. Multiplying the first equation of (2.12) by any constant c gives

$$\sum_{i=0}^{N} a_i(cy_1[n-i]) = \sum_{j=0}^{M} b_j(cx_1[n-j]) \quad (2.14)$$

so the scaling property holds.

To show time-invariance, replace n with n–N throughout the first equation of (2.12). This gives

$$\sum_{i=0}^{N} a_i y_1[(n-N)-i] = \sum_{j=0}^{M} b_j x_1[(n-N)-j] \quad (2.15)$$

so the system is time-invariant. If any of the coefficients had an n in it, this would no longer be true.

So any ARMA difference equation with constant coefficients is linear and time-invariant, so it is LTI.

2.3.5 Impulse Response $h[n]$

The impulse response of a discrete-time system is designated as $h[n]$. Unlike in continuous-time, there is no difficulty in measuring $h[n]$ for a real-world system, since discrete-time impulses exist physically.

- $\delta[n] \rightarrow$ SYSTEM $\rightarrow h[n]$

The impulse response of an LTI system can be computed from its ARMA difference equation description using the z-transform, which we present in the next two chapters. But for the special case of an MA difference equation, the impulse response $h[n]$ can be read off of its coefficients directly. Recall that the general form of an M^{th}-order MA system is

$$y[n] = \sum_{j=0}^{M} b_j x[n-j] = b_0 x[n] + \ldots + b_M x[n-M]. \quad (2.16)$$

Setting $x[n]$=$\delta[n]$ produces output $y[n]$=$h[n]$:

$$h[n] = \sum_{j=0}^{M} b_j \delta[n-j] = \{\underline{b_0}, b_1 \ldots b_M\}. \quad (2.17)$$

The impulse response $h[n]$ of the MA system $y[n]$=$\sum_{j=0}^{M} b_j x[n-j]$ is $h[n]$=$\{\underline{b_0}, b_1 \ldots b_M\}$.

We also define FIR and IIR systems as follows:

- For FIR (Finite Impulse Response) systems, $h[n]$ has finite duration.
- MA systems are FIR systems.
- For IIR (Infinite Impulse Response) systems, $h[n]$ has infinite duration.
- AR and ARMA systems are IIR systems.

2.4 Discrete-Time Convolution

Convolution is much simpler in discrete time than in continuous time, since there are no integrals to be evaluated. Convolution of two finite-duration signals is straightforward, and if either signal has infinite duration, the z-transform should be used (Chapter 3).

2.4.1 Convolution: Derivation

The response $y[n]$ of a discrete-time LTI system with impulse response $h[n]$ to input $x[n]$, if the initial conditions are all zero, is given by the *convolution* of $h[n]$ and $x[n]$, which is defined as the summation

$$y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i]. \qquad (2.18)$$

Discrete-time convolution has the same form as continuous-time convolution, except that the integral is now a summation, making its evaluation easier.

We now derive discrete-time convolution.

1. From the definition of impulse response,

$$\delta[n] \rightarrow \boxed{\text{LTI}} \rightarrow h[n].$$

2. From time-invariance of LTI systems, delaying the input $\delta[n]$ by any *integer* i delays the output $h[n]$ by the same integer i:

$$\delta[n-i] \rightarrow \boxed{\text{LTI}} \rightarrow h[n-i].$$

3. From the scaling property of LTI systems, multiplying the input $\delta[n–i]$ by any *constant* $x[i]$ multiplies the output $h[n–i]$ by the same $x[i]$:

$$x[i]\delta[n-i] \rightarrow \boxed{\text{LTI}} \rightarrow x[i]h[n-i].$$

4. From the additivity property of LTI systems, summing the input over i sums output over i:

$$\sum_{i=-\infty}^{\infty} x[i]\delta[n-i] \rightarrow \boxed{\text{LTI}} \rightarrow \sum_{i=-\infty}^{\infty} x[i]h[n-i].$$

5. From the sampling property of impulses, the input in step #4 is recognized to be just $x[n]$:

$$x[n] \rightarrow \boxed{\text{LTI}} \rightarrow \sum_{i=-\infty}^{\infty} x[i]h[n-i].$$

which is the convolution of $h[n]$ and $x[n]$.

6. $\boxed{y[n] = h[n] * x[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i].}$

2.4.2 Convolution: Properties

The properties of discrete-time convolution are identical to those of continuous-time convolution, except integrals become sums, and the duration of the convolution of two finite-duration signals is different:

1.	Commutative: $x * y = y * x$
2.	Associative: $x * (y * z) = (x * y) * z$
3.	Distributive: $(x+y) * z = (x * z) + (y * z)$
4.	Causal∗Causal=Causal
5.	Time shift: $h[n–a] * x[n–b] = y[n–a–b]$
6.	Sampling: $x[n] * \delta[n–a] = x[n–a]$
7.	Length($y[n]$)=Length($h[n]$)+Length($x[n]$)–1
8.	Area: $\sum_{-\infty}^{\infty} y[n] = (\sum_{-\infty}^{\infty} h[n])(\sum_{-\infty}^{\infty} x[n])$
9.	$x[n] * u[n] = \sum_{i=-\infty}^{n} x[i]$=summer

Why property #7 is different in discrete time:

Signal	From	To	Duration
$h[n]$	a	b	$b–a+1$
$x[n]$	c	d	$d–c+1$
$y[n]$	$a+c$	$b+d$	$(b+d)–(a+c)+1$

$$\begin{aligned} (b+d) &- (a+c)+1 = \\ (b-a+1) &- (d-c+1)-1 \end{aligned} \qquad (2.19)$$

shows that in discrete time we have

Length($y[n]$)=Length($h[n]$)+Length($x[n]$)–1

which is discrete-time convolution property #7.

2.4.3 Convolution: Computation

Convolution of finite-duration signals is much simpler in discrete time than in continuous time. Try:

$$y[n] = \{\underline{h[0]}, h[1], h[2]\} * \{\underline{x[0]}, x[1], x[2]\}.$$

Each signal has length=3, so their convolution has length=3+3–1=5, by convolution property #7. To compute the convolution, substitute this $h[n]$ and $x[n]$ in (2.18) and set n=0, 1, 2, 3, 4. Since $x[i]$=0 unless i=0,1,2 and $h[n–i]$=0 unless $i=n, n–1, n–2$, the convolution of two signals each of duration=3 is

$$y[0] = \sum_{i=0}^{0} h[0-i]x[i] \quad = \quad h[0]x[0]$$

$$
\begin{aligned}
y[1] = \sum_{i=0}^{1} h[1-i]x[i] &= h[1]x[0] + h[0]x[1] \\
y[2] = \sum_{i=0}^{2} h[2-i]x[i] &= h[2]x[0] + h[1]x[1] + h[0]x[2] \\
y[3] = \sum_{i=1}^{2} h[3-i]x[i] &= h[2]x[1] + h[1]x[2] \\
y[4] = \sum_{i=2}^{2} h[4-i]x[i] &= h[2]x[2]. \qquad (2.20)
\end{aligned}
$$

In general, the convolution of two causal signals $h[n]$ and $x[n]$ is a causal signal $y[n]$ with $y[0] = h[0]x[0]$.

Example: Convolution
Compute $\{\underline{2}, 3, 4\} * \{\underline{5}, 6, 7\}$.

Solution:

$$
\begin{aligned}
y[0] &= (2)(5) = 10. \\
y[1] &= (2)(6) + (3)(5) = 27. \\
y[2] &= (2)(7) + (3)(6) + (4)(5) = 52. \\
y[3] &= (3)(7) + (4)(6) = 45. \\
y[4] &= (4)(7) = 28. \\
y[n] &= 0 \quad \text{otherwise.} \qquad (2.21)
\end{aligned}
$$

So $\{\underline{2}, 3, 4\} * \{\underline{5}, 6, 7\} = \{\underline{10}, 27, 52, 45, 28\}$.

We can check this result using properties #8:

- (2+3+4)(5+6+7)=(10+27+52+45+28) checks.

Example: Time-Shifting.
Compute $\{1, \underline{2}\} * \{\underline{0}, 0, 3, 4\}$.

Solution:
Using property #5 with a=–1 and b=2 gives $\{1, \underline{2}\} * \{\underline{0}, 0, 3, 4\} = \{\underline{0}, 3, 10, 8\}$.

2.4.4 Flip-and-Slide Method

This convolution is illustrated using the "flip-and-slide" method in the next figure. $x[i]$ is shown in blue and $h[n–i]$ is shown in red. The five plots are for $n = 0, 1, 2, 3, 4$, going from left-to-right in each row. Summing the products in each plot gives a $y[n]$.

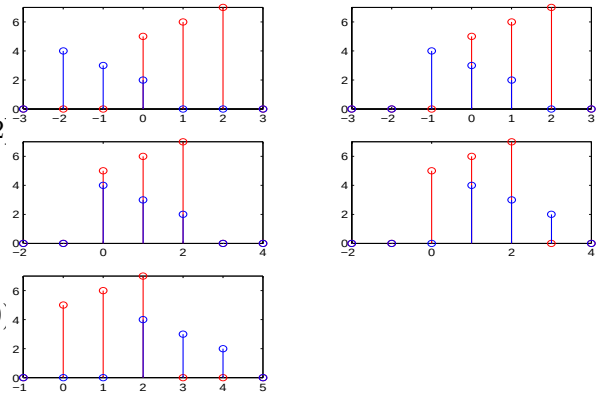


Figure 2.1: "Flip-and-slide" convolution example. Multiply and add values in each figure to get $y[n]$.

The Matlab code used to generate this figure:

```
N1=[-3:3];N2=[-2:4];N3=[-1:5];
H=[0 4 3 2 0 0 0];X=[0 0 0 5 6 7 0];
subplot(321),stem(N1,X,'r'),hold on
stem(N1,H,'b'),axis tight
H=[0 0 4 3 2 0 0];X=[0 0 0 5 6 7 0];
subplot(322),stem(N1,X,'r'),hold on
stem(N1,H,'b'),axis tight
H=[0 0 4 3 2 0 0];X=[0 0 5 6 7 0 0];
subplot(323),stem(N2,X,'r'),hold on
stem(N2,H,'b'),axis tight
H=[0 0 0 4 3 2 0];X=[0 0 5 6 7 0 0];
subplot(324),stem(N2,X,'r'),hold on
stem(N2,H,'b'),axis tight
H=[0 0 0 4 3 2 0];X=[0 5 6 7 0 0 0];
subplot(325),stem(N3,X,'r'),hold on
stem(N3,H,'b'),axis tight
```

2.4.5 Linear-Combination-of-Delayed-Versions Method

Another way to look at this discrete-time convolution is to write $h[n]$ (or $x[n]$) as a linear combination of delayed impulses and use convolution properties #3 (distributive) and #6 (sampling) to write:

$$
\begin{aligned}
h[n] * x[n] &= \{\underline{2}, 3, 4\} * \{\underline{5}, 6, 7\} \\
&= (2\delta[n] + 3\delta[n-1] + 4\delta[n-2]) * x[n]
\end{aligned}
$$

$$= 2x[n] + 3x[n-1] + 4x[n-2]. \qquad (2.22)$$

This is illustrated in the next figure. $2x[n]$ is plotted in blue; $3x[n-1]$ is plotted in green; $4x[n-2]$ is plotted in red. Add the values at each n to get $y[n]$.

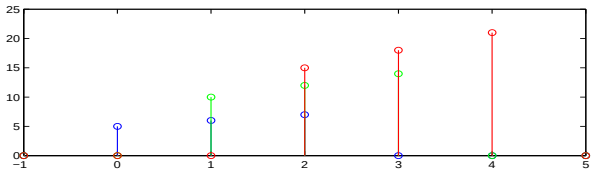


Figure 2.2: Convolution using sum of delayed $h[n]$. $2x[n]$ is plotted in blue; $3x[n-1]$ is plotted in green; $4x[n-2]$ is plotted in red. Add the values to get $y[n]$.

The Matlab code used to generate this figure:

```
subplot(211),stem(N,X,'b'),hold on
X=2*[0 0 5 6 7 0 0];stem(N,X,'g'),hold on
X=3*[0 0 0 5 6 7 0];stem(N,X,'r')
```

2.4.6 Using Matlab

This discrete-time convolution can be computed directly in Matlab using

```
conv([2 3 4],[5 6 7])=[10 27 52 45 28]
```

You may recognize the operation of convolving two finite-duration discrete-time signals as *polynomial multiplication*. This motivates the z-transform, which we consider in Chapter 3. Computation of discrete-time convolutions using the z-transform is so easy, we will not spend more time on computing it.

2.5 LTI in Series and Parallel

Denote an LTI system with impulse response $h[n]$ as:

- $\delta[n] \rightarrow \boxed{h[n]} \rightarrow h[n]$.

As in continuous time, LTI systems in series and parallel have the following overall impulse responses:

The overall impulse response of LTI systems connected in series (cascade) is the *convolution* of the individual impulse responses:

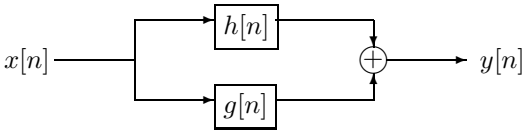
- $x[n] \rightarrow \boxed{g[n]} \rightarrow \boxed{h[n]} \rightarrow y[n]$
- Same as: $x[n] \rightarrow \boxed{h[n] * g[n]} \rightarrow y[n]$.

This follows by defining the intermediate signal $z[n]$:

- $x[n] \rightarrow \boxed{g[n]} \rightarrow z[n] \rightarrow \boxed{h[n]} \rightarrow y[n]$.
- $z[n]{=}g[n] * x[n]$ and $y[n]{=}h[n] * z[n]$.
- $y[n]{=}h[n] * (g[n] * x[n]){=}(h[n] * g[n]) * x[n]$.

since convolution is associative.

The overall impulse response of LTI systems connected in parallel is the *sum* of the impulse responses:



- Same as: $x[n] \rightarrow \boxed{g[n]{+}h[n]} \rightarrow y[n]$.

This follows by defining intermediate $z_1[n]$ and $z_2[n]$:

- $x[n] \rightarrow \boxed{g[n]} \rightarrow z_1[n]$; $\quad x[n] \rightarrow \boxed{h[n]} \rightarrow z_2[n]$
- $y[n]{=}z_1[n]{+}z_2[n]{=}g[n] * x[n]{+}h[n] * x[n]$
- $y[n]{=}(g[n]{+}h[n]) * x[n]$

since convolution is distributive.

To summarize connections of LTI systems:

Systems in series: Convolve $h[n]$'s.
Systems in parallel: Add the $h[n]$'s.

2.6 Causality & BIBO Stability

Causality is the issue of whether a system can "see into the future" and respond to a future input before it occurs. This is impossible physically, unless you are the Terminator. But non-causal systems *will* be useful in DSP, since some non-causal systems can be made causal using delays (the output is delayed).

Stability is the issue of whether the output of a system can blow up (diverge to infinity). "BIBO Stable" sounds like a Hobbit from "Lord of the Rings." But it is the most important definition of stability.

For LTI systems, there are very simple *necessary and sufficient* conditions for causality and BIBO stability, which depend on only the impulse response $h[n]$. Both conditions are derived using convolution.

2.6.1 Definitions

A signal $x[n]$ is *causal* if $x[n]$=0 for all $n < 0$.

A signal $x[n]$ is *bounded* if there exists a constant L such that $|x[n]| \leq L$ for all n. For example, $x[n]$=cos(ωn) is bounded since $|x[n]| \leq 1$.

A signal $x[n]$ is *absolutely summable* if there exists a constant M such that $\sum_{n=-\infty}^{\infty} |x[n]|$=$L$. Note

$$x[n] = \frac{(-1)^n}{n+1} u[n] = \{\underline{1}, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \ldots\} \qquad (2.23)$$

is summable but not *absolutely* summable, since:

$$\begin{aligned} 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots &= \log_e(2) \\ 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots &\rightarrow \infty \end{aligned} \qquad (2.24)$$

2.6.2 Causality

An LTI system is causal **if and only if** its impulse response $h[n]$ is causal.

To see why this is true, let $h[n]$ be causal. Then the response $y[n]$ of an LTI system to input $x[n]$ depends only on present and past inputs $\{x[i] : i \leq n\}$, since

$$y[n] = \sum_{i=0}^{\infty} h[i]x[n-i] \qquad (2.25)$$

On the other hand, let $h[-I] \neq 0$ for some $-I < 0$. Then the response $y[n]$ at present time n depends on the future input $x[n+I]$, since

$$y[n] = \sum_{i=0}^{\infty} h[i]x[n-i] + h[-I]x[n+I] \qquad (2.26)$$

so the system must be able to know a future input in advance. It is not causal (or physically realizable).

2.6.3 BIBO Stability

A system is BIBO stable if every Bounded Input (BI) results in a Bounded Output (BO):

- If: $|x[n]| \leq L$ for some L and all n
- And: $x[n] \rightarrow \boxed{\text{LTI}} \rightarrow y[n]$
- Then: $|y[n]| \leq M$ for some M and all n.

An LTI system is BIBO stable **if and only if** impulse response $h[n]$ is absolutely summable.

We will go through the proof of this result in some detail, since it is a useful exercise in learning how to prove mathematical theorems with conditionals.

Since the theorem is "if and only if," i.e., BIBO stability and absolute summability of $h[n]$ are equivalent, the proof has two parts. To prove $A \Leftrightarrow B$, we need to prove both $A \rightarrow B$ and $B \rightarrow A$. To prove $A \rightarrow B$, we *suppose* A is true and use this to prove B is true. Or we can suppose B is false and use this to prove A is false (the contrapositive of $A \rightarrow B$).

"If" Part of Proof

We suppose that $h[n]$ is absolutely summable (A). Then, for some constant L, we know that

$$\sum_{n=-\infty}^{\infty} |h[n]| = L. \qquad (2.27)$$

The goal is to prove that the system is BIBO stable (B). To prove this, we in turn suppose that the input $x[n]$ is bounded. Then, for some constant M and any constant i, we know that

$$|x[n]| \leq M \quad \text{and} \quad |x[n-i]| \leq M. \qquad (2.28)$$

The goal is now to prove that the output $y[n]$ is bounded, using the above two equations.

Having properly formulated the problem, it is now clear how to proceed. Using the triangle inequality,

$$|y[n]| = |\sum_{i=-\infty}^{\infty} h[i]x[n-i]|$$

$$\begin{aligned} &\leq \sum_{i=-\infty}^{\infty} |h[i]x[n-i]| \\ &= \sum_{i=-\infty}^{\infty} |h[i]| \cdot |x[n-i]| \\ &\leq \sum_{i=-\infty}^{\infty} |h[i]|M = LM. \end{aligned} \quad (2.29)$$

$|y[n]|$ is bounded by LM. A bounded input makes a bounded output, and the system is BIBO stable.

"Only If" Part of Proof

We could suppose BIBO stability and try to prove that $h[n]$ is absolutely summable: B→A. But it is easier to prove the (equivalent) contrapositive: Suppose that $h[n]$ is *not* absolutely summable (A is false), and find a bounded input resulting in an unbounded output (B is false). So, we need a counterexample.

Since we are supposing $\sum_{n=-\infty}^{\infty} |h[n]| \to \infty$, this would be a good candidate for our desired unbounded output. We can use for a bounded input

$$x[n] = \frac{h[-n]}{|h[-n]|} = \text{sign}(h[-n]) = \pm 1. \quad (2.30)$$

Output $y[n]$ *at time* n=0 from this input $x[n]$ is

$$\begin{aligned} y[0] &= \sum_{i=-\infty}^{\infty} h[i]x[0-i] \\ &= \sum_{i=-\infty}^{\infty} h[i]\text{sign}(h[i]) \\ &= \sum_{i=-\infty}^{\infty} |h[i]| \to \infty \end{aligned} \quad (2.31)$$

Note the time reversal in the convolution explains why we use a time-reversed $h[n]$ in $x[n]$. Recall that the time reversal in convolution comes from the time reversal in the sampling property of impulse $\delta[n]$.

Example: BIBO Stability
Prove that an LTI system with impulse response $h[n]=\frac{(-1)^n}{n+1}u[n]$ is not BIBO stable.

Solution:

$$\begin{aligned} \sum_{n=-\infty}^{\infty} h[n] &= 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots = \log_e(2) \text{ but} \\ \sum_{n=-\infty}^{\infty} |h[n]| &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots \to \infty. \end{aligned} \quad (2.32)$$

The system is *not* BIBO stable, since $h[n]$ is summable but not absolutely summable.

BIBO Stability: MA Systems

The above result shows immediately that

MA LTI systems are always BIBO stable.

To prove this, we restate a previous result:

The impulse response $h[n]$ of the MA system $y[n]=\sum_{j=0}^{M} b_j x[n–j]$ is $h[n]=\{\underline{b_0}, b_1 \ldots b_M\}$.

$\sum_{j=0}^{M} |b_j|$ is clearly finite, so the system is stable.

Example: MA system is BIBO stable
Prove $y[n]=2x[n]–3x[n–1]–4x[n–2]$ is stable.

Solution:
Since $\sum |h[n]|=2+3+4=9$ is finite, $h[n]$ is absolutely summable, and the system is BIBO stable.

BIBO Stability: Decaying Geometrics

The above result shows immediately that

An LTI system with impulse response $h[n] = Cp^n u[n]$ is BIBO stable if and only if $|p| < 1$.

To see this, recall the infinite geometric series converges if and only if its common ratio r has $|r| < 1$:

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r} \quad \text{if and only if } |r| < 1. \quad (2.33)$$

Setting r=p and checking absolute summability,

$$\sum_{n=-\infty}^{\infty} |h[n]| = |C| \sum_{n=0}^{\infty} |p|^n = \frac{|C|}{1-|p|} \quad (2.34)$$

and the system is stable **if and only if** $|p| < 1$**.**

This extends immediately to any finite linear combination of decaying geometric signals of the form

$$h[n] = \sum_{i=1}^{N} C_i p_i^n u[n] \tag{2.35}$$

By the triangle inequality, it can be seen that this also absolutely summable, and the LTI system with that impulse response is BIBO stable *if and only if all of the* p_i *have* $|p_i| < 1$. This is a fundamental result of discrete-time LTI system theory. Compare it to continuous-time LTI system theory, for which an LTI system with impulse response $h(t)$=$Ce^{at}u(t)$ is BIBO stable if and only if Real[a] < 0.

So the continuous-time stability condition of poles p_i in the left half-plane Re[s]<0 becomes the discrete-time stability condition of poles p_i inside the unit circle $|z|$=1. These conditions differ only because p_i is used in different ways in the exponential ($e^{p_i t}u(t)$) and geometric ($p_i^n u[n]$) formulae.

The following table compares discrete-time results with continuous time results for LTI systems:

Result	Discrete time	Continuous time
Notation	$x[n], n \in \{\text{integer}\}$	$x(t), t \in \{\text{reals}\}$
Signal	geometric: $p^n u[n]$	exp: $e^{-pt}u(t)$
Period of sinusoid	N in $\frac{2\pi}{\omega}=\frac{N}{D}$	$\frac{2\pi}{\omega}$ if $\omega \neq 0$
Stable	$\sum \|h[n]\|$ is finite	$\int \|h(t)\|dt$ finite
Stable	$\|p_i\| < 1$	Real[p_i] <0

2.7 APPLICATION: Continuous Convolution by Discrete Convolution

We are given continuous-time signals $h(t)$ and $x(t)$:

- $h(t)$=0 outside $0 \le t \le T_h$.
- $x(t)$=0 outside $0 \le t \le T_x$.

The goal is to compute their convolution

$$y(t) = \int_0^t h(\tau)x(t-\tau)d\tau, 0 \le t \le T_h + T_x \tag{2.36}$$

by discretizing to a discrete-time convolution. Let

- t=$n\Delta$ for some small Δ
- $h[n]$=$h(n\Delta)$ for $0 \le n \le L$=$\frac{T_h}{\Delta}$
- $x[n]$=$x(n\Delta)$ for $0 \le n \le M$=$\frac{T_x}{\Delta}$
- $y[n]$=$y(n\Delta)$ for $0 \le n \le N$=$\frac{T_h+T_x}{\Delta}$=L+M.

Using the rectangle rule of integration, the convolution discretizes to the discrete-time convolution

$$y[n] = \sum_{i=0}^{n-1} h[i]x[n-i]\Delta, \quad 0 \le n \le N \tag{2.37}$$

which can be computed using `conv(H,X)*DT`.

Example: Computing continuous-time convolution with discrete-time convolution.

Show $e^{-2t}u(t) * e^{-2t}u(t)$=$te^{-2t}u(t)$ numerically. Assume $e^{-2t}u(t)$ is negligible for $t > 5$=T_h=T_x. Use Δ=0.001. Then we have N=$\frac{5+5}{0.001}$=10000.

Solution:

We discretize $h(t)$ and $x(t)$ to 5001 points each ($0 \le t \le 5$ inclusive). Their discrete-time convolution length 5001+5001–1=10001 matches the discretization of $y(t)$ to 10001 points ($0 \le t \le 10$ inclusive). Remember that Matlab indexing starts at 1, not 0.

The Matlab code used for this example:

```
clear;N=10000;DT=(5+5)/N;T=[0:DT:5+5];
H=exp(-2*T(1:N/2+1));X=exp(-2*T(1:N/2+1));
Y=conv(H,X)*DT;%computed y(t)
Z=T.*exp(-2*T);%to compare
subplot(211),plot(T,Y,'b',T,Z,'r')
```

The results of this are shown in the next figure. The computed (blue) and actual (red) plots coincide.

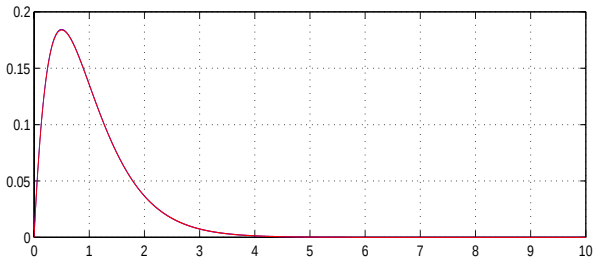


Figure 2.3: Computed (blue) and actual (red) results of numerical computation of continuous convolution.

Chapter 3

One-Sided z-Transforms

The z-transform plays the same role for discrete-time systems that the Laplace transform played for continuous-time systems. Computation of z-transforms and inverse z-transforms are performed in the same way that they are for Laplace transforms. The z-transform of a *delayed* geometric signal has the same form as the Laplace transform of an exponential signal. This is useful in computing inverse z-transforms from partial fraction expansions.

Throughout this chapter, we will use "z-transform" to denote the *one-sided* z-transform. In succeeding chapters, we will use "z-transform" to denote the *two-sided* z-transform. Similar comments apply to $X(z)$.

3.1 Simple z-Transforms

The *z-transform* of a discrete-time signal $x[n]$ is (the definition of Laplace transform is listed alongside)

$$\begin{aligned} Z(x[n]) = X(z) &= \sum_{n=0}^{\infty} x[n]z^{-n} \text{ vs.} \\ L[x(t)] = X(s) &= \int_0^{\infty} x(t)(e^s)^{-t}\,dt \end{aligned} \quad (3.1)$$

The analogy between Z and Laplace is evident.

For finite-duration signals $x[n]$, the z-transform $X(z)$ is a polynomial in z^{-1} with coefficients $x[n]$. For infinite-duration signals $x[n]$, the z-transform $X(z)$ is a power series in z^{-1} with coefficients $x[n]$. The z-transform is defined using $\frac{1}{z}$ instead of z so that poles of stable systems (defined as before as roots of the denominator of the transfer function $H(z)$) have magnitudes less than one. We discuss this below.

3.1.1 Finite-Duration Signals

For finite-duration signals, computation of the z-transform is trivial, as the following example shows.

Example: Compute $Z(\{3, 1, \underline{4}, 2, 5\})$.

Solution
$Z(\{3, 1, \underline{4}, 2, 5\})=4+2z^{-1}+5z^{-2}=\frac{4z^2+2z+5}{z^2}$.

Note that $x[n]$ *is ignored for* $n < 0$, just as the (one-sided) Laplace transform of $x(t)$ ignored $x(t)$ for $t < 0$. This may seem strange, but it is very useful for solving LCCDEs and ARMA difference equations with initial conditions. The two-sided z-transform will be computed from all $\{x[n], -\infty < n < \infty\}$.

It is customary to write z-transforms as rational functions, i.e., as a ratio of two polynomials. Also,

$$Z(\delta[n]) = 1 \text{ vs. } L[\delta(t)] = 1. \quad (3.2)$$

Z and Laplace transforms of impulses are both 1.

3.1.2 Geometric Signals

Using the sum of the infinite geometric series

$$\sum_{n=k}^{\infty} r^n = \frac{r^k}{1-r} \quad \text{if and only if} \quad |r| < 1. \quad (3.3)$$

and setting $r=\frac{a}{z}$ and $k=0$ gives

$$Z(a^n u[n]) = \sum_{n=0}^{\infty} a^n z^{-n} = \sum_{n=0}^{\infty} \left(\frac{a}{z}\right)^n$$

$$= \frac{1}{1-\left(\frac{a}{z}\right)} = \frac{z}{z-a} \quad (3.4)$$

provided $|\frac{a}{z}| < 1 \rightarrow |z| > |a|$ (otherwise the infinite geometric series does not converge). The *Region Of Convergence* (ROC) of the z-transform is thus $\{z : |z| > |a|\}$. Region of convergence is not important for the one-sided z-transform, but it is *very* important for the two-sided z-transform in the next chapter.

A minor modification of this derivation gives

$$\begin{aligned} Z(a^{n-1}u[n-1]) &= \sum_{n=1}^{\infty} a^{n-1}z^{-n} = \frac{1}{a}\sum_{n=1}^{\infty}\left(\frac{a}{z}\right)^n \\ &= \frac{(\frac{1}{a})(\frac{a}{z})}{1-\left(\frac{a}{z}\right)} = \frac{1}{z-a}. \end{aligned} \quad (3.5)$$

Compare to Laplace transform of an exponential:

$$Z(a^{n-1}u[n-1]) = \frac{1}{z-a}; L[e^{at}u(t)] = \frac{1}{s-a}. \quad (3.6)$$

The analogy is evident. But setting a=1 gives

$$Z(u[n-1]) = \frac{1}{z-1} \text{ vs. } L[u(t)] = \frac{1}{s}. \quad (3.7)$$

The Z and Laplace transforms of steps are different.

3.1.3 Causal Sinusoidal Signals

Trigonometric functions are most easily handled by writing them as sums of complex exponentials. To compute the z-transform of $A\cos(\omega n+\theta)u[n]$, write

$$\begin{aligned} A\cos(\omega n+\theta) &= \frac{A}{2}e^{j(\omega n+\theta)} + \frac{A}{2}e^{-j(\omega n+\theta)} \quad (3.8) \\ &= \frac{Ae^{j\theta}}{2}e^{j\omega n} + \frac{Ae^{-j\theta}}{2}e^{-j\omega n}. \end{aligned}$$

Setting a=$e^{j\omega}$ and a=$e^{-j\omega}$ in (3.4) and adding,

$$\begin{aligned} Z(A\cos(\omega n+\theta)u[n]) &= \quad (3.9) \\ \frac{Ae^{j\theta}}{2}Z(e^{j\omega n}u[n]) &+ \frac{Ae^{-j\theta}}{2}Z(e^{-j\omega n}u[n]). \end{aligned}$$

Using (3.4) on each term gives

$$\begin{aligned} Z(A\cos(\omega n+\theta)u[n]) &= \quad (3.10) \\ \frac{Ae^{j\theta}z/2}{z-e^{j\omega}}\left[\frac{z-e^{-j\omega}}{z-e^{-j\omega}}\right] &+ \frac{Ae^{-j\theta}z/2}{z-e^{-j\omega}}\left[\frac{z-e^{j\omega}}{z-e^{j\omega}}\right] \end{aligned}$$

provided $|z| > |e^{\pm j\omega}| = 1$. Note that we put both terms over a common denominator. From formula

$$\begin{aligned} (z-e^{j\omega})(z-e^{-j\omega}) &= z^2 - z[e^{j\omega}+e^{-j\omega}]+1 \\ &= z^2 - 2z\cos(\omega)+1 \quad (3.11) \end{aligned}$$

we obtain the final result:

$$\begin{aligned} Z(A\cos(\omega n+\theta)u[n]) &= \\ A\frac{z^2\cos(\theta) - z\cos(\theta-\omega)}{z^2-2z\cos(\omega)+1} & \end{aligned} \quad (3.12)$$

Setting θ=0 and $-\frac{\pi}{2}$ gives the two special cases

$$\begin{aligned} Z(A\cos(\omega n)u[n]) &= A\frac{z^2 - z\cos(\omega)}{z^2-2z\cos(\omega)+1} \\ Z(A\sin(\omega n)u[n]) &= A\frac{z\sin(\omega)}{z^2-2z\cos(\omega)+1} \end{aligned} \quad (3.13)$$

3.2 Properties of z-Transforms

The most important properties of the z-transform can be derived directly from its definition. We repeat it:

$$Z(x[n]) = \sum_{n=0}^{\infty} x[n]z^{-n} = x[0]+x[-1]z^{-1}+\ldots \quad (3.14)$$

3.2.1 Convolution

Recall that discrete-time convolution of two finite-duration sequences is equivalent to polynomial multiplication. More generally, the z-transform of the convolution of two causal signals $x[n]$ and $y[n]$ is

$$\begin{aligned} & Z(x[n]*y[n]) \quad (3.15) \\ &= \sum_{n=0}^{\infty} z^{-n}\sum_{i=0}^{\infty} x[i]y[n-i]u[n-i] \\ &= \sum_{i=0}^{\infty} x[i]z^{-i}\sum_{n=0}^{\infty} y[n-i]u[n-i]z^{-(n-i)} \\ &= \sum_{i=0}^{\infty} x[i]z^{-i}Y(z) = X(z)Y(z). \end{aligned}$$

The crucial step is $z^{-n} = z^{-i}z^{-(n-i)}$; this is why the z-transform maps convolutions to products. We

also used $y[n-i]u[n-i]$ to denote the causality of $y[n]$, and exchanged the order of summations, which is valid provided that the summations converge. So:

$$Z(x[n]*y[n]] = X(z)Y(z); L[x(t)*y(t)] = X(s)Y(s). \tag{3.16}$$

provided $x[n]$ and $y[n]$ are both causal signals.

Z-transforms, like Laplace & Fourier transforms, map convolutions in time to products in s or z.

Example: Compute $\{\underline{1}, 2, -3\} * u[n]$.

Solution:

We compute this convolution three different ways.
Method #1: Using convolution property #3.

$$\begin{aligned} &\{\underline{1}, 2, -3\} * u[n] \\ &= (\delta[n] + 2\delta[n-1] - 3\delta[n-2]) * u[n] \\ &= u[n] + 2u[n-1] - 3u[n-2] \\ &= \begin{cases} 0 & n \le -1 \\ 1 & n = 0 \\ 1+2=3 & n = 1 \\ 1+2-3=0 & n \ge 2 \end{cases} = \{\underline{1}, 3\}. \end{aligned} \tag{3.17}$$

Method #2: Using convolution property #8

$$\begin{aligned} &\{\underline{1}, 2, -3\} * u[n] \\ &= \sum_{i=-\infty}^{n} (\delta[i] + 2\delta[i-1] - 3\delta[i-2]) \\ &= \begin{cases} 0 & n \le -1 \\ 1 & n = 0 \\ 1+2=3 & n = 1 \\ 1+2-3=0 & n \ge 2 \end{cases} = \{\underline{1}, 3\}. \end{aligned} \tag{3.18}$$

Method #3: Using z-transforms

$$\begin{aligned} Z(\{\underline{1}, 2, -3\} * u[n]) &= \\ (1 + 2z^{-1} - 3z^{-2})\frac{z}{z-1} &= \frac{z^2+2z-3}{z(z-1)} \\ = \frac{(z+3)(z-1)}{z(z-1)} &= \frac{z+3}{z} \to \{\underline{1}, 3\}. \end{aligned} \tag{3.19}$$

This shows *why* the answer has duration=2.

3.2.2 Linearity

From the definition (3.14) we have immediately

$$Z\left(\sum_{i=1}^{N} a_i x_i[n]\right) = \sum_{i=1}^{N} a_i Z(x_i[n]). \tag{3.20}$$

So the z-transform of a linear combination of signals is the linear combination of the z-transforms of the signals. Laplace transforms also have this property.

Example: Z-transform of a linear combination of signals.
Compute the z-transform of $\{\underline{1}, 1\} + (-1)^n u[n]$.

Solution:

$$\begin{aligned} Z(\{\underline{1}, 1\} + (-1)^n u[n]) &= Z(\{\underline{1}, 1\}) + Z((-1)^n u[n]) \\ = (1 + z^{-1}) + \frac{z}{z+1} &= \frac{z+1}{z}\left[\frac{z+1}{z+1}\right] + \frac{z}{z+1}\left[\frac{z}{z}\right] \\ &= \frac{2z^2+2z+1}{z^2+z}. \end{aligned} \tag{3.21}$$

We wrote the z-transform as a rational function by putting both terms over a common denominator.

Example: Z-transform of a linear combination of signals.
Compute $Z(\{\underline{1}, 3\} + 4(2)^n u[n])$.

Solution:

$$\begin{aligned} Z(\{\underline{1}, 3\} + 4(2)^n u[n]) &= Z(\{\underline{1}, 3\}) + 4Z(2^n u[n]) \\ = (1 + 3z^{-1}) + 4\frac{z}{z-2} &= \frac{z+3}{z}\left[\frac{z-2}{z-2}\right] + \frac{4z}{z-2}\left[\frac{z}{z}\right] \\ &= \frac{5z^2+z-6}{z^2-2z}. \end{aligned} \tag{3.22}$$

We wrote the z-transform as a rational function by putting both terms over a common denominator.

3.2.3 Time Delay

Changing variables to n'=n–N (so n=n'+N) in (3.14) gives

$$Z(x[n-N]) = \sum_{n'=-N}^{\infty} x[n']z^{-(n'+N)} = z^{-N}X(z) \tag{3.23}$$

if $N > 0$ and $x[n]$ is causal, so $x[n-N]$ is also causal.

If $x[n]$ is not causal, then time delay becomes more complicated. Defining $n'' = -n'$, we obtain for $N > 0$

$$\begin{aligned} Z(x[n-N]) &= \sum_{n'=-N}^{\infty} x[n']z^{-(n'+N)} \\ &= \sum_{n'=-N}^{-1} x[n']z^{-(n'+N)} \\ &+ \sum_{n'=0}^{\infty} x[n']z^{-(n'+N)} \\ &= z^{-N}X(z) + \sum_{n''=1}^{N} x[-n'']z^{n''-N} \end{aligned} \quad (3.24)$$

The z-transform only uses the causal part of $x[n]$ ($x[n]$ for $n \geq 0$). Delaying $x[n]$ makes some of the anticausal part of $x[n]$ ($x[n]$ for $n < 0$) causal. These now-causal values must be added to the z-transform of the delayed causal part of $x[n]$.

Example: Z-transform of delayed non-causal signals.
$x[n] = \{3, \underline{2}, 4\}$. Compute the z-transform of $x[n-1]$.

Solution:

- $Z(x[n])$=$X(z)$=$2+4z^{-1}$. Ignore $x[-1] = 3$!
- $Z(x[n-1])$=$3+2z^{-1}+4z^{-2}$, *not* $z^{-1}X(z)$! Use:
- $Z(x[n-1])$=$z^{-1}X(z)+x[-1]$=$z^{-1}(2+4z^{-1})+3$ =$3+2z^{-1}+4z^{-2}$ is now the correct answer.

Compare (3.24) to the Laplace transform result

$$L\left(\frac{d^n y}{dt^n}\right) = s^n Y(s) - s^{n-1}y(0) - \ldots - \frac{d^{n-1}y}{dt^{n-1}}(0). \quad (3.25)$$

We can use (3.24) to solve difference equations with initial conditions, just as the Laplace transform can be used to solve differential equations with initial conditions. We give an example after we show how to compute inverse z-transforms.

3.2.4 Scaling

Replacing z with $\frac{z}{a}$ in (3.14) gives

$$Z[a^n x[n]] = X\left(\frac{z}{a}\right); L[e^{at}x(t)] = X\left(\frac{s}{a}\right). \quad (3.26)$$

Using this property on (3.13) shows that

$$\begin{aligned} Z(a^n \sin(\omega n)u[n]) &= \frac{\frac{z}{a}\sin(\omega)}{(\frac{z}{a})^2 - 2(\frac{z}{a})\cos(\omega) + 1} \\ &= \frac{za\sin(\omega)}{z^2 - 2za\cos(\omega) + a^2} \end{aligned} \quad (3.27)$$

3.2.5 Multiplication by Time

Differentiating (3.14) and multiplying by $-z$ gives

$$Z(nx[n]) = -z\frac{dX}{dz}; L[tx(t)] = -\frac{dX}{ds}. \quad (3.28)$$

The significance of this is in computing inverse transforms of rational functions with double poles:

$$\begin{aligned} Z((n-1)a^{n-2}u[n-1]) &= \frac{1}{(z-a)^2}. \\ L[te^{at}u(t)] &= \frac{1}{(s-a)^2}. \end{aligned} \quad (3.29)$$

In practice, double poles seldom arise except at z=0. And a double pole at z=0 is simply a delay by two.

3.2.6 Initial Value Theorem

Letting $z \to \infty$ in (3.14) gives

$$x[0] = \underset{z \to \infty}{\text{LIM}} X(z); x(0) = \underset{s \to \infty}{\text{LIM}} sX(s). \quad (3.30)$$

This shows that the inverse z-transform $x[n]$ of a strictly proper rational function has $x[0]$=0. This is useful in computing inverse z-transforms. Recall that a rational function is strictly proper if the degree of the numerator is strictly less than the degree of the denominator.

The following table compares some z-transforms with their analogous Laplace transforms:

z-transform	Laplace transform
$X(z)=\sum_{n=0}^{\infty}x[n]z^{-n}$	$X(s)=\int_0^{\infty}x(t)e^{-st}\,dt$
$Z(\delta[n])=1$	$L[\delta(t)]=1$
$Z(a^{n-1}u[n-1])=\frac{1}{z-a}$	$L[e^{at}u(t)]=\frac{1}{s-a}$
$Z(u[n-1])=\frac{1}{z-1}$	$L[u(t)]=\frac{1}{s}$

The following table compares some z-transform properties with analogous Laplace transform ones:

For causal $x[n], N>0$: $Z(x[n–N])=z^{-N}X(z)$	For causal $x(t)$: $L[\frac{d^Nx}{dt^N}]=s^NX(s)$
$Z(x[n–1])=z^{-1}X(z)+x[–1]$	$L[\frac{dx}{dt}]=sX(s)–x(0)$
$Z(a^nx[n])=X(z/a)$	$L[e^{at}x(t)]=X(s/a)$
$Z(nx[n])=-z\frac{dX}{dz}$	$L[tx(t)]=-\frac{dX}{ds}$
$x[0]=\lim_{z\to\infty}X(z)$	$x(0)=\lim_{s\to\infty}sX(s)$

3.3 Inverse z-transform

We now show how to compute the inverse z-transform $x[n]=Z^{-1}[X(z)]$. We consider two cases separately:

1. $X(z)$ is a polynomial in z^{-1}.
2. $X(z)$ is a rational function.

The inverse z-transform $x[n]$ of $X(z)$ is unique, provided $x[n]$ is constrained to be causal ($x[n]$=0 for $n<0$). If $x[n]$ is permitted to be non-causal, the inverse z-transform is not unique unless its region of convergence is also specified. This topic is covered in the next chapter. Throughout this chapter we constrain the inverse z-transform to be a causal signal, so that uniqueness of the inverse z-transform is assured.

3.3.1 Polynomials $X(z)$

To compute the inverse z-transform of a polynomial $X(z)$, simply read $x[n]$ off of the coefficients of $X(z)$:

$x[n]$ is the coefficient of z^{-n} in $X(z)$.

Three illustrative examples follow:

1. $Z^{-1}[2+4z^{-1}+5z^{-3}]=\{\underline{2},4,0,5\}$.
   Note $x[2]=0$ since the coefficient of z^{-2} is zero.
2. $Z^{-1}[\frac{7z^2+3z+6}{z^3}]=Z^{-1}[7z^{-1}+3z^{-2}+6z^{-3}]=\{\underline{0},7,3,6\}$.
3. $Z^{-1}[3z^{-4}]=3\delta[n-4]=\{\underline{0},0,0,0,3\}$.
   Zeros have been added to the bracket listing so that n=0 can be designated with an underline.

3.3.2 Rational Functions $X(z)$

If the denominator of $X(z)$ is a linear polynomial, proceed as in the following example:

Example: Compute $Z^{-1}[\frac{z-1}{z-2}]$.

Solution:

Use either of the following methods:

- $Z^{-1}[\frac{z-1}{z-2}]=Z^{-1}[\frac{z}{z-2}-\frac{1}{z-2}]=2^nu[n]–2^{n-1}u[n–1]$.
- $Z^{-1}[\frac{z-1}{z-2}]=Z^{-1}[\frac{z-2}{z-2}+\frac{1}{z-2}]=\delta[n]+2^{n-1}u[n–1]$.

Both expressions, when evaluated, are the same:

$Z^{-1}[\frac{z-1}{z-2}]=\{\underline{1},1,2,4,8,16\ldots\}$

The procedure for computing the inverse z-transform of a rational (ratio of polynomials) function is similar to the procedure for inverse Laplace transforms using partial fraction expansions. The only difference is in the final step of reading off the inverse transform from the partial fraction expansion. For this we need (3.5), which we repeat as

$$Z(a^{n-1}u[n-1])=\frac{1}{z-a} \tag{3.31}$$

and the *extremely* useful relation

$$\begin{aligned} Ap^n+A^*(p^*)^n &= 2|A||p|^n\cos(\omega n+\theta) \\ \text{for}\quad A=|A|e^{\theta} &\quad\text{and}\quad p=|p|e^{\omega}. \end{aligned} \tag{3.32}$$

The frequency ω of the discrete-time sinusoid is the *phase* $\arg[p]$ of the pole p, in *radians, not degrees.* In the continuous-time case, the frequency of the sinusoid is the *imaginary part* $\text{Imag}[p]$ of the pole p.

The complete procedure for computing the inverse z-transform of a rational function $X(z)$ is as follows:

1. **Given:** $X(z)=\frac{N(z)}{D(z)}$ where:
   - $N(z)$ is a polynomial with degree M.
   - $D(z)$ is a polynomial with degree N.

- $X(z)$ is a proper function: $M \leq N$.

2. **Compute:** The poles $\{p_i, i = 1 \ldots N\}$
   - $D(z)$=$D_0(z–p_1)\ldots(z–p_n)$ for some D_o.
   - **Assume:** The $\{p_i\}$ are distinct: $p_i \neq p_j$.
   - The $\{p_i\}$ occur in complex conjugate pairs.
3. **Compute:** The partial fraction expansion
   - $X(z) = A_0 + \sum_{i=1}^{N} \frac{A_i}{z-p_i}$.
   - $A_0 = 0$ if and only if $M < N$, that is, if and only if $X(z)$ is *strictly proper*.
4. **Read off:** The inverse z-transform as $x[n] = A_0\delta[n] + \sum_{i=1}^{N} A_i p_i^{n-1} u[n-1]$.
5. **Simplify:** $x[n]$ using the relation (3.32): $Ap^n + A^*(p^*)^n = 2|A||p|^n \cos(\omega n + \theta)$.

Example: Compute $Z^{-1}[\frac{z-3}{z^2-3z+2}]$.

Solution:
Following the above procedure, we compute:

1. Denominator $D(z)$=z^2–3z+2 has degree N=2.
2. Poles: $D(z)$=(z–1)(z–2)→ $p_1 = 1$; $p_2 = 2$.
3. $X(z) = \frac{z-3}{(z-1)(z-2)} = \frac{2}{z-1} - \frac{1}{z-2}$
   The partial fraction expansion is computed by
   `[R P]=residue([1 -3],[1 -3 2])`
   This gives residues –1 and 2 of poles 2 and 1.
   Since $X(z)$ is strictly proper ($M < N$), $x[0]$=0.
4. Read off $x[n]$=2(1)$^{n-1}u[n–1]$–1(2)$^{n-1}u[n–1]$.
   This simplifies to $x[n]$=2$u[n–1]$–2$^{n-1}u[n–1]$.

Example: Compute $Z^{-1}[\frac{z}{z^2-2z+2}]$.

Solution:
Following the above procedure, we compute:

1. Denominator=$D(z)$=z^2–2z+2 has degree N=2.
2. Poles: p_1=1+j; p_2=p_1^*=1–j.
3. $X(z) = \frac{z}{(z-(1+j))(z-(1-j))} = \frac{A}{z-p} + \frac{A^*}{z-p^*}$
   $A = \frac{1+j}{2j}$=$0.7e^{-j\pi/4}$ and p=1+j=$1.4e^{j\pi/4}$.
   The partial fraction expansion is computed by
   `[R P]=residue([1 0],[1 -2 2])`
   This gives residues $0.5 \mp j0.5$ of poles $1 \pm j$
   that is, residues $0.7e^{\mp j\pi/4}$ of poles $1.4e^{\pm j\pi/4}$.
4. $x[n] = 0.7e^{-j\pi/4}[1.4e^{j\pi/4}]^{n-1}u[n-1]$
   . $\quad + 0.7e^{j\pi/4}[1.4e^{-j\pi/4}]^{n-1}u[n-1]$.
5. $x[n]$=2(0.7)(1.4)$^{n-1}\cos(\frac{\pi}{4}(n–1)–\frac{\pi}{4})$
   $x[n]$=$(1.4)^n \sin(\frac{\pi}{4}n)u[n-1]$.

This agrees with (3.13) after modification using the scaling property with a=1.4. Note in particular that sin(ωn)$u[n–1]$=sin(ωn)$u[n]$ since sin(ω0)=0.

3.3.3 Rational Functions: Variation

A variation of the above approach will be useful in the next chapter. Instead of computing the partial fraction of $X(z)$, we compute the partial fraction expansion of $\frac{X(z)}{z}$. This creates an additional pole p_0=0. Then Steps #3 and #4 above are replaced with

$$\begin{aligned} \frac{X(z)}{z} &= \frac{A_0}{z} + \sum_{i=1}^{N} \frac{A_i}{z-p_i} \\ X(z) &= A_0 + \sum_{i=1}^{N} A_i \frac{z}{z-p_i} \\ x[n] &= A_0\delta[n] + \sum_{i=1}^{N} A_i p_i^n u[n]. \end{aligned} \quad (3.33)$$

where we use (3.4) instead of (3.5) to compute $x[n]$. This gives the same answer as the previous approach, but the answer is often in a different form. We now illustrate this by redoing the above two examples.

Example: Compute $Z^{-1}[\frac{z-3}{z^2-3z+2}]$.

Solution:
Following the second procedure, we compute:

1. Denominator $D(z)$=z^2–3z+2 has degree N=2.
2. Poles: $D(z)$=(z–1)(z–2)→ $p_1 = 1$; $p_2 = 2$.
3. $\frac{X(z)}{z} = \frac{z-3}{z(z-1)(z-2)} = \frac{-3/2}{z} + \frac{2}{z-1} - \frac{1/2}{z-2}$
   The partial fraction expansion is computed by
   `[R P]=residue([1 -3],[1 -3 2 0])`
   This gives residues $[-\frac{1}{2}, 2, -\frac{3}{2}]$ of poles [2,1,0].

4. Read off $x[n]=-\frac{3}{2}\delta[n]+2u[n]-\frac{1}{2}(2)^n u[n]$,

same as the previous answer $2u[n-1]-2^{n-1}u[n-1]$. The purpose of the $-\frac{3}{2}\delta[n]$ term is to make $x[0]=0$, as it must be since $X(z)$ is strictly proper.

Example: Compute $Z^{-1}[\frac{z}{z^2-2z+2}]$.

Solution:
Following the second procedure, we compute:

1. Denominator$=D(z)=z^2-2z+2$ has degree $N=2$.
2. Poles: $p_1=1+j$; $p_2=p_1^*=1-j$.
3. $\frac{X(z)}{z} = \frac{1}{(z-(1+j))(z-(1-j))} = \frac{A}{z-p} + \frac{A^*}{z-p^*}$
   $A = \frac{1}{2j}=0.5e^{-j\pi/2}$ and $p=1+j=1.4e^{j\pi/4}$.
   The partial fraction expansion is computed by
   `[R P]=residue([1],[1 -2 2])`
   This gives residues $\mp j/2$ of poles $1 \pm j$
   that is, residues $\frac{1}{2}e^{\mp j\pi/2}$ of poles $1.4e^{\pm j\pi/4}$.
4. $x[n] = 0.5e^{-j\pi/2}[1.4e^{j\pi/4}]^n u[n]$
   $\quad + 0.5e^{j\pi/2}[1.4e^{-j\pi/4}]^n u[n]$.
5. $x[n]=(1.4)^n\cos(\frac{\pi}{4}n-\frac{\pi}{2})=(1.4)^n\sin(\frac{\pi}{4}n)u[n]$,

same as the previous answer. Here, the extra pole $p_0=0$ cancels the zero at $z=0$, which simplifies things.

3.3.4 Multiple Poles at the Origin

Multiple poles do not arise in practice, due to round-off error. Instead of two poles at 2, there are poles at, say, 1.99 and 2.01 (in the real world, numbers are not "nice.") However, there is one case that can arise: multiple poles at the *origin* ($z=0$, the origin of the complex plane). We will see the need for this later in this chapter. Fortunately, this can be handled as in the following example.

Example: Multiple poles at the origin.
Compute the inverse z-transform of $X(z)=\frac{z^3+2z^2+3z+4}{z^2(z-1)}$, which has two poles at $z=0$.

Multiply $X(z)$ by $\frac{z}{z}$ to get

$$\begin{aligned} X(z) &= \frac{z^3+2z^2+3z+4}{z^2(z-1)}\left[\frac{z}{z}\right] \\ &= \underbrace{\frac{z^3+2z^2+3z+4}{z^3}}_{X_1(z)}\underbrace{\frac{z}{z-1}}_{X_2(z)} \\ &= X_1(z)X_2(z). \end{aligned} \tag{3.34}$$

Now we take the unusual step of using the inverse z-transform to convert a multiplication into a convolution! Using the sum-of-impulses procedure for convolution, the inverse z-transform of this is

$$\begin{aligned} x[n] &= x_1[n] * x_2[n] \\ &= \{\underline{1},2,3,4\} * u[n] \\ &= u[n] + 2u[n-1] + 3u[n-2] + 4u[n-3] \\ &= \begin{cases} 0 & n<0 \\ 1 & n=0 \\ 3 & n=1 \\ 6 & n=2 \\ 10 & n\geq 3 \end{cases} = \{\underline{1},3,6,10,10\ldots\} \end{aligned} \tag{3.35, 3.36}$$

Note that since $X(z)$ is proper but not strictly proper, $x[0] \neq 0$. The initial value theorem gives $x[0]=1$, which agrees with the above expression for $x[n]$.

Changing z^2 to, say, z^5 in the denominator of $X(z)$ would delay the above answer by 5–2=3, since multiplication by $z^{-D}, D>0$ delays the inverse z-transform by D, provided it was causal before the delay. Generalization of this example to more complicated examples is straightforward.

3.4 APPLICATION: Solving Difference Equations with Initial Conditions

The time-delay property of z-transforms can be used to solve difference equations with initial conditions, just as the one-sided Laplace transform can be used to solve LCCDEs with initial conditions. The following example illustrates the procedure.

Example: Solving difference equations with initial conditions.
Solve $y[n]-3y[n-1]+2y[n-2]=\{\underline{0},0,12\}$ with the two initial conditions $y[-1]=2$ and $y[-2]=3$.

Solution:

Taking the z-transform of the difference equation and using the time-delay property (3.24) gives

$$\begin{aligned} Y(z) &- 3(z^{-1}Y(z) + y[-1]) \\ &+ 2(z^{-2}Y(z) + y[-1]z^{-1} + y[-2]) = 12z^{-2}. \end{aligned} \tag{3.37}$$

Inserting the initial conditions $y[-1]=2$ and $y[-2]=3$ and solving for $Y(z)$ gives

$$Y(z) = \frac{12-4z}{z^2-3z+2} = -4\frac{z-3}{z^2-3z+2}. \tag{3.38}$$

The solution is the result of the inverse z-transform computed in a previous example multiplied by –4. So

$$\begin{aligned} y[n] &= -4(2u[n-1] - 2^{n-1}u[n-1]) \\ &= 2^{n+1}u[n-1] - 8u[n-1]. \end{aligned} \tag{3.39}$$

We check this result by recursively computing $y[n]$:

- y[0]=3y[–1]–2y[–2]=3(2)–2(3)=0.
- y[1]=3y[0]–2y[–1]=3(0)–2(2)=–4.
- y[2]=3y[1]–2y[0]+12=3(–4)–2(0)+12=0.
- y[3]=3y[2]–2y[1]=3(0)–2(–4)=8.
- y[4]=3y[3]–2y[2]=3(8)–2(0)=24, etc.

The recursively-computed values of $y[n]$ agree with (3.39) for $n \geq 0$. Note that (3.39) does *not* produce the initial conditions, since (3.39), derived using the one-sided z-transform, is only valid for $n \geq 0$.

3.5 Transfer Functions

3.5.1 Definition

The transfer (or system) function $H(z)$ is defined as the z-transform of the impulse response:

$$H(z) = Z(h[n]) \text{ vs. } H(s) = L[h(t)]. \tag{3.40}$$

In practice, $H(z)$ is usually a rational function

$$H(z) = \frac{\sum_{i=0}^{M} b_i z^i}{\sum_{i=0}^{N} a_i z^i} \tag{3.41}$$

The *poles* and *zeros* are the roots of the denominator and numerator polynomials, respectively, set to zero:

$$\begin{aligned} \sum_{i=0}^{N} a_i z^i = 0 \quad &\rightarrow \quad \text{Poles}\{p_1 \ldots p_N\} \\ \sum_{i=0}^{M} b_i z^i = 0 \quad &\rightarrow \quad \text{Zeros}\{z_1 \ldots z_M\} \end{aligned} \tag{3.42}$$

Furthermore, let $y[n]$ be the response to a input $x[n]$ that is not a pure sinusoid (note that a causal sinusoid $A\cos(\omega n)u[n]$ is not a pure sinusoid $A\cos(\omega n)$). Then, **if all initial conditions are zero**, we have

$$\begin{aligned} y[n] &= h[n] * x[n] \\ Y(z) &= H(z)X(z) \\ H(z) &= Y(z)/X(z). \end{aligned} \tag{3.43}$$

So the transfer function $H(z)$ can be computed from almost any input-output pair as $H(z)=Y(z)/X(z)$:

- Input=$x[n]$ $\rightarrow$ $\boxed{\text{LTI}}$ $\rightarrow$ $y[n]$=output.

Now consider the general ARMA difference equation

$$\sum_{i=0}^{N} a_i y[n-i] = \sum_{j=0}^{M} b_j x[n-j]. \tag{3.44}$$

Taking z-transforms, and setting all initial conditions to zero, gives

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{j=0}^{M} b_j z^{-j}}{\sum_{i=0}^{N} a_i z^{-i}} \tag{3.45}$$

which is a rational function of z^{-1}, not z. But multiplying by $\frac{z^N}{z^N}$ converts this into a rational function of z if $N \geq M$. Reversing this procedure is tricky; using cross-multiplication, as in the examples below, is more straightforward, and is recommended.

3.5.2 Relating Different Descriptions of LTI Systems

Transfer functions play a central role in relating different descriptions of LTI systems to each other, just

as they did in continuous time. To go from any description to any other one, first compute the transfer function from the given description, and then use the transfer function to compute the desired description:

Input $x[n] \rightarrow$ output $y[n]$		Input $\delta[n] \rightarrow$ output $h[n]$
$H(z)=\frac{Z(y[n])}{Z(x[n])}$		$H(z)=Z(h[n])$
	$H(z)$ $=\frac{N(z)}{D(z)}$	
$Z^{-1}\{D(z)Y(z)$ $=N(z)X(z)\}$		$N(z_i)=0$ $D(p_i)=0$
Difference Equation		Zeros $\{z_i\}$ Poles $\{p_i\}$

This is best illustrated with a series of examples.

Example: Input-output pair to other LTI system descriptions

The response of an LTI system to given input $x[n] = (-2)^n u[n]$ is $y[n] = \frac{2}{3}(-2)^n u[n] + \frac{1}{3}u[n]$. Find: (1) transfer function $H(z)$; (2) poles and zeros; (3) impulse response $h[n]$; (4) difference equation.

Solution:

First, we compute the transfer function $H(z)$ using

$$y[n] = h[n] * x[n] \rightarrow H(z) = Y(z)/X(z). \quad (3.46)$$

The z-transform of the input $x[n]$ is

$$X(z) = Z((-2)^n u[n]) = \frac{z}{z+2}. \quad (3.47)$$

The z-transform of the output $y[n]$ is

$$\begin{aligned} Y(z) &= Z[(2/3)(-2)^n u[n] + (1/3)u[n]] \quad (3.48)\\ &= \frac{2}{3}\frac{z}{z+2}\left[\frac{z-1}{z-1}\right] + \frac{1}{3}\frac{z}{z-1}\left[\frac{z+2}{z+2}\right] \\ &= \frac{z^2}{(z+2)(z-1)}. \end{aligned}$$

where we have put both terms over a common denominator in order to express $Y(z)$ as a ratio of two polynomials. The transfer function $H(z)$ is then

$$H(z) = \frac{Y(z)}{X(z)} = \frac{z^2}{(z+2)(z-1)} / \frac{z}{z+2} = \frac{z}{z-1}. \quad (3.49)$$

Then, having obtained the transfer function $H(z)$, we can now compute everything else. The zeros are $\{0\}$ and the poles are $\{1\}$. The impulse response is

$$h[n] = Z^{-1}\left[\frac{z}{z-1}\right] = u[n]. \quad (3.50)$$

To compute the difference equation, write

$$H(z) = \frac{Y(z)}{X(z)} = \frac{z}{z-1}\left[\frac{z^{-1}}{z^{-1}}\right] = \frac{1}{1-z^{-1}} \quad (3.51)$$

We multiplied $H(z)$ by $\frac{z^{-D}}{z^{-D}}$, where D is the largest exponent in $H(z)$, to convert $H(z)$ into a polynomial in z^{-1} instead of z. Next, cross-multiply to get

$$Y(z)(1-z^{-1}) = Y(z) - z^{-1}Y(z) = X(z) \quad (3.52)$$

and read off the inverse z-transform as

$$y[n] - y[n-1] = x[n]. \quad (3.53)$$

The system is a *summer*: it sums the input. It is an unstable system. There is a pole *on* the unit circle at $z = 1$, and the response to the bounded input $x[n]=u[n]$ is the unbounded output $y[n]=nu[n]$.

Example: Zeros and poles to other LTI system descriptions

An LTI system has a zero at 1, a pole at 3, and $H(0) = 1$. Compute: (1) the transfer function $H(z)$; (2) impulse response $h[n]$; (3) difference equation.

The poles and zeros determine $H(z)$ only to an unknown scale factor C. More information, such as $H(z_o)$ for some z_o, is needed to compute C.

Solution:

First, we compute the transfer function $H(z)$. The specified poles and zeros show that $H(z)$ has the form

$$H(z) = C\frac{z-1}{z-3}. \quad (3.54)$$

The specified value $H(0) = 1$ determines C using

$$1 = H(0) = C\frac{0-1}{0-3} \rightarrow C = 3 \quad (3.55)$$

so the transfer function $H(z)$ is

$$H(z) = 3\frac{z-1}{z-3}. \quad (3.56)$$

Then, having obtained the transfer function $H(z)$, we can now compute other LTI system descriptions.

The impulse response is the inverse z-transform of $H(z)$, which is computed by writing $H(z)$ as

$$H(z) = 3\frac{z-1}{z-3} = \frac{3z}{z-3} - \frac{3}{z-3}. \quad (3.57)$$

and then reading off $h[n]$ as

$$h[n] = 3(3)^n u[n] - 3(3)^{n-1}u[n-1]. \quad (3.58)$$

To compute the difference equation, write

$$H(z) = \frac{Y(z)}{X(z)} = 3\frac{z-1}{z-3}\left[\frac{z^{-1}}{z^{-1}}\right] = \frac{3-3z^{-1}}{1-3z^{-1}} \quad (3.59)$$

cross-multiply to obtain

$$Y(z)(1-3z^{-1}) = X(z)3(1-z^{-1}) \quad (3.60)$$

and read off the inverse z-transform as

$$y[n] - 3y[n-1] = 3x[n] - 3x[n-1]. \quad (3.61)$$

The system is unstable, since it has a pole at $|3| > 1$.

Example: Difference equation to other LTI system descriptions

An LTI system is described by the (1,3)-order ARMA difference equation

$$y[n] - y[n-1] = \quad (3.62)$$
$$x[n] + 2x[n-1] + 3x[n-2] + 4x[n-3].$$

Compute the transfer function and impulse response.

Solution: This example shows an important issue that arises when the order of the MA part exceeds the order of the AR part. Taking the z-transform,

$$Y(z)(1-z^{-1}) = X(z)(1+2z^{-1}+3z^{-2}+4z^{-3}). \quad (3.63)$$

Solving for transfer function $H(z) = \frac{Y(z)}{X(z)}$ gives

$$\begin{aligned} H(z) &= \frac{1+2z^{-1}+3z^{-2}+4z^{-3}}{1-z^{-1}}\left[\frac{z^3}{z^3}\right] \\ &= \frac{z^3+2z^2+3z+4}{z^2(z-1)}. \end{aligned} \quad (3.64)$$

$H(z)$ has a double pole at the origin $z = 0$. This is the inverse z-transform that was computed in the subsection dealing with multiple poles at the origin:

$$h[n] = \{\underline{1}, 3, 6, 10, 10, 10 \ldots\}$$

Now we see how such a problem can arise.

3.6 APPLICATION: LCCDEs to Difference Equations

Recall that while there is a clear analogy between continuous-time differential equations and discrete-time difference equations, the coefficients are quite different. This section discusses the general problem of discretizing continuous-time systems into discrete-time systems, so that they can be simulated directly on a computer. This is an important topic in many branches of engineering, in which systems modelled by differential equations are simulated on a computer.

We proceed by deriving the *backward difference* approximation to a derivative, and computing its frequency response. We then show how to convert a continuous-time LCCDE into a discrete-time ARMA difference equation whose behavior closely approximates the behavior of the continuous-time LCCDE.

3.6.1 Backward Difference

First, we derive an approximation to differentiation. Let T be the sampling interval. Eventually we will sample $x(t)$ and $y(t)$ at integer multiplies $t = nT$. For now, perform a Taylor series expansion of $x(t)$:

$$x(t-T) = x(t) - T\frac{dx}{dt}(t) + \frac{T^2}{2!}\frac{d^2x}{dt^2}(t) + \ldots \quad (3.65)$$

We use $x(t–T)$ instead of $x(t+T)$ in order to obtain a causal discrete-time system below. Note $\frac{dx}{dt}(t)$ means the derivative of $x(t)$ evaluated at time t.

We wish to truncate this series to obtain an approximation to the derivative $\frac{dx}{dt}(t)$ using $x(t)$ and $x(t–T)$. The usual step at this point is to assume that $T << 1$, so that we may neglect terms of order T^2 and higher. However, what actually matters is

the relative sizes of the terms $\frac{T^n}{n!}\frac{d^n x}{dt^n}(t), n = 1, 2 \ldots$:

$$T\frac{dx}{dt}(t) >> \frac{T^n}{n!}\frac{d^n x}{dt^n}(t), n = 2, 3 \ldots \tag{3.66}$$

It is not apparent if this is true. This motivates the frequency response analysis in the section to follow.

In order to derive the backward difference appproximation, suppose for now that (3.66) is true. Then

$$x(t - T) \approx x(t) - T\frac{dx}{dt}(t). \tag{3.67}$$

Solving for $\frac{dx}{dt}(t)$ gives the *backward difference* approximation to $\frac{dx}{dt}(t)$ in terms of $x(t)$ and $x(t–T)$:

$$\frac{dx}{dt}(t) \approx \frac{x(t) - x(t - T)}{T}. \tag{3.68}$$

This suggests the ideal differentiator $y(t) = \frac{dx}{dt}(t)$ can be approximated by the backward-difference system

$$y(t) \approx \frac{x(t) - x(t - T)}{T}. \tag{3.69}$$

3.6.2 Frequency Response of Backward Difference Approximation

We now avoid dealing with the condition (3.66) by computing the frequency response of (3.69), and comparing it to the frequency response of the ideal differentiator $y(t) = \frac{dx}{dt}(t)$. The analysis in this subsection uses *continuous-time* results: $\Omega{=}2\pi f$ instead of ω.

The frequency response of the backward difference system can be computed by taking the Fourier transform of (3.69). Recall

$$F[x(t - T)] = e^{-j\Omega T}F[x(t)] = e^{-j\Omega T}X(j\Omega), \tag{3.70}$$

where $X(j\Omega)$ is the Fourier transform of $x(t)$. Using this property, the Fourier transform of (3.69) is

$$\begin{aligned} Y(j\Omega) &= \frac{1}{T}[X(j\Omega) - e^{-j\Omega T}X(j\Omega)] \\ &= X(j\Omega)\frac{1 - e^{-j\Omega T}}{T}, \end{aligned} \tag{3.71}$$

so that the frequency response $H(j\Omega)_{\text{back}}$ of the backward difference system (3.69) is

$$H(j\Omega)_{\text{back}} = \frac{Y(j\Omega)}{X(j\Omega)} = \frac{1 - e^{-j\Omega T}}{T}. \tag{3.72}$$

Recall that the series expansion of e^x is

$$e^x = 1 + x + x^2/2! + \ldots \tag{3.73}$$

If $|x| << 1$, we may neglect all terms of order higher than one in x. This leaves

$$e^x \approx 1 + x \text{ if } |x| << 1. \tag{3.74}$$

Now let us assume not that $T << 1$, but $\Omega T << 1$. Setting $x{=}{-}j\Omega T$ in (3.74) gives

$$\begin{aligned} H(j\omega)_{\text{back}} &= \frac{1 - e^{-j\Omega T}}{T} \\ &\approx \frac{1 - [1 - (j\Omega T)]}{T} \\ &= j\Omega. \end{aligned} \tag{3.75}$$

Now we determine the frequency response of the ideal differentiator $y(t) = \frac{dx}{dt}$. Recall

$$F\left[\frac{dx}{dt}\right] = j\Omega F[x(t)] = j\Omega X(j\Omega). \tag{3.76}$$

The frequency response of the ideal differentiator is

$$H(j\Omega)_{\text{diff}} = \frac{Y(j\Omega)}{X(j\Omega)} = j\Omega. \tag{3.77}$$

The result of all this is that $H(j\Omega)_{\text{back}} \approx H(j\Omega)_{\text{diff}}$ when $\Omega T << 1$. This is illustrated in the next figure.

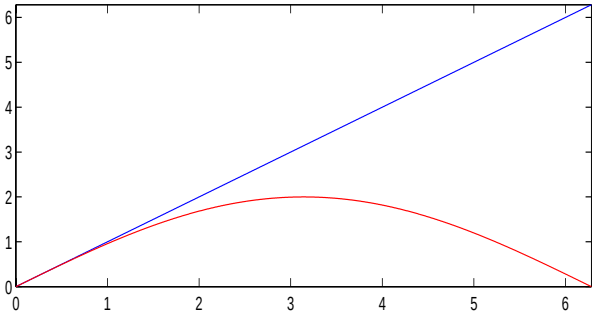


Figure 3.1: Gains of ideal differentiator (blue) and backward difference (red) plotted for $0 \le \Omega \le \frac{\pi}{T}$.

$|H(j\Omega)_{\text{diff}}|$ is in blue, and $|H(j\Omega)_{\text{back}}|$ is in red, for $0 \le \Omega \le \frac{2\pi}{T}$. Both scales are in multiples of $\frac{1}{T}$.

Matlab code for this plot:

```
W=linspace(0,2*pi,1000);
H1=W;H2=abs(1-exp(j*W));
subplot(211),plot(W,H1,W,H2,'r')
```

The plot shows that $|H(j\Omega)_{\text{back}}| \approx |H(j\Omega)_{\text{diff}}|$ for $0 \leq \Omega << \frac{2\pi}{T}$, confirming the analysis. But when $\Omega=\frac{1}{T}$, then $|H(j\Omega)_{\text{back}}|$ becomes smaller than $|H(j\Omega)_{\text{diff}}|$, and actually reaches zero when $\Omega=\frac{2\pi}{T}$!

3.6.3 Sampling

The result of the previous subsection is that the backward difference operator will approximate the ideal differentiator for $0 \leq \Omega << \frac{2\pi}{T}$. This is a much more useful result than relying on (3.66), since now we can use the *sampling theorem*.

Sampling $x(t)$ and $y(t)$ at integer multiples $t=nT$ of T results in the discrete-time signals $x[n]=x(nT)$ and $y[n]=y(nT)$ for integers n. The backward difference system becomes the MA difference equation

$$y[n] = \frac{x[n] - x[n-1]}{T}. \tag{3.78}$$

The impulse response can be read off as

$$h_{\text{back}}[n] = \left\{ \underline{\frac{1}{T}}, -\frac{1}{T} \right\} \tag{3.79}$$

and the transfer function is then

$$H_{\text{back}}(z) = Z[h_{\text{back}}[n]] = \frac{1 - z^{-1}}{T} \tag{3.80}$$

Derivatives of order higher than the first can be approximated by repeatedly applying the backward difference operator:

$$\begin{aligned} \frac{d^2x}{dt^2}(t) &\approx \frac{\frac{dx}{dt}(t) - \frac{dx}{dt}(t-T)}{T}. \\ \frac{d^3x}{dt^3}(t) &\approx \frac{\frac{d^2x}{dt^2}(t) - \frac{d^2x}{dt^2}(t-T)}{T}. \end{aligned} \tag{3.81}$$

So $\frac{d^Nx}{dt^N}$ is approximated by a series connection of N backward difference operators. After sampling $x(t)$ to $x[n] = x(nT)$ and $y(t)$ to $y[n] = y(nT)$, the transfer function of a series connection of N systems, each having transfer function $H_{\text{back}}(z)$, is

$$H^N(z) = \left(\frac{1 - z^{-1}}{T} \right)^N. \tag{3.82}$$

Now we show how to discretize a differential equation (LCCDE). The general form of an LCCDE is

$$\sum_{i=0}^{N} a_{N-i} \frac{d^i y}{dt^i} = \sum_{i=0}^{M} b_{M-i} \frac{d^i x}{dt^i}. \tag{3.83}$$

Now sample $x(t)$ and $y(t)$ at integer multiples $t = nT$ of T. This results in the discrete-time signals $x[n] = x(nT)$ and $y[n] = y(nT)$ for integers n. $x[n]$ and $y[n]$ have z-transforms $X(z)$ and $Y(z)$, respectively.

The derivatives $\frac{d^iy}{dt^i}$ and $\frac{d^iy}{dt^i}$ are all approximated by series connections of backward difference operators. Each series connection has a transfer function of the form (3.82). Substituting (3.82) in the discretized (3.83) gives

$$\sum_{i=0}^{N} a_{N-i} \left(\frac{1 - z^{-1}}{T} \right)^i Y(z) = \sum_{i=0}^{M} b_{M-i} \left(\frac{1 - z^{-1}}{T} \right)^i X(z). \tag{3.84}$$

This expression must be simplified by expanding all of the $\left(\frac{1-z^{-1}}{T}\right)^i$ and collecting terms of coefficients of powers of z^{-i}. The ARMA difference equation can then be read off of the inverse z-transform. If $x(t)$ is given, it must be discretized by substituting $t=nT$.

The procedure for discretizing an LCCDE into a discrete-time system for purposes of simulation is best illustrated by an example.

Example: Discrete-time simulation of differential equation

We are given the LCCDE

$$\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 2y(t) = 2e^{-3t}u(t). \tag{3.85}$$

The goal is to discretize this LCCDE into an ARMA difference equation and compute the solution to the latter numerically by iteration. We then compare this numerical solution to the actual solution computed using the Laplace transform. All initial conditions are zero. We use T=0.001 for clarity.

Solution:
First, we discretize the right side $x(t) = 2e^{-3t}u(t)$:

$$x[n] = x(0.001n) = 2e^{-0.003n}u[n] \tag{3.86}$$

which has the z-transform

$$X(z) = Z[x[n]] = \frac{2z}{z - e^{-0.003}}. \tag{3.87}$$

Second, we simplify $\left(\frac{1-z^{-1}}{T}\right)$ and $\left(\frac{1-z^{-1}}{T}\right)^2$:

$$\begin{aligned}\left(\frac{1-z^{-1}}{0.001}\right) &= 1000(1-z^{-1}).\\ \left(\frac{1-z^{-1}}{0.001}\right)^2 &= 10^6(1-2z^{-1}+z^{-2}).\end{aligned} \tag{3.88}$$

Third, we map the left side of the LCCDE to (3.84):

$$\begin{aligned}&[10^6(1-2z^{-1}+z^{-2}) + 3(10^3)(1-z^{-1}) + 2]Y(z)\\ &= \frac{2z}{z - e^{-0.003}}.\end{aligned} \tag{3.89}$$

Fourth, we collect terms of coefficients of z^0, z^{-1}, and z^{-2} on the left side. This gives

$$\begin{aligned}&[z^0(1003002) - z^{-1}(2003000) + z^{-2}(10^6)]Y(z)\\ &= \frac{2z}{z - e^{-0.003}}.\end{aligned} \tag{3.90}$$

Finally, we read off the difference equation, which is

$$\begin{aligned}&(1003002)y[n] - (2003000)y[n-1] + (10^6)y[n-2]\\ &= 2e^{-0.003n}u[n].\end{aligned} \tag{3.91}$$

This ARMA difference equation can be implemented numerically by rewriting it as

$$y[n] = \frac{2e^{-0.003n}}{1003002} + \frac{2003000}{1003002}y[n-1] - \frac{1000000}{1003002}y[n-2] \tag{3.92}$$

and then recursively computing $y[n]$ by setting $n = 0, 1, 2\ldots$. This recursive computation can be implemented very quickly on a computer using the Direct Form II realization presented earlier.

The *actual* solution to the LCCDE can be computed easily using the Laplace transform.

Taking the Laplace transform of the LCCDE gives

$$Y(s)[s^2 + 3s + 2] = L[2e^{-3t}u(t)] = 2/(s+3). \tag{3.93}$$

Solving for $Y(s)$ gives

$$Y(s) = \frac{2}{(s^2+3s+2)(s+3)}. \tag{3.94}$$

A partial fraction expansion gives

$$Y(s) = \frac{1}{s+1} - -\frac{2}{s+2} + \frac{1}{s+3} \tag{3.95}$$

from which we can read off $y(t)$ as

$$y(t) = [e^{-t} - 2e^{-2t} + e^{-3t}]u(t). \tag{3.96}$$

The computed solution (in blue) and the actual solution (in red) are both plotted in the next figure. It is clear that the two solutions are very close to each other. The maximum error is 0.00035 at t=0.25.

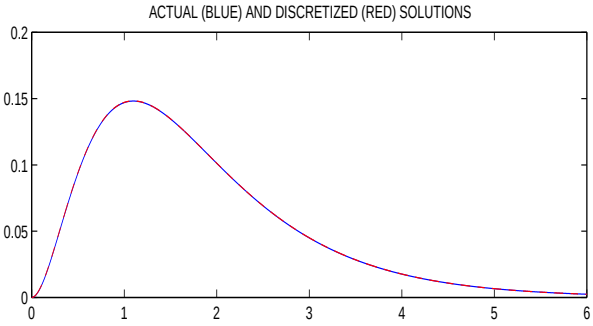


Figure 3.2: Computed (in blue) and actual (in red).

Matlab code for this ((3.91) is divided by 10^6):

```
T=linspace(0,6,6000);
X=exp(-T)-2*exp(-2*T)+exp(-3*T);
B=[0.000002];A=[1.003002 -2.003 1];
Y=filter(B,A,exp(-3*T));
plot(T,X,T,Y,'r--')
```

Chapter 4

Two-Sided Z-Transforms

4.1 Definitions

The *two*-sided (*bi*-lateral) z-transform is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{4.1}$$

Compare to the *one*-sided (*uni*-lateral) z-transform

$$X(z) = \sum_{n=0}^{\infty} x[n]z^{-n} \tag{4.2}$$

These are identical for causal $x[n]$.

The two-sided time delay property is simpler:

$$Z(x[n-D]) = z^{-D}X(z) \text{ for all } D. \tag{4.3}$$

$X(z)$ is now the *two*-sided z-transform of $x[n]$.

Example: Z-transforms for finite-duration. Compute both z-transforms of $\{3, \underline{1}, 4\}$.

Solution:
One-sided: $1{+}4z^{-1}$ (omit $x[-1]{=}3$).
Two-sided: $3z{+}1{+}4z^{-1}$ (include $x[-1]{=}3$).

- Use the *one*-sided z-transform for:
  - Solving difference equations with non-zero initial conditions;
  - *Causal* (unstable?) inverse z-transforms;
  - Computing convolutions of infinite-duration *causal* geometric signals.
- Use the *two*-sided z-transform for:
  - *Stable* and *non-causal* inverse z-transforms;
  - Computing convolutions of infinite-duration *two*-sided geometric signals.

We also define signals in terms of their support (the regions where they are non-zero) as follows. $x[n]$ is:

- Causal if $x[n]{=}0$ for $n < 0$.
- Anticausal if $x[n] = 0$ for $n \geq 0$.
- Two-sided if it is neither causal nor anticausal.
- Right-sided if $x[n]{=}0$ for $n < N < 0$ for some N.

Example: $\{3, 1, 4, \underline{1}, 5, 9 \ldots\}$ is a right-sided signal.

- Left-sided if $x[n]{=}0$ for $n > N \geq 0$ for some N.

Example: $\{\ldots 3, 1, 4, \underline{1}, 5, 9\}$ is a left-sided signal.

4.2 Geometric Signals

4.2.1 Causal Geometric Signals

Recall the infinite geometric series

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r} \text{ if and only if } |r| < 1. \tag{4.4}$$

The two-sided z-transform of the causal geometric

$$x[n] = a^n u[n] = 0 \text{ for } n < 0 \tag{4.5}$$

is computed by setting $r{=}(a/z)$:

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} a^n u[n] z^{-n} = \sum_{n=0}^{\infty} \left(\frac{a}{z}\right)^n \\ &= \frac{1}{1-\left(\frac{a}{z}\right)} = \frac{z}{z-a} \\ &\text{if} \quad |a/z| < 1 \rightarrow \text{ ROC } |z| > |a|. \end{aligned} \tag{4.6}$$

The *Region Of Convergence* (ROC) is the set of z for which the infinite geometric series converges. This expression for $X(z)$ is only true for $z \in$ROC.

This is the same as the one-sided z-transform.

4.2.2 Anticausal Geometric Signals

Note that the causal geometric signal

$$x[n] = \left(\frac{1}{2}\right)^n u[n] = \left\{\underline{1}, \frac{1}{2}, \frac{1}{4}, \frac{1}{8} \ldots\right\} \tag{4.7}$$

and the anticausal geometric signal

$$x[-n] = (2)^n u[-n] = \left\{\ldots \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, \underline{1}\right\} \tag{4.8}$$

both decay to zero as $|n| \to \infty$.

The two-sided z-transform of the anticausal geometric signal (note the signs)

$$x[n] = -a^n u[-n-1] = 0 \text{ for } n \geq 0 \tag{4.9}$$

is computed by setting $r{=}(z/a)$, not (a/z):

$$\begin{aligned} X(z) &= -\sum_{n=-\infty}^{\infty} a^n u[-n-1] z^{-n} = -\sum_{n=-\infty}^{-1} \left(\frac{a}{z}\right)^n \\ &= -\sum_{n'=1}^{\infty} \left(\frac{a}{z}\right)^{-n'} = -\left(\frac{z}{a}\right) \sum_{n'=0}^{\infty} \left(\frac{z}{a}\right)^{n'} \\ &= -\left(\frac{z}{a}\right) \frac{1}{1-\left(\frac{z}{a}\right)} \left[\frac{-a}{-a}\right] = \frac{z}{z-a} \\ &\text{if} \quad |z/a| < 1 \rightarrow \text{ ROC } |z| < |a|. \end{aligned} \tag{4.10}$$

Two signals have the same z-transform!

This means that the inverse z-transform is no longer unique: The inverse z-transform of $\frac{z}{z-2}$ could be either $(2)^n u[n]$ or $-(2)^n u[-n{-}1]$! While this may double the chance of getting the right answer, clearly there is a problem.

The solution to this issue is the inverse z-transform is not unique unless the ROC of $X(z)$ is also specified. Specifically, the inverse z-transform of

- $\frac{z}{z-2}$ with ROC $\{z : |z| > 2\}$ is $(2)^n u[n]$.
- $\frac{z}{z-2}$ with ROC $\{z : |z| < 2\}$ is $-(2)^n u[-n{-}1]$.

In fact, we shall soon see that the ROC, *not* $X(z)$, determines whether the inverse z-transform is stable, causal, anticausal, or two-sided.

4.2.3 Sums of Geometric Signals

The z-transform of a sum of geometric signals, where each can be either causal or anticausal, is the sum of the z-transforms. The ROC of the sum is the intersection of the ROCs for each signal. If this is the empty set, the z-transform is undefined.

Example: Sum of two geometric signals.
Compute the z-transform of

$$x[n] = a^n u[n] + b^n u[-n-1]. \tag{4.11}$$

Solution:

$X(z){=}z/(z{-}a) \quad - \quad z/(z{-}b) \quad {=}z/(z{-}a){-}z/(z{-}b).$
$\text{ROC}{=}\{|z| > |a|\} \bigcap \{|z| < |b|\} \ {=}\{|a| < |z| < |b|\}.$

- If $|b| < |a|$, then the ROC=$\emptyset$.
  The z-transform of $x[n]$ is then undefined!
- Watch the sign in $X(z)$ (this is easy to miss).
- Although $\{z : |z| > |a|\}$ is correct, we usually abbreviate this to $\{|z| > |a|\}$

In general, we have

$$\begin{aligned} x[n] &= \underbrace{\sum_i A_i p_i^n u[n]}_{\text{CAUSAL}} + \underbrace{\sum_j B_j q_j^n u[-n-1]}_{\text{ANTICAUSAL}} \\ X(z) &= \sum_i A_i \frac{z}{z-p_i} - \sum_j B_j \frac{z}{z-q_j} \\ \text{ROC} &= \left[\bigcap_i \{|z| > |p_i|\}\right] \bigcap \left[\bigcap_j \{|z| < |q_j|\}\right] \\ &= [\max\{p_i\}] < |z| < [\min\{q_j\}]. \end{aligned} \tag{4.12}$$

• The ROC is an *annulus* (ring) whose:
• Inner radius is the largest magnitude of the poles comprising the causal part of $x[n]$;
• Outer radius is the smallest magnitude of the poles comprising the anticausal part of $x[n]$.
• Unless all of the causal poles have magnitudes smaller than all of the anticausal poles, then the z-transform is undefined.
• If $x[n]$ is causal, the outer radius is ∞.
• If $x[n]$ is anticausal, the inner radius is 0.

Example: Two-sided z-transform.
Compute the two-sided z-transform of
$9(2)^n u[n]+8(3)^n u[n]–7(4)^n u[–n–1]+6(5)^n u[–n–1]$.

Solution:
$X(z)=\frac{9z}{z-2}+\frac{8z}{z-3}+\frac{7z}{z-4}-\frac{6z}{z-5}$.
Note the signs of each term. Watch this!
ROC=$\{|z|>2\}\cap\{|z|>3\}\cap\{|z|<4\}\cap\{|z|<5\}$
ROC=$\{z:3<|z|<4\}$.
Largest causal pole<smallest anticausal pole.

Example: Two-sided z-transform.
Compute the two-sided z-transform of
$(-2)^n u[n]+(\frac{1}{2})^n u[-n-1]$.

Solution:
ROC=$\{|z|>|-2|\}\bigcap\{|z|<\frac{1}{2}\}=\emptyset$.
The z-transform is undefined.
There is no z for which both geometric series converge. We avoided some algebra by noticing this first.

4.2.4 ROCs for Finite Duration, Left-sided and Right-sided Signals

The above rules for ROC must be modified slightly for signals that are not sums of geometric signals:

Signal Type and Length	ROC
Causal & finite duration	$\{0<\|z\|\leq\infty\}$
Anticausal & finite length	$\{0\leq\|z\|<\infty\}$
Two-sided & finite length	$\{0<\|z\|<\infty\}$
Right-sided	$\{0<\|z\|<\infty\}$
Left-sided	$\{0<\|z\|<\infty\}$
Impulse $\delta[n]$	$\{0\leq\|z\|\leq\infty\}$

These are easily seen by considering whether

$$X(z)=\ldots+x[-1]z+x[0]+x[1]z^{-1}+\ldots \quad (4.13)$$

will blow up at z=0 or $z\rightarrow\infty$.

4.3 Inverse Z-transforms

The ROC is just as important as $X(z)$ in computing inverse two-sided z-transforms. For causality and stability, the ROC is important; $X(z)$ is irrelevant!

4.3.1 Multiple Inverse Z-transforms

Let the partial fraction expansion of $\frac{X(z)}{z}$ be

$$\frac{X(z)}{z}=A_0\frac{1}{z-0}+\sum_{i=1}^{N}A_i\frac{1}{z-p_i} \quad (4.14)$$

and multiply through by z. This gives

$$X(z)=A_0+\sum_{i=1}^{N}A_i\frac{z}{z-p_i} \quad (4.15)$$

where $\{p_i\}$ are the poles of $X(z)$ and $\{A_i\}$ are their associated residues. The (causal) inverse one-sided z-transform of $X(z)$ is, as in the previous chapter,

$$x[n]=A_0\delta[n]+\sum_{i=1}^{N}A_i p_i^n u[n] \quad (4.16)$$

The inverse two-sided z-transform is

$$x[n]=A_0\delta[n]+\sum_{i=1}^{N}A_i\begin{cases}p_i^n u[n] & \{|z|>|p_i|\}\\ -p_i^n u[-n-1] & \{|z|<|p_i|\}\end{cases} \quad (4.17)$$

since there are two choices (causal or anticausal) for each term in the partial fraction expansion, depending on the choice of ROC for that term.

This suggests that there are 2^N possible inverse z-transforms, since the choice for each term can be made independently. But there are actually only N+1 possible inverse z-transforms, since the other choices lead to ROCs that are empty sets. To see why, consider the following two examples:

Example: Inverse 2-sided z-transform.
Compute all of the inverse z-transforms of
$X(z)=\frac{2z^2-2.5z}{z^2-2.5z+1}$.

Solution:
The partial fraction of $\frac{X(z)}{z}$ is

$$\frac{X(z)}{z}=\frac{1}{z-(1/2)}+\frac{1}{z-2}. \quad (4.18)$$

Multiplying through by z gives

$$X(z)=\frac{z}{z-(1/2)}+\frac{z}{z-2}. \quad (4.19)$$

Each of the two terms has two possible inverse z-transforms, one causal and one anticausal. So it seems that there are 2^2=4 possible inverse two-sided z-transforms of $X(z)$:

- $x_1[n]=(\frac{1}{2})^n u[n]+(2)^n u[n]$ with ROC $\{|z|>\frac{1}{2}\}\bigcap\{|z|>2\}=\{|z|>2\}$ (causal).
- $x_2[n]=-(\frac{1}{2})^n u[-n-1]-(2)^n u[-n-1]$ with ROC $\{|z|<\frac{1}{2}\}\bigcap\{|z|<2\}=\{|z|<\frac{1}{2}\}$ (anticausal).
- $x_3[n]=(\frac{1}{2})^n u[n]-(2)^n u[-n-1]$ with ROC $\{|z|>\frac{1}{2}\}\bigcap\{|z|<2\}=\{\frac{1}{2}<|z|<2\}$ (stable).
- $x_4[n]=-(\frac{1}{2})^n u[-n-1]+(2)^n u[n]$ with ROC $\{|z|<\frac{1}{2}\}\bigcap\{|z|>2\}=\emptyset$. (valid for no z).

So there are actually only 2+1=3, not 2^2=4, possible inverse two-sided z-transforms of $X(z)$.

Example: Inverse 2-sided z-transform.
Compute all of the inverse z-transforms of $X(z)=\frac{z}{z-2}+\frac{z}{z-3}+\frac{z}{z-4}$.

Solution:
It seems there are 2^3=8 inverse z-transforms

$$\begin{cases}(2)^n u[n]\\-(2)^n u[-n-1]\end{cases}+\begin{cases}(3)^n u[n]\\-(3)^n u[-n-1]\end{cases}+\begin{cases}(4)^n u[n]\\-(4)^n u[-n-1]\end{cases}.$$

But examine the ROCs for each of these choices:

- $\{|z|<2\}\cap\{|z|<3\}\cap\{|z|<4\}=\{|z|<2\}$.
- $\{|z|>2\}\cap\{|z|<3\}\cap\{|z|<4\}=\{2<|z|<3\}$.
- $\{|z|<2\}\cap\{|z|>3\}\cap\{|z|<4\}=\emptyset$.
- $\{|z|>2\}\cap\{|z|>3\}\cap\{|z|<4\}=\{3<|z|<4\}$.
- $\{|z|<2\}\cap\{|z|<3\}\cap\{|z|>4\}=\emptyset$.
- $\{|z|>2\}\cap\{|z|<3\}\cap\{|z|>4\}=\emptyset$.
- $\{|z|<2\}\cap\{|z|>3\}\cap\{|z|>4\}=\emptyset$
- $\{|z|>2\}\cap\{|z|>3\}\cap\{|z|>4\}=\{|z|>4\}$.

So there are only 3+1=4 inverse z-transforms:

$\underline{\{|z|<2\}}$: $-(2)^n u[-n-1]-(3)^n u[-n-1]-(4)^n u[-n-1]$.
$\underline{\{2<|z|<3\}}$: $(2)^n u[n]-(3)^n u[-n-1]-(4)^n u[-n-1]$.
$\underline{\{3<|z|<4\}}$: $(2)^n u[n]+(3)^n u[n]-(4)^n u[-n-1]$.
$\underline{\{|z|>4\}}$: $(2)^n u[n]+(3)^n u[n]+(4)^n u[n]$ (causal).

4.3.2 Stable Inverse Z-transform

In general, there is always one causal, and one anticausal, inverse two-sided z-transforms. If there are no poles on the unit circle $\{|z|=1\}$, then there is also one BIBO stable inverse two-sided z-transform.

To compute the BIBO stable inverse two-sided z-transform of $X(z)$, proceed as follows:

- Order the poles $\{p_1 \dots p_N\}$ of $X(z)$ in increasing order of magnitudes (group conjugate poles): $|p_1|<\ldots<|p_{M-1}|<1<|p_M|<\ldots<|p_N|$.
- Compute the partial fraction expansion of $\frac{X(z)}{z}$: $\frac{X(z)}{z}=A_0\frac{1}{z-0}+\sum_{i=1}^{N}A_i\frac{1}{z-p_i}$.
- Multiply this by z: $X(z)=A_0+\sum_{i=1}^{N}A_i\frac{z}{z-p_i}$.
- The stable inverse two-sided z-transform is $A_0\delta[n]+\sum_{i=1}^{M-1}A_i p_i^n u[n]-\sum_{i=M}^{N}A_i p_i^n u[-n-1]$.

4.4 ROCs, Stability, Causality

The above shows that the ROC, not $X(z)$, determine whether the inverse two-sided z-transform is stable or causal. Let ROC have the form $a<|z|<b$. Then

Signal Type	ROC For It
BIBO Stable	$\|a\|<1<\|b\|$
Causal	$\|b\|\to\infty$
Anticausal	$\{a=0\}\subset$ROC
Two-sided	$0<\|a\|$ & $\|b\|<\infty$
Stable &causal	$\|a\|<1$ & $\|b\|\to\infty$

Note that a stable and causal system has an ROC of the form $\{|z|>|a|<1\}$, so there are no poles outside the unit circle (all poles inside the unit circle).

Example: ROC→Stability and Causality.
What can you say about the stability and causality of systems whose ROC has the following forms:
(1) $\{z:|z|>a>1\}$
(2) $\{z:|z|>a<1\}$
(3) $\{z:|z|<a<1\}$
(4) $\{z:|z|<a>1\}$
(5) $\{z:1<a<|z|<b\}$
(6) $\{z:a<|z|<b<1\}$
(7) $\{z:1>a<|z|<b>1\}$

Solution:
(1) $\{z : |z| > a > 1\} \rightarrow$ causal & unstable.
(2) $\{z : |z| > a < 1\} \rightarrow$ causal & stable.
(3) $\{z : |z| < a < 1\} \rightarrow$ anticausal & unstable.
(4) $\{z : |z| < a > 1\} \rightarrow$ anticausal & stable.
(5) $\{z : 1 < a < |z| < b\} \rightarrow$ 2-sided & unstable
(6) $\{z : a < |z| < b < 1\} \rightarrow$ 2-sided & unstable
(7) $\{z : 1 > a < |z| < b > 1\} \rightarrow$ 2-sided & stable

Knowledge of $X(z)$ would be a red herring–it would be of no use in answering these questions!

Example: Two-sided convolution.
$2^n u[n]–3^n u[-n–1] \rightarrow \boxed{y[n]=y[n–1]+2x[n]} \rightarrow y[n]$.

Solution: $h[n]=2u[n]$ from before. We need to compute $y[n]=(2^n u[n]–3^n u[-n–1]) * 2u[n]$. So:

$$\begin{aligned} Y(z) &= \left[\underbrace{\frac{z}{z-2}\frac{z-3}{z-3}}_{\text{ROC:}|z|>2} + \underbrace{\frac{z}{z-3}\frac{z-2}{z-2}}_{\text{ROC:}|z|<3} \right] \underbrace{\frac{2z}{z-1}}_{\text{ROC:}|z|>1} \\ &= \underbrace{\frac{2z^2-5z}{z^2-5z+6}}_{\text{ROC:}2<|z|<3} \underbrace{\frac{2z}{z-1}}_{\text{ROC:}|z|>1} = \frac{4z^3-10z^2}{z^3-6z^2+11z-6} \\ &= 3\frac{z}{z-3} + 4\frac{z}{z-2} - 3\frac{z}{z-1} \end{aligned} \quad (4.20)$$

where we computed the partial fractions using

```
[R P]=residue([4 -10 0],[1 -6 11 -6]);
```

The ROC of $Y(z)$ is then

$\{|z| > 2\} \bigcap \{|z| > 1\} \bigcap \{|z| < 3\} = \{2 < |z| < 3\}$.

So $y[n]=4(2)^n u[n]–3(3)^n u[-n–1]–3u[n]$.

Check this in Matlab by *truncating* the input:

```
X=[-3.^ [-10:-1],2.^ [0:10]];
Y=[-3*3.^ [-10:-1],4*2.^ [0:10]-3];
Z=filter([2],[1 -1],X);[Z;Y]. Z and Y agree.
```

4.5 Deconvolution: Overview

A *deconvolution* problem is to reconstruct the input of a system from the output of the system and its impulse response. We wish to solve $y[n]=h[n] * x[n]$ for $x[n]$, given knowledge of $y[n]$ and $h[n]$. We wish to undo the convolution, hence "deconvolution."

4.5.1 Deconvolution by Inverses

One possible way to solve a deconvolution problem is to determine the *inverse system* of the given system. The inverse system of a given system undoes the original system. This is exactly what we want to do to solve the deconvolution problem.

Let the impulse response of the original system be $h[n]$, and that of the inverse system be $g[n]$. Then

- $x[n] \rightarrow \boxed{h[n]} \rightarrow y[n] \rightarrow \boxed{g[n]} \rightarrow x[n]$.

Let the transfer function of the original system be $H(z)$ and that of the inverse system be $G(z)$. Then

- $y[n] = h[n] * x[n] \rightarrow Y(z) = H(z)X(z)$
- $x[n] = g[n] * y[n] \rightarrow X(z) = G(z)Y(z)$
- $H(z)G(z) = 1 \rightarrow G(z) = \frac{1}{H(z)}$.

This is straightforward. However,

- Original system $H(z)$ is causal and BIBO-stable if and only if all of its poles lie in $|z| < 1$.
- Inverse system $G(z)$ is causal and BIBO-stable if and only if all of its poles lie in $|z| < 1$.

Since the poles of $G(z)$ are the zeros of $H(z)$, both the original system and the inverse system are BIBO stable if and only if $H(z)$ has all of its poles and zeros inside the unit circle. Such a system is called a *minimum phase* system.

A *continuous-time* system is minimum phase if all of its poles and zeros are in the left half-plane. For discrete-time systems, the condition "in the left half-plane" is changed to "inside the unit circle." In both cases, the significance of being minimum phase is that the system is causal and stable and has a causal and stable inverse system.

4.5.2 Deconvolution by ARMA Difference Equations

Any LTI system described by an ARMA difference equation in which the coefficient of $x[n]$ is nonzero has a *proper* transfer function, meaning that the numerator and denominator polynomials have the same degree. To show this, let NM be the larger of N and M. Then the ARMA difference equation

$$\sum_{i=0}^{N} a_i y[n-i] = \sum_{i=0}^{M} b_i x[n-i] \tag{4.21}$$

has the transfer function

$$H(z) = \frac{\sum_{i=0}^{M} b_i z^{-i}}{\sum_{i=0}^{N} a_i z^{-i}} \left[\frac{z^{NM}}{z^{NM}}\right] = \frac{\sum_{i=0}^{M} b_i z^{NM-i}}{\sum_{i=0}^{N} a_i z^{NM-i}}. \tag{4.22}$$

$a_0 = 1$ by definition, so if $b_0 \neq 0$ then $H(z)$ is proper, and the inverse system $G(z) = \frac{1}{H(z)}$ is also proper.

Consider the system described by ARMA difference equation (4.21) and having transfer function (4.22). The transfer function $G(z)$ for the inverse system is then

$$G(z) = \frac{1}{H(z)} = \frac{X(z)}{Y(z)} = \frac{\sum_{i=0}^{N} a_i z^{-i}}{\sum_{i=0}^{M} b_i z^{-i}}. \tag{4.23}$$

Cross-multiplying and taking the inverse z-transform,

$$\sum_{i=0}^{M} b_i x[n-i] = \sum_{i=0}^{N} a_i y[n-i] \tag{4.24}$$

The ARMA difference equation for the inverse system is the ARMA difference equation for the original system with input and output interchanged! So implementation of the inverse system using an ARMA difference equation is no more complicated than implementation of the original system using an ARMA difference equation. *This only works if the original system is minimum phase.*

4.6 APPLICATION: Dereverberation

Dereverberation is a special case of deconvolution, in which the original system is minimum phase.

4.6.1 Dereverberation: Problem

Reverberation of a signal is the act of adding delayed and geometrically-weighted copies of the signal to itself. Each copy of the signal is the previous copy multiplied by a *reflection coefficient* r and delayed by D sec. The reverbed version $y(t)$ of a signal $x(t)$ is

$$\begin{aligned} y(t) &= x(t) + rx(t-D) + r^2 x(t-2D) + \dots \\ &= \sum_{i=0}^{\infty} r^i x(t-iD). \end{aligned} \tag{4.25}$$

Since reflections cannot gain strength, $|r| < 1$.

Reverberation of a voice can make it sound richer and fuller if the delay is a few msec. This is why a singing voice sounds much better in a shower. In music recording, reverbing is called *overdubbing*.

However, reverberation of an acoustic signal in a shallow sea (such as the Gulf of Mexico) can mask important features of the signal. The signal is reflected off both the sea surface and sea bottom, creating what are called *water-column multiple reflections*. In this case, *de*-reverberation of the signal is desired. This is a special case of deconvolution.

We employ a digital signal processing system with a sampling rate of S samples per sec. The discrete-time sampled original signal is $x[n]$, and $y[n]$ is the reverbed version of $x[n]$. A delay of D seconds is a delay of DS samples. We assume that the sampling rate S is chosen so that the discrete-time delay is an integer $M{=}DS$. After sampling, the reverberating system is

$$\begin{aligned} y[n] &= x[n] + rx[n-M] + r^2 x[n-2M] + \dots \\ &= \sum_{i=0}^{\infty} r^i x[n-iM]. \end{aligned} \tag{4.26}$$

The goal is to recover $x[n]$ from $y[n]$.

4.6.2 Dereverberation: Solution

A good rule of thumb for problems involving passage of a signal through an LTI system is:

If you have no idea how to start, take z-transforms.

The z-transform of (4.26) is

$$Y(z) = \sum_{i=0}^{\infty} r^i X(z) z^{-iM} = X(z) \sum_{i=0}^{\infty} (rz^{-M})^i. \quad (4.27)$$

Recalling the formula for the sum of an infinite geometric series, this becomes

$$Y(z) = \frac{X(z)}{1 - rz^{-M}} = \frac{z^M X(z)}{z^M - r} \quad (4.28)$$

provided $|rz^{-M}| < 1$. Then

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - rz^{-M}} = \frac{z^M}{z^M - r}. \quad (4.29)$$

The transfer function $H(z)$ for reverberation has M zeros at the origin z=0 and M poles inside the unit circle at $\{r^{1/M} e^{j2\pi k/M}, k{=}0 \ldots (M{-}1)\}$. So $H(z)$ is minimum phase, and it has a BIBO-stable and causal inverse system. The transfer function of the inverse system is

$$G(z) = \frac{1}{H(z)} = \frac{z^M - r}{z^M} = 1 - rz^{-M}. \quad (4.30)$$

The impulse response of the inverse system is then

$$g[n] = Z^{-1}[G(z)] = \{\underline{1}, \underbrace{0 \ldots 0}_{M-1}, -r\}. \quad (4.31)$$

Since $g[n]$ has finite duration, the difference equation can be read off as the MA equation

$$x[n] = y[n] - ry[n - M]. \quad (4.32)$$

Example: Dereverberation.

The signal shown in the next figure was produced by reverbing a short-duration signal with reflection coefficient r=0.6 and M=1. Compute the signal.

Solution:

Inserting M=1 and r=0.6 in (4.32) gives

$$x[n] = y[n] - 0.6y[n - 1] \quad (4.33)$$

Filtering the signal with this system produces the signal shown in the next figure. This is the original signal; its values are the first few digits of π.

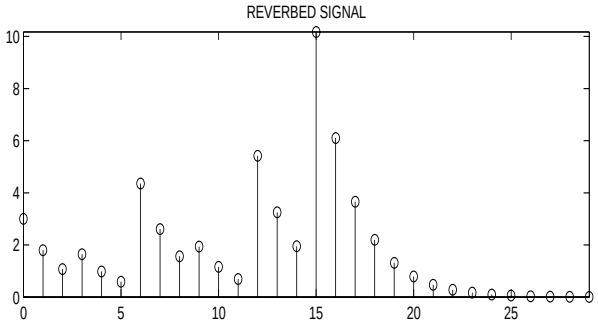


Figure 4.1: Reverbed signal for example.

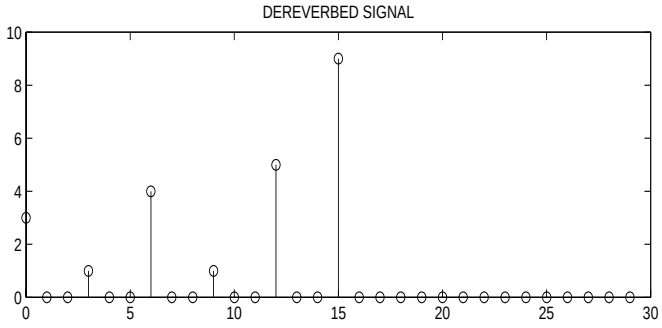


Figure 4.2: Dereverbed signal for example.

Matlab code for this example:

```
X=[3 0 0 1 0 0 4 0 0 1 0 0 5 0 0 9];
H=0.6.^ [0:29];Y=conv(H,X);
Z=filter([1 -0.6],[1],Y);
subplot(211),stem([0:29],Y(1:30))
subplot(212),stem([0:29],Z(1:30))
```

4.7 APPLICATION: Deconvolving Multipath Systems

In radio progagation between a cell phone and the nearest cell phone tower, *multipath* is a problem in cities. Multipath means that there is more than one path between the tower and phone (or vice-versa). Instead of receiving just the signal, the received signal consists of the original signal plus delayed ver-

sions of it, scaled by reflection coefficients from the signal bouncing off of buildings. After sampling, the received signal has the form

$$y[n] = \sum_{i=1}^{L} r_i x[n - d_i] \tag{4.34}$$

- r_i=reflection coefficient off of the i^{th} building. We know $|r_i| < 1$: reflections can't gain strength.
- d_i=time delay from reflection off the i^{th} building. We assume $\{d_i\}$ are integers after sampling.

We assume here that $\{d_i\}$ and $\{r_i\}$ are all known. Adaptive identification methods beyond the scope of these notes are used to determine $\{d_i\}$ and $\{r_i\}$. Note this model of multipath is a MA system in which most of the MA coefficients are zero.

Multipath also arises in sonar signal processing, since sound waves propagating in the ocean take multiple paths from one point to another.

4.7.1 Deconvolution of Minimum-Phase Multipath Systems.

The following example shows how to deconvolve multipath systems, if the system is minimum phase.

Example: Deconvolution of Minimum-Phase Multipath System.

We are given the system

$$y[n] = x[n] - \frac{5}{6}x[n-1] + \frac{1}{6}x[n-2]. \tag{4.35}$$

The goal is to solve the deconvolution problem of reconstructing $x[n]$ from $y[n]$ by computing: (1) the transfer function; (2) the impulse response; and (3) the difference equation for the inverse system.

Solution:

The impulse response is $h[n]=\{\underline{1}, -\frac{5}{6}, \frac{1}{6}\}$. The transfer function is the z-transform of $h[n]$, or

$$H(z) = 1 - \frac{5}{6}z^{-1} + \frac{1}{6}z^{-2} = \frac{z^2 - \frac{5}{6}z + \frac{1}{6}}{z^2}. \tag{4.36}$$

The zeros are $\{\frac{1}{2}, \frac{1}{3}\}$ and the poles are $\{0, 0\}$. Since all of these are inside the unit circle, the system is minimum phase, and since $b_0 = 1 \neq 0$ it has a BIBO-stable and causal inverse system.

The transfer function of the inverse system is

$$G(z) = \frac{1}{H(z)} = \frac{z^2}{z^2 - \frac{5}{6}z + \frac{1}{6}}. \tag{4.37}$$

The partial fraction expansion of $G(z)$ is

$$G(z) = 1 + \frac{3/2}{z - \frac{1}{2}} - \frac{2/3}{z - \frac{1}{3}} \tag{4.38}$$

so the impulse response of the inverse system is

$$g[n] = \delta[n] + \left[\left(\frac{3}{2}\right)\left(\frac{1}{2}\right)^{n-1} - \left(\frac{2}{3}\right)\left(\frac{1}{3}\right)^{n-1} \right] u[n-1]. \tag{4.39}$$

The easiest way to implement the inverse system is using the difference equation

$$x[n] - \frac{5}{6}x[n-1] + \frac{1}{6}x[n-2] = y[n] \tag{4.40}$$

which is just the original difference equation with its input and output interchanged.

4.7.2 Deconvolving Non-Minimum-Phase Multipath Systems

If the original system is not minimum phase, there is no stable and causal inverse system. In continuous time, this is the end of the story-there is no way to reconstruct the input from the output in real time. Non-minimum phase systems arise in:

- Flexible links manipulation in robotics;
- Rudders in ships and ailerons in planes;
- Driving a car backwards;
- Multipath systems, even though $|r_i| < 1$.

However, in discrete-time signal processing, we can reconstruct a *delayed* version of the input from the output, if the original system has no zeros *on* the unit circle. We can do this by taking the inverse *two*-sided z-transform and choosing the ROC so that the resulting inverse system $g[n]$ is stable but non-causal. The non-causal part of $g[n]$ decays rapidly as $n \to -\infty$, so it can be *truncated* to a finite number

N of non-causal terms. Then $g[n–N]$ is causal, and the output is $x[n–N]$, a *delayed* original signal $x[n]$.

The following example shows how to deconvolve non-minimum-phase multipath systems.

Example: Deconvolution of Non-Minimum-Phase Multipath System.

We are given the system

$$y[n] = x[n] - \frac{3}{5}x[n-1] - \frac{18}{25}x[n-2]. \tag{4.41}$$

The goal is to solve the deconvolution problem of reconstructing $x[n]$ from $y[n]$.

Solution:

The impulse response is $h[n]=\{\underline{1}, -\frac{3}{5}, -\frac{18}{25}\}$. The transfer function is the z-transform of $h[n]$, or

$$H(z) = 1 - \frac{3}{5}z^{-1} - \frac{18}{25}z^{-2} = \frac{z^2 - \frac{3}{5}z - \frac{18}{25}}{z^2}. \tag{4.42}$$

The zeros are $\{\frac{6}{5}, -\frac{3}{5}\}$ and the poles are $\{0, 0\}$. Although both reflection coefficients have $|-\frac{3}{5}| < 1$ and $|-\frac{18}{25}| < 1$, the system is not minimum phase. So there is no stable and causal inverse system. In particular, this inverse system is *unstable*:

$$x[n] - \frac{3}{5}x[n-1] - \frac{18}{25}x[n-2] = y[n]. \tag{4.43}$$

But there is a stable and *non*-causal inverse system, which we can find using the *two*-sided z-transform.

The transfer function of the inverse system is

$$G(z) = \frac{1}{H(z)} = \frac{z^2}{z^2 - \frac{3}{5}z - \frac{18}{25}}. \tag{4.44}$$

The partial fraction expansion of $\frac{G(z)}{z}$ is

$$\frac{G(z)}{z} = \frac{2/3}{z - \frac{6}{5}} + \frac{1/3}{z + \frac{3}{5}} \tag{4.45}$$

This is computed in Matlab using

```
[R P]=residue([1 0],[1 -3/5 -18/25])
```

We want the *stable and non-causal* inverse *two*-sided z-transform (after multiplying by z), which is:

$$g[n] = \underbrace{\frac{1}{3}\left(-\frac{3}{5}\right)^n u[n]}_{\text{CAUSAL}} \underbrace{- \frac{2}{3}\left(\frac{6}{5}\right)^n u[-n-1]}_{\text{ANTICAUSAL}} \tag{4.46}$$

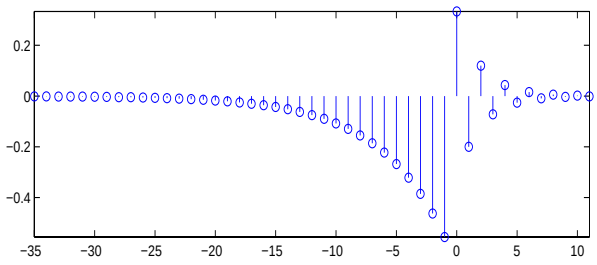


Figure 4.3: Stable but noncausal inverse filter $g[n]$.

```
G=-(2/3)*(6/5).^ [-35:-1];
G=[G (1/3)*(-3/5).^ [0:11]];
subplot(211),stem([-35:11],G)
```

While $g[n]$ is noncausal, $|g[n]| < 0.001$ for $n < -35$. We assume values of $|g[n]| < 0.001$ are negligible. $g[n–35]$ is causal, since $|g[n–35]| < 0.001$ for $n < 0$. Furthermore, $|g[n]| < 0.001$ for $n > 11$. So we can define the *approximate inverse filter* $\tilde{g}[n]$ as

$$\tilde{g}[n] = \begin{cases} g[n-35] & 0 \le n \le 46 \\ 0 & \text{otherwise} \end{cases} \tag{4.47}$$

$\tilde{g}[n]$ is causal, MA, with length 11–(–35)+1=47. And it is a *delayed* inverse system:

$$h[n] * \tilde{g}[n] \approx \delta[n-35]. \tag{4.48}$$

since by time invariance

- $y[n] \rightarrow \boxed{g[n]} \rightarrow x[n]$ implies that
- $y[n] \rightarrow \boxed{g[n–35]} \rightarrow x[n–35]$.

We can, to a very good approximation (neglecting $|g[n]| < 0.001$), recover $x[n]$ *delayed by 35*. At the standard CD sampling rate of 44100 samples/second, this is less than 1 msec., which may be acceptable.

This can be confirmed using the following Matlab program. Omitting the first two and last two values of the computed convolution, which use only part of $h[n]$ and are therefore *end effects*, the output is 0 to roundoff error except at Matlab index 36 (n=35).

```
H=[1 -3/5 -18/25];
G=-(2/3)*(6/5).^ [-35:-1];
G=[G (1/3)*(-3/5).^ [0:11]];
D=conv(H,G);max(abs(D([3:35 37:47])))
```

Chapter 5

Frequency Response of LTI Systems

5.1 LTI System Response to a Complex Exponential

Let $x[n]=e^{j\omega n}$ be the input into a BIBO-stable LTI system with impulse response $h[n]$. The output is

$$\begin{aligned} y[n] &= h[n] * e^{j\omega n} = \sum_{i=-\infty}^{\infty} h[i] e^{j\omega(n-i)} \\ &= e^{j\omega n} \sum_{i=-\infty}^{\infty} h[i] e^{-j\omega i} = H(e^{j\omega}) e^{j\omega n} \end{aligned} \quad (5.1)$$

where $H(e^{j\omega})$ is the **discrete-time frequency response** function, defined as

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n] e^{-j\omega n}. \quad (5.2)$$

$e^{j\omega n} \rightarrow$ LTI $\rightarrow H(e^{j\omega}) e^{j\omega n}$
$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n] e^{-j\omega n}$

Compare the discrete-time and continuous-time frequency response functions alongside each other:

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n] e^{-j\omega n}; H(j\Omega) = \int_{-\infty}^{\infty} h(t) e^{-j\Omega t} dt \quad (5.3)$$

These two functions exist provided that $h[n]$ is absolutely summable and $h(t)$ is absolutely integrable, equivalent to the LTI systems being BIBO stable.

If $h[n]$ is real, $H(e^{j\omega})$ has the same conjugate symmetry properties as $H(j\Omega)$. These properties are:

- $H(e^{j\omega})^*=H(e^{-j\omega})$ (conjugate symmetry). So:
- $|H(e^{j\omega})|=|H(e^{-j\omega})|$ (magnitude is even);
- $\angle H(e^{j\omega})=-\angle H(e^{-j\omega})$ (phase is odd);
- Real$[H(e^{j\omega})]$=Real$[H(e^{-j\omega})]$ (real part is even);
- Imag$[H(e^{j\omega})]$=–Imag$[H(e^{-j\omega})]$ (imag part odd).

These all follow immediately from (5.2).

$H(e^{j\omega})$ has an important additional property:

$H(e^{j\omega})$ is periodic in ω with period 2π.

This is why we write the discrete-time frequency response as $H(e^{j\omega})$, which is explicitly periodic in ω with period 2π (some books do not do this). We have already seen that discrete-time frequency ω is periodic with period 2π, so the periodicity of discrete-time frequency response is a logical corollary.

5.2 LTI System Response to a Sinusoidal Input

Let $x[n]=\cos(\omega n)$ be input into a BIBO-stable LTI system with impulse response $h[n]$. Since $2\cos(\omega n)=e^{j\omega n}+e^{-j\omega n}$, the output can be computed by adding the responses to two complex exponential signals:

$$\begin{aligned} e^{+j\omega n} &\rightarrow H(e^{+j\omega}) e^{+j\omega n} = |H(e^{j\omega})| e^{+j\theta} e^{+j\omega n} \\ e^{-j\omega n} &\rightarrow H(e^{-j\omega}) e^{-j\omega n} = |H(e^{j\omega})| e^{-j\theta} e^{-j\omega n} \\ 2\cos(\omega n) &\rightarrow 2|H|\cos(\omega n + \theta) \end{aligned}$$

where θ =arg$[H(e^{j\omega})]$ and we have used the conjugate symmetry of $H(e^{j\omega})$. Dividing by two gives

$\cos(\omega n) \rightarrow$	LTI	$\rightarrow |H(e^{j\omega})| \cos(\omega n + \arg[H(e^{j\omega})])$
Frequency Response=$H(e^{j\omega})=\sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}$		

Now let the input be a sinusoid with a phase shift:

$$x[n] = A\cos(\omega n + \phi) = A\cos(\omega(n + \phi/\omega)). \quad (5.4)$$

Then, by time-invariance, the output will be:

$$\begin{aligned} y[n] &= A|H(e^{j\omega})|\cos(\omega(n+\phi/\omega)+\theta) \\ &= A|H(e^{j\omega})|\cos(\omega n+\phi+\theta) \\ \text{for} \quad \theta &= \arg[H(e^{j\omega})] \end{aligned} \quad (5.5)$$

since a phase shift for a pure sinusoid is equivalent to a time delay. The same result holds in continuous time, where it is often called "sine-in-sine-out."

5.3 Computing and Using Frequency Response Functions

We illustrate this with a series of examples.

Example: LTI system response to a sinusoid.
An LTI system has impulse response $(\frac{1}{2})^n u[n]$. Compute the response to input $x[n] = \cos(\frac{\pi}{3}n)$.

Solution:
The frequency response function is

$$\begin{aligned} H(e^{j\omega}) &= \sum_{n=0}^{\infty}(1/2)^n e^{-j\omega n} = \sum_{n=0}^{\infty}[(1/2)e^{-j\omega}]^n \\ &= \frac{1}{1-(1/2)e^{-j\omega}} = \frac{e^{j\omega}}{e^{j\omega}-(1/2)}. \end{aligned} \quad (5.6)$$

Note $|\frac{1}{2}e^{-j\omega}| = \frac{1}{2} < 1$ so the infinite series converges. This is a result of the system being BIBO stable.

The magnitude response (gain) $|H(e^{j\omega})|$ and phase $\arg(H(e^{j\omega}))$ are plotted in the next figure. Note that the system increases low-frequency components (the DC gain is $H(e^{j0}) = 2$) and reduces high-frequency components ($H(e^{j\pi}) = \frac{2}{3}$).

The Matlab code used to generate this:

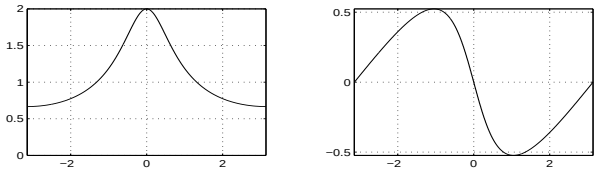


Figure 5.1: $|H(e^{j\omega})|$ (left) and arg$[H(e^{j\omega})]$ (right).

```
W=linspace(-pi,pi,1000);
H=exp(j*W)./(exp(j*W)-0.5);
subplot(221),plot(W,abs(H)),
axis([-pi pi 0 2])
subplot(222),plot(W,angle(H)),axis tight
```

Inserting $\omega = \frac{\pi}{3}$ into $H(e^{j\omega})$ gives

$$H(e^{j\pi/3}) = \frac{e^{j\pi/3}}{e^{j\pi/3}-(1/2)} = \frac{e^{j\pi/3}}{j\sqrt{3}/2} = 1.155e^{-j\pi/6}. \quad (5.7)$$

So the output is $y[n] = 1.155\cos(\frac{\pi}{3}n - \frac{\pi}{6})$.

Example: Frequency response of the two-point averager.
Compute $H(e^{j\omega})$ for $y[n]$=$\frac{1}{2}$($x[n]$+$x[n$–$1]$).
The system averages the two most recent inputs.

Solution:
The impulse response $h[n]$=$y[n]$ when $x[n]$=$\delta[n]$:

$$h[n] = \frac{1}{2}(\delta[n] + \delta[n-1]) = \{\underline{\frac{1}{2}}, \frac{1}{2}\}. \quad (5.8)$$

The frequency response function $H(e^{j\omega})$ is then

$$\begin{aligned} H(e^{j\omega}) &= h[0]e^{-j\omega 0} + h[1]e^{-j\omega 1} \\ &= \frac{1}{2}(1+e^{-j\omega}) \\ &= \cos(\omega/2)e^{-j\omega/2} \end{aligned} \quad (5.9)$$

The transformation from the second to the final equation, sometimes called "phase-splitting," comes from

$$\begin{aligned} \cos(\omega/2) &= \frac{1}{2}(e^{j\omega/2} + e^{-j\omega/2}) \\ \cos(\omega/2)e^{-j\omega/2} &= \frac{1}{2}(1+e^{-j\omega}). \end{aligned} \quad (5.10)$$

Over the single period $-\pi < \omega \leq \pi$ we have

- Gain $|H(e^{j\omega})|$=$\cos(\frac{\omega}{2}) \geq 0$.
- Phase $\arg[H(e^{j\omega})] = -\omega/2$.

These are plotted in the next figure. It is evident that this system has little effect on low-frequency sinusoids but greatly reduces high-frequency ($\omega \approx \pi$) sinusoids. This system acts as a *low-pass filter* that *smoothes* the input $x[n]$ by reducing the high-frequency components of $x[n]$ that cause fast variation of $x[n]$. This agrees with our intuition about averaging, but puts it on a more solid mathematical basis.

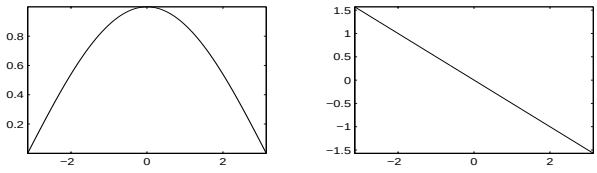


Figure 5.2: $|H(e^{j\omega})|$ (left) and $\arg[H(e^{j\omega})]$ (right) of the two-point averager.

```
W=linspace(-pi,pi,1000);A=cos(W/2);P=-W/2;
subplot(221),plot(W,A),axis tight
subplot(222),plot(W,P),axis tight
```

Note that the gain is an even function of ω, and the phase is an odd function of ω, as expected, due to conjugate symmetry resulting from the real $h[n]$.

Example: Frequency response of the two-point differencer.
Compute $H(e^{j\omega})$ for $y[n] = \frac{1}{2}(x[n] - x[n-1])$. The output of this system is half the difference of the two most recent inputs.

Solution:
The impulse response $h[n]$=$y[n]$ when $x[n]$=$\delta[n]$:

$$h[n] = \frac{1}{2}(\delta[n] - \delta[n-1]) = \{\underline{\frac{1}{2}}, -\frac{1}{2}\}. \qquad (5.11)$$

The frequency response function $H(e^{j\omega})$ is then

$$\begin{aligned} H(e^{j\omega}) &= h[0]e^{-j\omega 0} + h[1]e^{-j\omega 1} \\ &= \frac{1}{2}(1 - e^{-j\omega}) \\ &= \sin(\omega/2)je^{-j\omega/2} \end{aligned} \qquad (5.12)$$

where phase-splitting now gives

$$\begin{aligned} \sin(\omega/2) &= \frac{1}{2j}(e^{j\omega/2} - e^{-j\omega/2}) \\ \sin(\omega/2)je^{-j\omega/2} &= \frac{1}{2}(1 - e^{-j\omega}). \end{aligned} \qquad (5.13)$$

$\sin(\omega/2)je^{-j\omega/2}$ is a continuous function of ω. However, the phase is *not* a continuous function of ω, since the sign of $\sin(\omega/2)$ changes at frequency ω=0.

Over the single period $-\pi < \omega \leq \pi$ we have

- Gain $|H(e^{j\omega})|$=$|\sin(\frac{\omega}{2})|$ is even, *not* $\sin(\frac{\omega}{2})$, which is negative for $-\pi < \omega < 0$.
- Phase $\arg[H(e^{j\omega})] = \frac{\pi}{2}\text{sign}[\omega] - \frac{\omega}{2}$ is odd, since $\pm j = e^{\pm j\frac{\pi}{2}}$. Note the jump at ω=0.

These are plotted in the next figure.

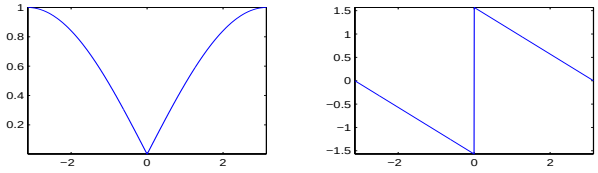


Figure 5.3: $|H(e^{j\omega})|$ (left) and $\arg[H(e^{j\omega})]$ (right) of the two-point differencer.

```
W=linspace(-pi,pi,1000);H=(1-exp(-j*W))/2;
subplot(221),plot(W,abs(H)),axis tight
subplot(222),plot(W,angle(H)),axis tight
```

The system acts as a crude *high*-pass filter.

The phase jumps from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ at frequency ω=0, since the magnitude is zero at ω=0; the phase can be discontinuous when the magnitude is near zero. The phase is *undefined* when the magnitude is zero.

5.4 Frequency Response Function of an ARMA Difference Equation

If we know the impulse response $h[n]$ of an LTI system, we can compute its frequency response function

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}. \qquad (5.14)$$

But if we are given an ARMA difference equation

$$\sum_{i=0}^{N} a_i y[n-i] = \sum_{i=0}^{M} b_i x[n-i] \tag{5.15}$$

of the LTI system, then we can immediately read off $H(e^{j\omega})$. Let $x[n]$=$e^{j\omega n}$. Since the ARMA difference equation describes an LTI system, we already know the output $y[n]$=$H(e^{j\omega})e^{j\omega n}$. Substituting these,

$$\sum_{i=0}^{N} a_i H(e^{j\omega})e^{j\omega(n-i)} = \sum_{i=0}^{M} b_i e^{j\omega(n-i)}. \tag{5.16}$$

Since $e^{j\omega n} \neq 0$ for all n and ω, we can cancel this factor and solve for $H(e^{j\omega})$. This gives

$$H(e^{j\omega}) = \sum_{i=0}^{M} b_i e^{-j\omega i} / \sum_{i=0}^{N} a_i e^{-j\omega i}. \tag{5.17}$$

For the special case of an MA system, $h[n]$=b_n, so the denominator of this expression is just one, and the numerator is the definition of $H(e^{j\omega})$.

Example: $H(e^{j\omega})$ from ARMA.
Compute $H(e^{j\omega})$ for

$$y[n]+2y[n-1]+3y[n-2] = 4x[n]+3x[n-1] \tag{5.18}$$

and compute output $y[n]$ if the input is

$$x[n] = 6 + 2\sqrt{2}\cos(\frac{\pi}{2}n - 48^o). \tag{5.19}$$

Solution:
Plugging into the above formula,

$$H(e^{j\omega}) = \frac{4+3e^{-j\omega}}{1+2e^{-j\omega}+3e^{-2j\omega}}. \tag{5.20}$$

Inserting ω=0 and noting that e^{-j0}=1 gives

$$H(e^{j0}) = \frac{4+3}{1+2+3} = \frac{7}{6}. \tag{5.21}$$

Inserting ω=$\frac{\pi}{2}$ and noting $e^{-j\frac{\pi}{2}}$=$-j$ gives

$$H(e^{j\frac{\pi}{2}}) = \frac{4-j3}{1-j2-3} = \frac{5e^{-j37^o}}{2\sqrt{2}e^{-j135^o}} = \frac{5}{2\sqrt{2}}e^{j98^o}. \tag{5.22}$$

Since the system is LTI, we can compute the response to each term of the input separately. Recalling that a constant input is a sinusoid with frequency ω=0, we form a table, in which X and Y are the phasors for $x[n]$ and $y[n]$. Recall that phasors are defined as

$$x[n] = A\cos(\omega n + \theta) \Leftrightarrow \text{Phasor} \quad X = Ae^{j\theta}. \tag{5.23}$$

ω :	0	$\pi/2$
X:	6	$2\sqrt{2}e^{-j48^o}$
$H(e^{j\omega})$:	$\frac{7}{6}e^{j0}$	$\frac{5}{2\sqrt{2}}e^{j98^o}$
Y=HX:	7	$5e^{j50^o}$

The response to the given $x[n]$ is therefore

$$y[n] = 7 + 5\cos(\frac{\pi}{2}n + 50^o). \tag{5.24}$$

5.5 From Frequency Response to Other Descriptions

The frequency response function $H(e^{j\omega})$ is computed by setting z=$e^{j\omega}$ in $H(z)$. This is analogous to, in continuous-time, obtaining $H(j\Omega)$ by substituting s=$j\Omega$ into the Laplace transform $H(s)$ of $h(t)$.

To go from $H(e^{j\omega})$ to $H(z)$, and then to all other LTI system descriptions, we must write $H(e^{j\omega})$ as a function of $e^{j\omega}$, not just ω. We illustrate this with a pair of examples.

Example: Frequency response to other LTI system descriptions.
An (unstable) LTI system has $H(e^{j\omega})$=$j\tan(\omega)$. Find: (1) transfer function; (2) its poles and zeros; (3) ARMA difference equation.

Solution:

$$H(e^{j\omega}) = j\tan(\omega) = \frac{2j\sin(\omega)}{2\cos(\omega)} = \frac{e^{j\omega}-e^{-j\omega}}{e^{j\omega}+e^{-j\omega}} \tag{5.25}$$

Now we can substitute $e^{j\omega}$=z. This gives

$$H(z) = \frac{Y(z)}{X(z)} = \frac{z-z^{-1}}{z+z^{-1}} = \frac{1-z^{-2}}{1+z^{-2}} = \frac{z^2-1}{z^2+1}. \tag{5.26}$$

The poles are $\{\pm j\}$ from z^2+1=0.
The zeros are $\{\pm 1\}$ from z^2–1=0.

Cross-multiplying $H(z)$=$Y(z)/X(z)$ gives

$$Y(z)(1+z^{-2}) = X(z)(1-z^{-2})$$
$$y[n]+y[n-2] = x[n]-x[n-2]. \quad (5.27)$$

Note the poles at $\{\pm j\}$ make the system unstable. The frequency response should blow up at ω=$\pm\frac{\pi}{2}$, since $\pm j$=$e^{\pm j\frac{\pi}{2}}$, as indeed it does. The system is, however, *marginally* stable.

Example: Frequency response to other LTI system descriptions.

An LTI system has the frequency response function

$$\frac{[\cos(2\omega)-\cos(\omega)]+j[\sin(2\omega)-\sin(\omega)]}{[\cos(2\omega)-5\cos(\omega)+6]+j[\sin(2\omega)-5\sin(\omega)]}. \quad (5.28)$$

Find: (1) transfer function $H(z)$; (2) poles and zeros; (3) the *stable* impulse response $h[n]$.

Solution:

First, we compute the transfer function $H(z)$ by writing (5.28) as an explicit function of $e^{j\omega}$. Then we can substitute z=$e^{j\omega}$ (compare to s=$j\Omega$ in continuous time) to obtain $H(z)$.

To write (5.28) as an explicit function of $e^{j\omega}$, we substitute the two identities

$$2\cos(\omega) = e^{j\omega}+e^{-j\omega}; 2j\sin(\omega) = e^{j\omega}-e^{-j\omega} \quad (5.29)$$

into (5.28). This gives, after some algebra,

$$H(e^{j\omega}) = \frac{e^{j2\omega}-e^{j\omega}}{e^{j2\omega}-5e^{j\omega}+6}. \quad (5.30)$$

Now we can substitute z=$e^{j\omega}$ to get

$$H(z) = \frac{z^2-z}{z^2-5z+6}. \quad (5.31)$$

Then, having obtained the transfer function $H(z)$,
The zeros are $\{0,1\}$ from z^2–z=0.
The poles are $\{2,3\}$ from z^2–5z+6=0.

The *causal* impulse response is unstable: $|2|,|3|>1$. But the *anti*-causal impulse response is stable.

The impulse response is determined by computing the partial fraction expansion of $\frac{H(z)}{z}$. This gives

$$H(z) = \frac{z^2-z}{z^2-5z+6} = 2\frac{z}{z-3}-\frac{z}{z-2} \quad (5.32)$$

and then reading off the *anti*-causal impulse response

$$h[n] = -2(3)^n u[-n-1]+(2)^n u[-n-1]. \quad (5.33)$$

5.6 Effect of Poles and Zeros on Frequency Response Function Gain $|H(e^{j\omega})|$

As in continuous time, knowledge of the poles and zeros of an LTI system gives considerable insight into the form of its gain function $|H(e^{j\omega})|$. The difference is that in discrete time, the locations of poles and zeros relative to the *unit circle* $\{|z|=1\}$ is significant, not the locations relative to the imaginary axis.

Let an LTI system has zeros $\{z_i, i=1\ldots M\}$ and poles $\{p_i, i=1\ldots N\}$. Then the transfer function is, for some constant C,

$$H(z) = C\frac{(z-z_1)\ldots(z-z_M)}{(z-p_1)\ldots(z-p_N)}. \quad (5.34)$$

Substituting z=$e^{j\omega}$ and taking the magnitude gives

$$|H(e^{j\omega})| = |C|\frac{|e^{j\omega}-z_1|\ldots|e^{j\omega}-z_M|}{|e^{j\omega}-p_1|\ldots|e^{j\omega}-p_N|}. \quad (5.35)$$

Suppose one of the zeros (say z_1) is on the unit circle $|z|$=1. Then z_1=$e^{j\omega_1}$ for some ω_1 and $|H(e^{j\omega_1})|$=0, since the factor in the numerator of $|H(e^{j\omega})|$ is

$$|e^{j\omega}-z_1| = |e^{j\omega}-e^{j\omega_1}|$$
$$= 0 \text{ at } \omega=\omega_1 \quad (5.36)$$

Suppose one of the poles (say p_2) is near the unit circle $|z|$=1. Let p_2=$0.999e^{j\omega_2}$ for some ω_2. Then the factor in the denominator of $|H(e^{j\omega})|$ is

$$|e^{j\omega}-p_2| = |e^{j\omega}-0.999e^{j\omega_2}|$$
$$= 0.001, \text{ at } \omega=\omega_2 \quad (5.37)$$

$|H(e^{j\omega_2})|$ is very large, since it includes a factor $\frac{1}{0.001}$.

If there is a pole *on* the unit circle, say p_2=$e^{j\omega_2}$, then $|H(e^{j\omega_2})|\to\infty$ and the system is not BIBO stable. Recall that all poles must be *not on* the unit circle for an LTI system to be BIBO stable, just as a

continuous-time LTI system must have all of its poles in the left half-plane for the system to be stable.

We can summarize these two results as follows:

> If there is a zero near the unit circle at $z = e^{j\omega_1}$ then there is a dip in gain $|H(e^{j\omega})|$ at $\omega = \omega_1$.
> If there is a pole near the unit circle at $z = e^{j\omega_2}$ then there is a peak in gain $|H(e^{j\omega})|$ at $\omega = \omega_2$.

Example: Poles and zeros effect on gain.
An LTI system has
• Zeros at $\{e^{\pm j\pi/4}, e^{\pm j\pi/2}, e^{j\pi}\}$.
• Poles at $\{0.8e^{\pm j\pi/3}, 0.8e^{\pm j2\pi/3}\}$.
Sketch on the "back of an envelope" a gain plot.

Solution:
• $|H(e^{j\omega})|$ will dip to zero at $\omega = \pm\frac{\pi}{4}, \pm\frac{\pi}{2}, \pi$.
• $|H(e^{j\omega})|$ will have peaks at $\omega = \pm\frac{\pi}{3}, \pm\frac{2\pi}{3}$.
• The actual $|H(e^{j\omega})|$ is plotted in the next figure.

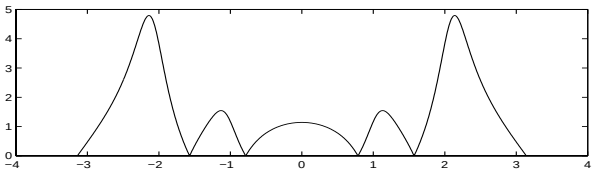


Figure 5.4: $|H(e^{j\omega})|$ for Pole-Zero Example.

Matlab code for this example:

```
Z=exp(j*pi/4*[1 -1 2 -2 4]);
P=0.8*exp(j*pi/3*[1 -1 2 -2]);
N=poly(Z);D=poly(P);
W=linspace(-pi,pi,100);
H=polyval(N,exp(j*W));
H=H./polyval(D,exp(j*W));
subplot(211),plot(W,abs(H))
```

Example: Eight-point moving averager.
Find $|H(e^{j\omega})|$ for the eight-point moving averager

$$y[n] = \frac{1}{8}(x[n] + x[n-1] + \ldots + x[n-7]) \quad (5.38)$$

which averages the eight most recent input values.

Solution:

We can read off the impulse response $h[n]$ as

$$h[n] = \frac{1}{8}\{\underline{1}, 1, 1, 1, 1, 1, 1, 1\}. \quad (5.39)$$

The transfer function is then

$$\begin{aligned} H(z) &= \frac{1}{8}(1 + z^{-1} + z^{-2} + \ldots + z^{-7}) \quad (5.40)\\ &= \frac{z^7 + z^6 + \ldots + z + 1}{8z^7} = \frac{z^8 - 1}{8z^7(z-1)}. \end{aligned}$$

$H(z)$ has zeros $\{e^{j2\pi k/8}, k{=}1\ldots7\}$ and seven poles at the origin $\{0\}$. The zero and pole at $z{=}1$ cancel. So $|H(e^{j\omega})|$ will dip to zero at $\omega{=}\{\frac{2\pi k}{8}, k{=}1\ldots7\}$. These frequencies are evenly spaced, except that $\omega{=}0$ is missing, so $|H(e^{j\omega})|$ should look like a rabbit fence with a higher bump at $\omega{=}0$.

Computing the frequency response function $H(e^{j\omega})$ requires some additional algebra. Setting $z{=}e^{j\omega}$ in the last expression for $H(z)$ and using the same "phase-splitting" trick we used previously,

$$\begin{aligned} H(e^{j\omega}) &= \frac{e^{j8\omega} - 1}{8e^{j7\omega}(e^{j\omega} - 1)}\\ &= \frac{e^{j4\omega} - e^{-j4\omega}}{e^{j\omega/2} - e^{-j\omega/2}} \frac{e^{j4\omega}}{8e^{j7.5\omega}} \quad (5.41) \end{aligned}$$

Using the definition of sine, this simplifies to

$$H(e^{j\omega}) = \frac{2j\sin(4\omega)}{2j\sin(\omega/2)} \frac{e^{-j3.5\omega}}{8} \quad (5.42)$$

The magnitude of this expression is

$$|H(e^{j\omega})| = \frac{|\sin(4\omega)|}{8|\sin(\omega/2)|}. \quad (5.43)$$

Students who have taken an electromagnetics class will recognize that this is the formula for the directional antenna gain of a linear array of radiators.

The eight-point moving averager, like the two-point averager, is a low-pass filter. But the cutoff frequency $\omega{=}\frac{2\pi}{8}$ is more evident.

$|H(e^{j\omega})|$ is plotted in the next figure. It does look like a rabbit fence with a bump at $\omega{=}0$.

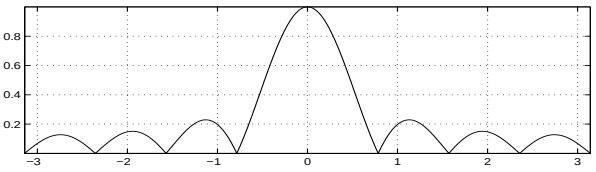


Figure 5.5: $|H(e^{j\omega})|$ for eight-point moving averager.

5.7 APPLICATION: Notch Filters: Removing Sinusoidal Interference

When measuring signals in a lab, interference in the form of an additive 60 Hertz sinusoid is often present. The source of this interference is the electrical wires in the wall of the lab. These wires carry 120 volts rms at 60 Hertz, and radiate electromagnetic energy that is picked up by the leads of the measuring device. While proper grounding can reduce this problem, we would still like to design a simple discrete-time system that eliminates the 60 Hertz interference while leaving the signal otherwise unaffected. We say we would like to filter the signal to eliminate *only* 60 Hertz and leave other frequencies.

If the spectrum of the signal is negligible near and above 60 Hertz, we could use a low-pass filter. But most signals have frequency components well above 60 Hertz. What we need is a system that passes below and above 60 Hertz, but rejects 60 Hertz. Since the gain of such a system has a *notch* (see the gain plots below), it is called a *notch filter*.

5.7.1 Derivation of Notch Filter

We wish to eliminate an interfering sinusoid of frequency F Hertz from a signal $x(t)$, while leaving other frequencies of $x(t)$ (almost) unaffected. We sample the signal every T sec. (a rate $\frac{1}{T}$ samples per sec.).

The interfering sinusoid, after sampling, is at *discrete-time* frequency $\omega_o{=}2\pi FT$. To see this, recall sampling every T sec. amounts to setting $t{=}nT$ for integers n. Now define $x[n]{=}x(nT)$ using

$$\begin{aligned} x(t) &= A\cos(2\pi Ft+\theta) \\ x[n] &= A\cos(2\pi F(nT)+\theta) \\ &= A\cos([2\pi FT]n+\theta) \\ &= A\cos(\omega_o n+\theta). \end{aligned} \tag{5.44}$$

This sinusoid can be written as the sum of two discrete-time complex exponential signals as

$$A\cos(\omega_o n+\theta) = \frac{Ae^{j\theta}}{2}e^{j\omega_o n} + \frac{Ae^{-j\theta}}{2}e^{-j\omega_o n} \tag{5.45}$$

Hence the filter should have two zeros at $e^{\pm j\omega_o}$ to eliminate the interfering sinusoid. It should also have two poles at $ae^{\pm j\omega_o}$, for some value of a is slightly less than one, so that the filter is stable, and leaves other frequencies unaltered. The transfer function is

$$\begin{aligned} H(z) &= \frac{(z-e^{j\omega_o})(z-e^{-j\omega_o})}{(z-ae^{j\omega_o})(z-ae^{-j\omega_o})} \\ &= \frac{z^2-2\cos(\omega_o)z+1}{z^2-2a\cos(\omega_o)z+a^2}. \end{aligned} \tag{5.46}$$

The impulse response will have the form

$$h[n] = \delta[n] + Ca^n\cos(\omega_o n+\theta)u[n-1] \tag{5.47}$$

for some constants C and θ which are very complicated functions of a and ω_o. There is a tradeoff between resolution (sharpness of the dip in gain at ω_o) and duration of $h[n]$: the closer a is to one, the sharper the dip, but the slower the decay of $h[n]$.

The notch filter is actually implemented using an ARMA difference equation, which we derive from

$$H(z) = \frac{Y(z)}{X(z)} = \frac{z^2-2\cos(\omega_o)z+1}{z^2-2a\cos(\omega_o)z+a^2}\left[\frac{z^{-2}}{z^{-2}}\right] \tag{5.48}$$

Cross-multiplying gives

$$\begin{aligned} & Y(z)[1-2a\cos(\omega_o)z^{-1}+a^2z^{-2}] \\ = \;& X(z)[1-2\cos(\omega_o)z^{-1}+z^{-2}]. \end{aligned} \tag{5.49}$$

An inverse z-transform gives the difference equation

$$\begin{aligned} & y[n]-2a\cos(\omega_o)y[n-1]+a^2y[n-2] \\ = \;& x[n]-2\cos(\omega_o)x[n-1]+x[n-2]. \end{aligned} \tag{5.50}$$

so the notch filter is implemented as a linear combination of the present input and two most recent inputs and outputs. Note that the ARMA difference equation coefficients can be read off of $H(z)$. We do this in the two examples below.

5.7.2 Notch Filter Examples

Example: Simple notch filter.

Find the ARMA difference equation for a notch filter that rejects a 250 Hertz sinusoid. The sampling rate is 1000 samples per sec. Use a=0.99.

Solution:

The discrete-time frequency ω_o to be rejected is $\omega_o = 2\pi\frac{250}{1000} = \frac{\pi}{2}$. The middle coefficient in the notch filter is then $-2\cos(\omega_o) = -2\cos(2\pi\frac{250}{1000}) = 0$. Inserting a=0.99 and a^2=0.98 gives the difference equation

$$y[n] - 0.98y[n-2] = x[n] - x[n-2]. \tag{5.51}$$

Example: Implemented notch filter.

We are given the sum of 2 sinusoidal signals $\sin(2\pi 125t)+\sin(2\pi 200t)$. We use a DSP system with a sampling rate of 1000 samples per sec. Design a notch filter with a=0.9 that keeps the 125 Hertz sinusoid and rejects the 200 Hertz sinusoid.

Solution:

The discrete-time frequency ω_o to be rejected is $\omega_o = 2\pi\frac{200}{1000} = 0.4\pi$. The transfer function is then

$$\begin{aligned} H(z) &= \frac{(z - e^{j0.4\pi})(z - e^{-j0.4\pi})}{(z - 0.9e^{j0.4\pi})(z - 0.9e^{-j0.4\pi})} \\ &= \frac{z^2 - 2\cos(0.4\pi)z + 1}{z^2 - 2(0.9)\cos(0.4\pi)z + (0.9)^2} \\ &= \frac{z^2 - 0.62z + 1}{z^2 - 0.56z + 0.81}. \end{aligned} \tag{5.52}$$

The notch filter is implemented using an ARMA difference equation, read off of $H(z)$ as

$$\begin{aligned} &y[n] - 0.56y[n-1] + 0.81y[n-2] \\ &= x[n] - 0.62x[n-1] + x[n-2]. \end{aligned} \tag{5.53}$$

The original (left) and filtered (right) signals are shown in the next figure. One sinusoid remains.

The impulse response of the notch filter decays as $(0.9)^n$, since a=0.9. Since $(0.9)^{30} \approx 0.0424$, the impulse response is judged to be negligible after roughly 30 time samples, so the duration of the transient response is roughly 30 time samples. The output is actually plotted starting at time sample n=31, and it is visually apparent that the output of the notch filter has reached steady state. This was used to determine the rough duration of the transient response.

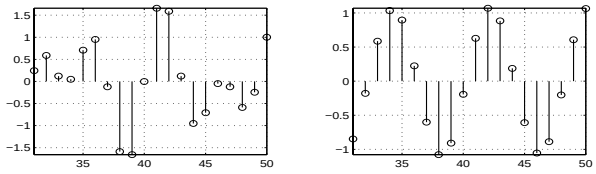


Figure 5.6: Time waveforms of the original (left) and filtered (right) signals for notch filter.

The two-sided gain (left) and pole-zero diagram (right) are shown in the next figure. Note that the gains at ω=0 and ω=π are slightly larger than one:

$$\begin{aligned} H(e^{j0}) &= \frac{1 - 0.62 + 1}{1 - 0.56 + 0.81} = 1.104. \\ H(e^{j\pi}) &= \frac{1 + 0.62 + 1}{1 + 0.56 + 0.81} = 1.105. \end{aligned} \tag{5.54}$$

This can easily be corrected by dividing $H(z)$ by 1.104 to bring the dc gain down to one. We use

$$H(z) = \frac{1}{1.104}\frac{z^2 - 0.62z + 1}{z^2 - 0.56z + 0.81}. \tag{5.55}$$

The ARMA difference equation is now

$$\begin{aligned} &y[n] - 0.56y[n-1] + 0.81y[n-2] \\ &= 0.905x[n] - 0.56x[n-1] + 0.905x[n-2]. \end{aligned} \tag{5.56}$$

where $\frac{1}{1.105}$=0.905 and $\frac{0.62}{1.105}$=0.56.

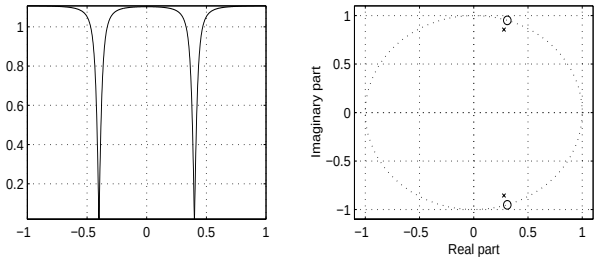


Figure 5.7: Two-sided gain (left) and pole-zero diagram (right) for notch filter.

The Matlab code for this example:

```
W1=2*pi*125/1000;N=1:50;
W2=2*pi*200/1000;M=[31:50];
X=sin(W1*N)+sin(W2*N);a=0.9;
B=[1 -2*cos(W2) 1];
A=[1 -2*a*cos(W2) a*a];
Y=filter(B,A,X);
subplot(221),stem(M,X(M))
subplot(222),stem(M,Y(M))
W=linspace(-1,1,1000);
Z=exp(j*pi*W);
H=polyval(B,Z)./polyval(A,Z);
subplot(223),plot(W,abs(H))
subplot(224),zplane(B,A)
```

Example: Real-world notch filter.

An actual trumpet playing note B, which has a period of $\frac{1}{494}$ sec., had a interfering 1200-Hertz sinusoid added to it. The trumpet-plus-sinusoid signal was sampled at the standard CD sampling rate of 44100 samples per sec. Design and implement a discrete-time notch filter to eliminate the interfering sinusoid. Use $a = 0.99$.

Solution:

The sampling interval is $\frac{1}{44100}$ sec. The discrete-time frequency to be rejected is $2\pi\frac{1200}{44100}$=0.171. The harmonics of the trumpet signal are at frequencies $\{494k, k$=1,2...$\}$ Hertz. The corresponding discrete-time frequencies are $\frac{2\pi 494k}{44100}$=$0.0704k$. So $H(z)$ is

$$\begin{aligned} H(z) &= \frac{(z-e^{j0.171})(z-e^{-j0.171})}{(z-0.99e^{j0.171})(z-0.99e^{-j0.171})} \\ &= \frac{z^2-2\cos(0.171)z+1}{z^2-2(0.99)\cos(0.171)z+(0.99)^2} \\ &= \frac{z^2-1.97z+1}{z^2-1.95z+0.98}. \end{aligned} \quad (5.57)$$

The notch filter is implemented using an ARMA difference equation, read off of $H(z)$ as

$$\begin{aligned} &y[n]-1.95y[n-1]+0.98y[n-2] \\ &= x[n]-1.97x[n-1]+x[n-2]. \end{aligned} \quad (5.58)$$

The frequency response of the discrete-time notch filter is plotted, in red, in the next figure. The spectrum of the sampled trumpet-plus-sinusoid signal is also plotted, in blue, in the next figure. One-sided plots ($\omega \geq 0$) have been used for both, so more detail can be shown. The trumpet spectrum has been multiplied by five to make it easier to see on the plot. Since the trumpet-plus-sinusoid signal has been sampled, its spectrum is plotted vs. discrete-time frequency ω. Although discrete-time spectra are periodic in ω with period 2π, the trumpet-plus-noise spectrum is plotted only for $0 \leq \omega \leq 1$, since the trumpet-plus-sinusoid spectrum is zero for $1 \leq |\omega| \leq \pi$. It is clear that the notch filter will reject the interfering discrete-time sinusoid having frequency ω_o=0.171.

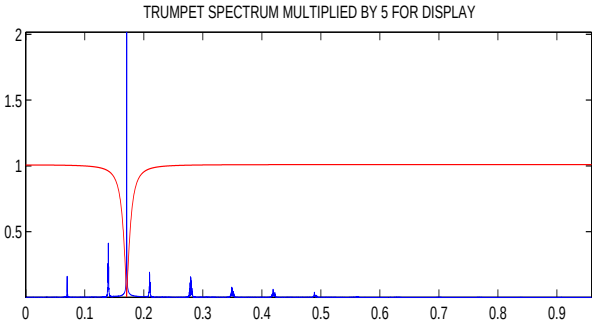


Figure 5.8: Notch filter gain (red) and trumpet spectrum (blue) for notch filter. The trumpet spectrum has been multiplied by five to make it easier to see.

The time waveform of the trumpet-plus-sinusoid signal is plotted in the next figure. The interfering sinusoid is overwhelming the trumpet signal.

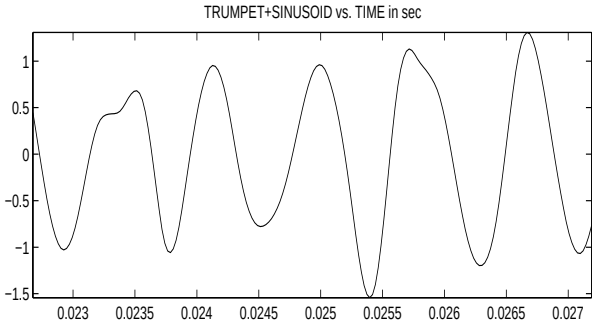


Figure 5.9: Time waveform of trumpet-plus-sinusoid for notch filter example.

The time waveform of the notch-filtered trumpet-plus-sinusoid signal is plotted in the next figure. The filtered signal clearly is similar to the trumpet waveform in the first chapter on discrete-time signals.

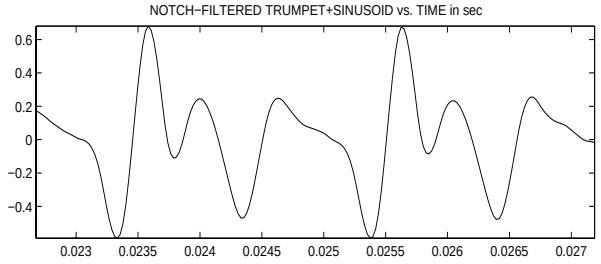


Figure 5.10: Time waveform of notch-filtered trumpet-plus-sinusoid for notch filter example. This is similar to the original trumpet signal.

Strictly speaking, both sampled signals should be plotted using stem plots and plotted against discrete time n. For ease of display, both sampled signals are plotted as continuous waveforms, and plotted against actual time. Both plots use 200 discrete points.

The impulse response of the notch filter decays as $(0.99)^n$, since a=0.99. Since $(0.99)^{315} \approx 0.0422$, the impulse response is judged to be negligible after roughly 315 time samples, so the duration of the transient response is roughly 315 time samples. At a sampling rate of 44100 samples per sec., 315 samples is $\frac{315}{44100}$=0.007 sec., so the duration of the transient response is 0.007 sec., not aurally noticeable.

The Matlab code for this example:

```
load trumpet.mat;S=44100;N=length(X);
soundsc(X,S),pause,W0=2*pi*(1200/S);
X=X+cos(W0*[1:N]);a=0.99;
B=[1 -2*cos(W0) 1];
A=[1 -2*a*cos(W0) a*a];
F=[0:4999]*S/N;W=2*pi/N*[0:4999];
EW=exp(j*W);FX=abs(fft(X));
H=polyval(B,EW)./polyval(A,EW);
plot(W,FX(1:5000)/N*5,W,abs(H),'r')
Y=filter(B,A,X);NN=[1000:1199];T=NN/S;
subplot(211),plot(T,X(NN))
subplot(212),plot(T,Y(NN))
soundsc(X,S),pause,soundsc(Y,S)
```

Example: Design notch filter to meet specs. Design a notch filter that fulfills these specs:

- It rejects 1000 Hertz;
- Gains at 900 and 1100 Hertz are both 0.90;
- Gains at frequencies outside the frequency band 900< f <1100 Hertz exceed 0.90;
- Gain at 0 Hz (dc) is 1.00.
- Sampling rate is 6000 samples per sec.

Solution:
1000 Hertz sampled at 6000 samples per sec. is

$$\omega_o = 2\pi\frac{1000}{6000} = \frac{\pi}{3}. \tag{5.59}$$

The transfer function is, for some constant K,

$$\begin{aligned} H(z) &= K\frac{(z-e^{j\pi/3})(z-e^{-j\pi/3})}{(z-ae^{j\pi/3})(z-ae^{-j\pi/3})} \\ &= K\frac{z^2 - z(e^{j\pi/3}+e^{-j\pi/3}) + e^{j\pi/3}e^{-j\pi/3}}{z^2 - za(e^{j\pi/3}+e^{-j\pi/3}) + a^2e^{j\pi/3}e^{-j\pi/3}} \\ &= K\frac{z^2 - 2z\cos(\pi/3) + 1}{z^2 - 2az\cos(\pi/3) + a^2} \\ &= K\frac{z^2 - z + 1}{z^2 - az + a^2}. \end{aligned} \tag{5.60}$$

The frequency response of the filter is obtained by setting $z = e^{j\omega}$ in $H(z)$. This gives

$$H(e^{j\omega}) = K\frac{e^{j2\omega} - e^{j\omega} + 1}{e^{j2\omega} - ae^{j\omega} + a^2}. \tag{5.61}$$

The constant K is determined by the spec that the dc frequency response $H(e^{j0}) = 1$. This requires

$$1 = H(e^{j0}) = K\frac{1-1+1}{1-a+a^2}. \tag{5.62}$$

Solving for K in terms of a gives

$$K = 1 - a + a^2. \tag{5.63}$$

The gain is the magnitude of the frequency response:

$$\begin{aligned} |H(e^{j\omega})| &= K\frac{|e^{j\omega} - 1 + e^{-j\omega}|}{|e^{j\omega} - a + a^2e^{-j\omega}|} \\ &= (1-a+a^2)\frac{|e^{j\omega} - 1 + e^{-j\omega}|}{|e^{j\omega} - a + a^2e^{-j\omega}|}. \end{aligned} \tag{5.64}$$

The constant a (the magnitude of the poles) is determined by the spec that the gains at 900 and 1100 Hertz are both 0.90. The discrete-time frequencies corresponding to 900 and 1100 Hertz are

$$\begin{aligned}\omega_L &= 2\pi\frac{900}{6000} = 0.300\pi. \\ \omega_H &= 2\pi\frac{1100}{6000} = 0.367\pi. \end{aligned} \quad (5.65)$$

a must be determined by solving these equations:

$$\begin{aligned}0.90 &= (1-a+a^2)\frac{|e^{j0.300\pi}-1+e^{-j0.300\pi}|}{|e^{j0.300\pi}-a+a^2e^{-j0.300\pi}|}. \\ 0.90 &= (1-a+a^2)\frac{|e^{j0.367\pi}-1+e^{-j0.367\pi}|}{|e^{j0.367\pi}-a+a^2e^{-j0.367\pi}|} \end{aligned} \quad (5.66)$$

These equations can be satisfied simultaneously since the gain is very close to symmetric about f=1000 Hertz. The easiest way to solve these equations is to plot the gain for different values of a. Using trial-and-error, $a = 0.95$ gives the gain plotted in the two figures below, one for 900$\leq f \leq$1100 Hertz and one for $0 \leq f \leq 3000$ Hertz. This satisfies all the specs.

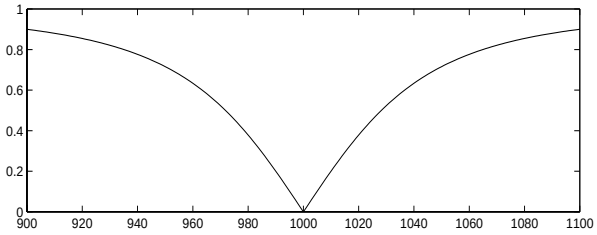


Figure 5.11: Filter gain for 900$\leq f \leq$1100 Hertz.

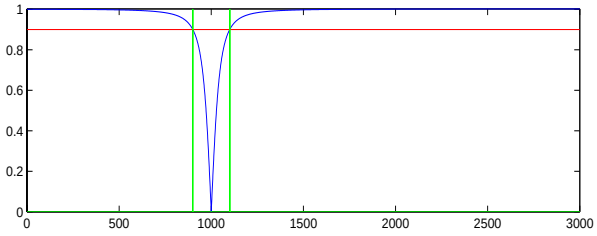


Figure 5.12: Filter gain for 0$\leq f \leq$3000 Hertz.

The Matlab code for this example:

```
F=[900:1100];EW=exp(j*2*pi*F/6000);
A=0.95;%Try different values of A.
H=polyval([1 -1 1],EW);
H=H./polyval([1 -A A*A],EW);
H=H*(1-A+A*A);plot(F,abs(H))
F=[0:3000];EW=exp(j*2*pi*F/6000);A=0.95;
H=polyval([1 -1 1],EW);H=H*(1-A+A*A);
H=H./polyval([1 -A A*A],EW);
ZZ=[zeros(1,900) 1 zeros(1,200)];
ZZ=[ZZ 1 zeros(1,1899)];
subplot(211),plot(F,abs(H)),hold on
subplot(211),plot(F,ZZ,'g'),hold on
subplot(211),plot(F,0.90*ones(1,3001),'r')
zplane([1 -1 1],[1 -.95 .9025]
```

Using a=0.95, the transfer function is

$$H(z) = 0.9525\frac{z^2-z+1}{z^2-0.95z+0.9025}. \quad (5.67)$$

The difference equation can be read off of $H(z)$ as

$$\begin{aligned}&y[n]-0.95y[n-1]+0.9025y[n-2] \\ &= 0.9525(x[n]-x[n-1]+x[n-2]). \end{aligned} \quad (5.68)$$

The pole-zero diagram is shown in the next figure.

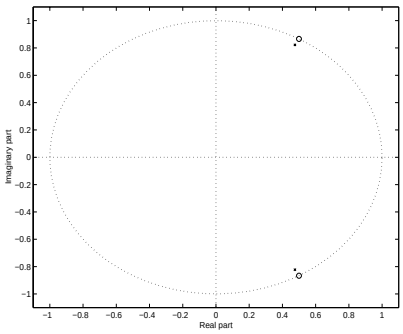


Figure 5.13: Pole-zero diagram for notch filter.

5.8 APPLICATION: Comb Filters: Removing Periodic Interference from Signals

Now suppose we are measuring a signal $x(t)$ to which *periodic* interference has been added. This might oc-

cur in a lab in which a machine (such as a motor) plugged into a wall socket is generating interference with a period of $\frac{1}{60}$ sec. The goal is to eliminate *periodic* interference and leave the signal unaffected.

Since the interference is periodic with period $\frac{1}{60}$ second, it has a Fourier series expansion with harmonics at 60,120,180... Hertz. Let us assume that the interference is bandlimited, either because the interference actually is bandlimited, or because the signal $x(t)$ is bandlimited and we can use a low-pass filter, such as a Butterworth filter, to eliminate higher frequencies of both $x(t)$ and the interference. The maximum frequency of the interference is then $60K$ Hertz for some integer K.

The gain of a system that rejects such interference will have a *series of notches* at $60k$ Hertz. Since the notches are equally spaced, this system is called a *comb filter*, since the gain resembles a comb.

5.8.1 Derivation of Comb Filter

In continuous time, the periodic interference can be expanded in a Fourier series. Hence it can be eliminated using a cascade (series connection) of notch filters, each of which eliminates a single harmonic of the periodic interference.

After sampling, each harmonic of the interference becomes a discrete-time sinusoid, which can be eliminated using a discrete-time notch filter. So the periodic interference can be removed using a cascade of notch filters, just as in continuous time.

However, note that a signal that is periodic in continuous time will usually *not* be periodic after sampling, unless the period of the signal is an integer multiple of the sampling interval T. Equivalently, the sampling rate (in samples per second) is an integer multiple of the frequency (in Hertz) of the fundamental frequency of the periodic signal. So it is *not* correct to state that a discrete-time comb filter eliminates harmonics of the DTFS (discussed in the next chapter) of the sampled periodic signal: The sampled periodic signal is usually not periodic, so it does not have a DTFS expansion. But this doesn't matter.

Indeed, if the interference is still periodic with period N after sampling, and the sampled desired signal has no components near the harmonics of the interference, then the interference can be eliminated using the simple MA filter $y[n]=x[n]–x[n–N]$. This quite clearly eliminates any signal with period N! A more selective filter should be used, such as

$$y[n] - 0.99y[n-N] = x[n] - x[n-N] \qquad (5.69)$$

But in practice, interference is seldom periodic after sampling, so a cascade of notch filters is needed.

5.8.2 Comb Filter Examples

Example: Simple comb filter.

Find the ARMA difference equation for a comb filter that rejects periodic interference that has period=0.01 sec. and is bandlimited to 200 Hertz. The sampling rate is 600 samples per sec. Use a=0.99.

Solution:

A periodic signal with period=0.01 sec. bandlimited to 200 Hertz has harmonics at 100 and 200 Hertz. The comb filter is the series connection of notch filters with middle coefficients $-2\cos(2\pi\frac{100}{600})$=–1 and $-2\cos(2\pi\frac{200}{600})$=1. The MA part of the comb filter is $\{1,-1,1\} * \{1,1,1\} = \{1,0,1,0,1\}$. The AR part is the MA part weighted by a^i. a^2=0.98 and a^4=0.96.

$$y[n]–0.98y[n–2]+0.96y[n–4]=x[n]–x[n–2]+x[n+4]$$

Example: Implemented comb filter.

A sinusoidal signal at 30 Hertz is corrupted by zero-mean periodic interference with period $\frac{1}{60}$ sec. from a motor. The interference is bandlimited to 180 Hertz. Using a DSP system with a sampling rate of 480 samples per sec., design a comb filter that keeps the sinusoid and rejects the interference. Use a=0.95.

Solution:

Since the interfering motor signal has period $\frac{1}{60}$ sec., it has harmonics at integer multiples of 60 Hertz. Since the interference is zero-mean and bandlimited to 180 Hertz, it has harmonics only at 60, 120, and 180 Hertz ("zero-mean" means there is no dc (ω=0) component). After sampling, the discrete-time frequencies to be rejected are $\omega_o=\frac{2\pi 60k}{480}=\pi\frac{k}{4}, k = 1,2,3$.

The transfer function is a series connection of three

notch filters with frequencies $\omega=\frac{\pi}{4}$, $\frac{2\pi}{4}$, and $\frac{3\pi}{4}$:

$$\begin{aligned} H(z) &= \prod_{k=1}^{3} \frac{(z-e^{jk\pi/4})(z-e^{-jk\pi/4})}{(z-0.95e^{jk\pi/4})(z-0.95e^{-jk\pi/4})} \\ &= \prod_{k=1}^{3} \frac{z^2-2\cos(k\pi/4)z+1}{z^2-2(0.95)\cos(k\pi/4)z+(0.95)^2} \end{aligned} \quad (5.70)$$

After some algebra, $H(z)$ simplifies to

$$H(z) = \frac{z^6+z^4+z^2+1}{z^6+(0.95)^2z^4+(0.95)^4z^2+(0.95)^6}. \quad (5.71)$$

The comb filter is implemented using the ARMA difference equation, which is read off of $H(z)$ as

$$\begin{aligned} &y[n]+(0.95)^2y[n-2]+(0.95)^4y[n-4]+(0.95)^6y[n-6] \\ &= x[n]+x[n-2]+x[n-4]+x[n-6]. \end{aligned} \quad (5.72)$$

The original (left) and filtered (right) signals are shown next. Only the sinusoid remains.

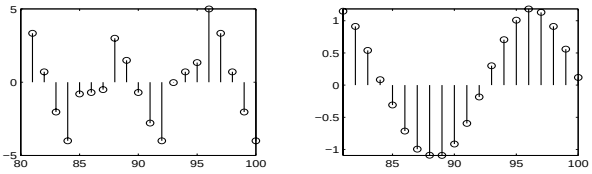


Figure 5.14: Time waveforms of the original (left) and filtered (right) signals.

The impulse response, although it is not just a decaying sinusoid like the notch filter impulse response, decays as $(0.95)^n$. It becomes negligible after about 80 time samples, which is $\frac{80}{480}$=0.167 sec. So the transient response has a duration of 80 time samples. The output is plotted starting at time sample n=81.

The one-sided gain (left) and pole-zero diagram (right) are shown in the next figure. As with notch filter, the dc gain $H(e^{j0})$ is slightly greater than one. This is corrected by dividing $H(z)$ by $H(e^{j0})$.

The alert reader will have noticed that the sampling rate of 480 samples/second is an integer multiple of the fundamental frequency 60 Hertz of the interference. This means that the interference is still

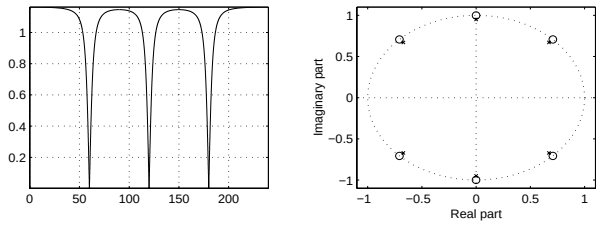


Figure 5.15: One-sided gain (left) and pole-zero diagram (right) for comb filter example.

periodic after sampling, with a period of N=$\frac{480}{60}$=8, so it could be eliminated using the simpler filter

$$y[n]-0.95^8y[n-8] = x[n]-x[n-8]. \quad (5.73)$$

To relate this to the comb filter used above, we compute the transfer function associated with this difference equation. Taking the z-transform gives

$$Y(z)[1-0.95^8z^{-8}] = X(z)[1-z^{-8}] \quad (5.74)$$

from which the transfer function is determined as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1-z^{-8}}{1-0.95^8z^{-8}}\frac{z^8}{z^8} = \frac{z^8-1}{z^8-0.95^8}. \quad (5.75)$$

$H(z)$ has zeros at $\{e^{j2\pi k/8}, k$=$0\ldots7\}$ and poles at $\{0.95e^{j2\pi k/8}, k$=$0\ldots7\}$. These are the same locations as the zeros and poles for the comb filter, with additional zeros at ±1 and additional poles at ±0.95. So the "simpler" system (5.73) is actually the comb filter cascaded with two *additional* notch filters that reject ω=0 and ω=π!

The Matlab code for this example:

```
N=[1:100];M=[81:100];X=cos(2*pi*30*N/480);
X=X+3*cos(2*pi*60*N/480);
X=X+sin(2*pi*120*N/480)+cos(2*pi*180*N/480);
B=[1 0 1 0 1 0 1];A=[1 0 .9 0 .81 0 .73];
Y=filter(B,A,X);
subplot(221),stem(M,X(M))
subplot(222),stem(M,Y(M))
F=linspace(0,240,10000);W=exp(j*2*pi*F/480);
H=polyval(B,W)./polyval(A,W);
subplot(223),plot(F,abs(H))
subplot(224),zplane(B,A)
```

Example: Real-world comb filter.

We are given the signal of two actual trumpets playing simultaneously. One trumpet is playing note A (which has a period of $\frac{1}{440}$ sec.) and the other trumpet is playing note B (which has a period of $\frac{1}{494}$ sec.). The signal was sampled at the standard CD sampling rate of 44100 samples per sec. Design and implement a discrete-time comb filter to eliminate the trumpet playing note A, while keeping the trumpet playing note B. Use a=0.99.

Solution:

The sampling interval is $\frac{1}{44100}$ sec. The continuous-time and discrete-time frequencies of the trumpet are:

Note	Continuous	Discrete
Reject A	$440k$ Hertz	$\frac{2\pi 440k}{44100}$=$0.0627k$
Pass B	$494k$ Hertz	$\frac{2\pi 494k}{44100}$=$0.0704k$

We note that the sampling rate of 44100 samples per second has 44100=(101)(437)=(100)(441). So we may use a simple comb filter of form (5.73) with

N	Continuous	Discrete
101=$\frac{44100}{437}$	$437k$ Hertz	$\frac{2\pi 437k}{44100}$=$0.0622k$
100=$\frac{44100}{441}$	$441k$ Hertz	$\frac{2\pi 441k}{44100}$=$0.0628k$

N=100 and N=101 were both tried; N=101 was found to work better. So the comb filter used is

$$y[n] - 0.99^{101}y[n-101] = x[n] - x[n-101]. \quad (5.76)$$

The frequency response of the discrete-time comb filter is plotted, in red, in the next figure. The spectrum of the two-trumpets signal is also plotted, in blue, in the next figure. The spectra of the trumpets have been multiplied by ten to make them easier to see on the plot. Since the two-trumpets signal has been sampled, its spectrum is plotted vs. discrete-time frequency ω. Although discrete-time spectra are periodic in ω with period 2π, the two-trumpets spectrum is plotted only for $0 \leq \omega \leq 1$, since the two-trumpets spectrum is negligible for $1 \leq |\omega| \leq \pi$. It is clear that the comb filter will reject the trumpet playing note A and pass the trumpet playing note B.

The time waveform of the two-trumpets signal is plotted in the next figure. It makes little sense.

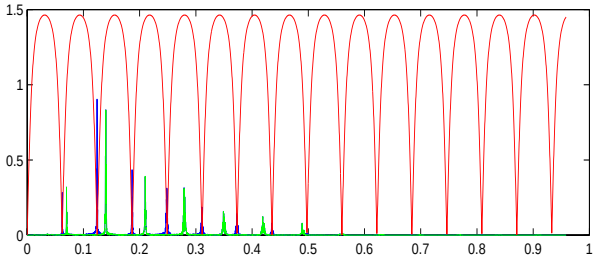


Figure 5.16: Comb filter gain (red) and two-trumpets spectrum (blue) for comb filter. The spectra of the trumpets have been multiplied by ten.

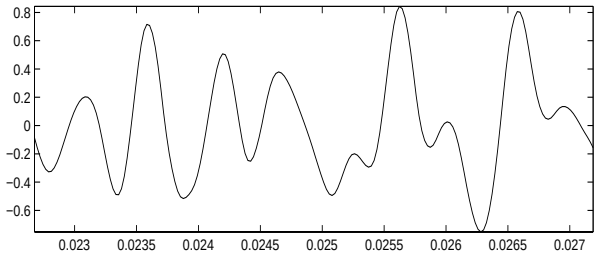


Figure 5.17: Time waveform of two-trumpets signal for comb filter example.

The time waveform of the comb-filtered two-trumpets signal is plotted in the next figure. The filtered signal clearly is similar to the trumpet waveform in the first chapter. Both plots use 200 points.

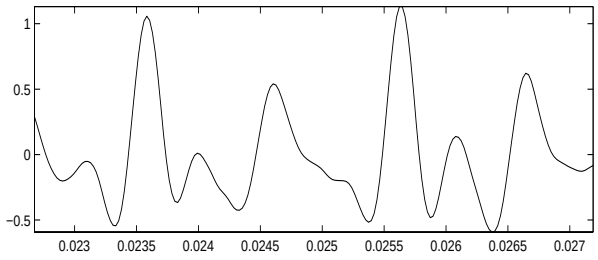


Figure 5.18: Time waveform of comb-filtered two-trumpets signal for comb filter example. This is similar to the trumpet signal in the first chapter.

The Matlab code for this example:

```
load twotrumpets.mat;S=44100;N=length(X);
A=[1 zeros(1,100) -0.99^(100)];
B=[1 zeros(1,100) -1];F=[0:4999]*S/N;
W=2*pi/N*[0:4999];EW=exp(j*W);
H=polyval(B,EW)./polyval(A,EW);
FX=abs(fft(X));Y=filter(B,A,X);
plot(W,FX(1:5000)/N*10,W,abs(H),'r')
NN=[1000:1199];T=NN/S;
subplot(211),plot(T,X(NN))
subplot(212),plot(T,Y(NN))
soundsc(X,S),pause,soundsc(Y,S)
```

For completeness, we also provide, without implementation, the design of the comb filter as a series connection of six notch filters at discrete-time frequencies $\{0.0627k, k{=}1\ldots6\}$. Only six harmonics of the trumpet playing note A are rejected, since (1) the seventh and higher harmonics are negligible, and (2) the frequencies of the seventh and eight harmonics of note A (0.4388 and 0.5015) are very close to the sixth and seventh harmonics of note B (0.4223 and 0.4927). This is why the comb filter above did not work perfectly–some harmonics of note B (which were to be kept) were eliminated, along with the corresponding harmonics of note A.

The transfer function of this comb filter is

$$\begin{aligned} H(z) &= \prod_{k=1}^{6} \frac{(z-e^{j0.0627k})(z-e^{-j0.0627k})}{(z-0.99e^{j0.0627k})(z-0.99e^{-j0.0627k})} \\ &= \prod_{k=1}^{6} \frac{z^2-2z\cos(0.0627k)+1}{z^2-1.98z\cos(0.0627k)+0.98}. \end{aligned} \quad (5.77)$$

The impulse response and frequency response must be computed numerically, and are not shown.

We will revisit this problem, and do a much better job of eliminating one trumpet while leaving the other unaffected, in a later chapter, using *batch filtering*.

5.9 APPLICATION: Low-Pass Filters using Poles & Zeros

Design of discrete-time filters is discussed in detail in a later chapter. In practice, discrete-time filters other than notch and comb filters is usually performed using numerical algorithms that solve some sort of optimization problem. We discuss this in a later chapter.

Despite this, we will now design a discrete-time *half-band* low-pass filter by placing ten poles and zeros. Half-band filters pass discrete-time frequencies less than $\frac{\pi}{2}$, and reject ones greater than $\frac{\pi}{2}$:

$$H_{\rm HALF-BAND}(e^{j\omega}) = \begin{cases} 1, & |\omega| < \frac{\pi}{2} \\ 0, & |\omega| > \frac{\pi}{2} \end{cases} \quad (5.78)$$

for $|\omega| < \pi$ (remember the frequency response $H(e^{j\omega})$ is periodic in ω with period 2π). This is the discrete-time equivalent of a "brick-wall" low-pass filter.

We may attempt to design a discrete-time half-band low-pass filter by placing poles inside the right half of the unit circle to enhance low frequencies, and zeros on the left half of the unit circle to almost eliminate high frequencies. This will virtually eliminate the high frequencies, as desired, but the low frequencies will be non-uniformly boosted, since each pole produces a peak in the gain. So the poles must be moved farther inside the unit circle non-uniformly.

Example: Half-band filter design using poles & zeros.

Design a half-band filter using five zeros, and five poles, each of which has magnitude 0.8.

Solution:

It is clear that the five zeros should be equally-spaced on the left half of the unit circle. It may seem that the five poles should be equally-spaced inside the right half of the unit circle, but this makes $H(e^{j\omega})$ very wavy for $0 \le \omega < \frac{\pi}{2}$. Better results are obtained by moving the pole with phase angle zero farther inside the unit circle, at radius 0.6 instead of 0.8. In summary, we use the following:

- **Zeros:** $\{e^{\pm j\pi/2}, e^{\pm j3\pi/4}, e^{j\pi}\}$.
- **Poles:** $\{0.6, 0.8e^{\pm j\pi/4}, 0.8e^{\pm j\pi/2}\}$.

The transfer function with these poles and zeros is

$$H(z) = \frac{z^5+2.414z^4+3.414z^3+3.414z^2+2.414z+1}{z^5-1.73z^4+1.96z^3-1.49z^2+0.84z-0.25}. \quad (5.79)$$

The difference equation is read off of $H(z)$:

$$y[n] - 1.73y[n-1] + 1.96y[n-2]$$

$$-1.49y[n-3]+0.84y[n-4]-0.25y[n-5] = x[n]+2.414x[n-1]+3.414x[n-2] +3.414x[n-3]+2.414x[n-4]+x[n-5] \quad (5.80)$$

The one-sided gain (left) and pole-zero diagram (right) are shown in the next figure. The dc gain is nearly 40! But this can easily be corrected, as before, by dividing $H(z)$ by 40.

There is still a waviness in the gain for low frequencies, and the transition from high to low gain is gradual, but this is indeed a crude half-band filter.

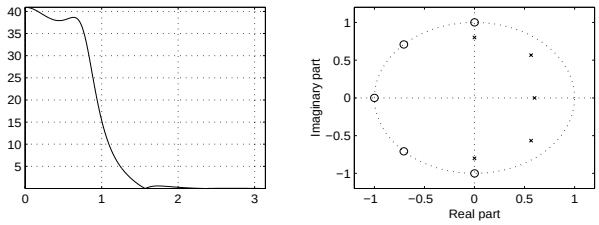


Figure 5.19: One-sided gain (left) and pole-zero diagram (right) for half-band filter example.

The Matlab code for this example:

```
clear;Z=exp(j*pi*[2:6]/4);
P=0.8*exp(j*pi*[1 2 -1 -2]/4);
P=[P 0.6];W=linspace(0,pi,1000);
B=poly(Z);A=poly(P);Z1=exp(j*W);
H=polyval(B,Z1)./polyval(A,Z1);
subplot(221),plot(W,abs(H))
subplot(222),zplane(B,A)
```

Example: Half-band filter using zeros.
Design a half-band filter using ten zeros.

Solution:
We choose for the half-band filter the ten zeros

- $2e^{\pm j18^o}, 1.864e^{\pm j53.3^o}, 1.26e^{\pm j117.4^o}$,
- $1.41e^{\pm j139.7^o}, 1.40e^{\pm j166^o}$.

Why these zeros? They are the zeros resulting from using a Hamming window on the ideal impulse response of a half-band filter. We will discuss ideal impulse response using inverse DTFT, Hamming windows, and FIR filter design techniques in three different later chapters in these notes.

The impulse response (top) and gain (bottom) are plotted in the next figure. The transition is less sharp, but there is no waviness in the pass band.

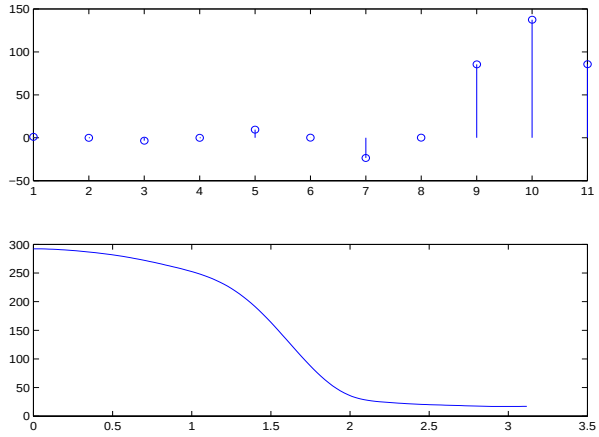


Figure 5.20: Impulse response (top) and one-sided gain (bottom) for all-zero half-band filter example.

The pole-zero diagram is plotted in the next figure.

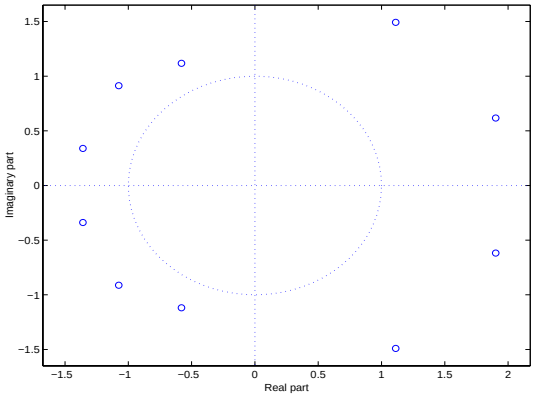


Figure 5.21: Pole-zero diagram for half-band filter.

```
M=[2,1.86,1.26,1.41,1.40];
P=[18,53.3,117.4,139.7,166];
Z=M.*exp(j*pi/180*P);Z=[Z conj(Z)];
h=poly(Z);subplot(211),stem(h)
w=2*pi*[0:127]/256;H=abs(fft(h,256));
subplot(212),plot(w,H(1:128))
figure,zplane(Z')
```

Chapter 6

Discrete-Time Fourier Series and Transforms

The Discrete-Time Fourier Series (DTFS) and Discrete-Time Fourier Transform (DTFT) are the discrete-time counterparts to the continuous-time Fourier series and transform, respectively, and play the same roles in discrete time as they do in continuous time. The main changes in discrete time are:

- DTFS has only a finite number of terms;
- DTFT is periodic in frequency with period 2π.
- In practice, both the DTFS and DTFT are computed numerically using the Discrete Fourier Transform (DFT), covered in the next chapter.

We cover DTFS first since, as in continous time, it is conceptually simpler.

6.1 The Discrete-Time Fourier Series (DTFS)

The discrete-time Fourier series (DTFS) is the discrete-time counterpart to Fourier series expansions of periodic signals. Unlike the continuous-time Fourier series (CTFS), the DTFS is finite. Computation of its coefficients requires summations rather than integrals. The DTFS plays two roles:

(1) The response of an LTI system to any periodic signal can be computed by computing the DTFS of the input, computing the response to each DTFS term using the frequency response function, and summing the results to obtain the DTFS of the output. This is the basic concept of *filtering* periodic signals.

(2) The DTFS computes the *(line) spectrum* of a discrete-time periodic signal. The spectrum of a periodic signal can reveal details such as modes of vibration of a rotating machine, periodicities of signals, and fundamentals and harmonics of sampled music.

6.1.1 Forms of the DTFS

Let $x[n]$ be a *periodic* signal with period N, so that $x[n]$=$x[n+N]$ for all times n. Then $x[n]$ can be expanded in the *Discrete-Time Fourier Series (DTFS)* (listed next to the CTFS for comparison)

$$x[n] = \sum_{k=0}^{N-1} x_k e^{j2\pi \frac{k}{N} n}; x(t) = \sum_{k=-\infty}^{\infty} x_k e^{j2\pi \frac{k}{T} t} \quad (6.1)$$

If $x[n]$ is real-valued and N is odd, the DTFS is

$$\begin{aligned} x[n] &= x_0 \quad (6.2)\\ &+x_1 e^{j\frac{2\pi n}{N}} + x_2 e^{j\frac{4\pi n}{N}} + \ldots + x_{\frac{N-1}{2}} e^{j\frac{\pi(N-1)n}{N}} \\ &+x_1^* e^{-j\frac{2\pi n}{N}} + x_2^* e^{-j\frac{4\pi n}{N}} + \ldots + x_{\frac{N-1}{2}}^* e^{-j\frac{\pi(N-1)n}{N}}. \end{aligned}$$

The second series of terms comes from

$$\begin{aligned} x_{N-k} &= x_k^* \quad (6.3)\\ e^{j2\pi\frac{N-k}{N}n} &= e^{-j2\pi\frac{k}{N}n} \\ x_{N-k} e^{j2\pi\frac{N-k}{N}n} &= x_k^* e^{-j2\pi\frac{k}{N}n} \end{aligned}$$

If $x[n]$ is real-valued and N is even, the DTFS has an additional term at $\omega=\frac{2\pi}{N}\frac{N}{2}=\pi$:

$$\begin{aligned} x[n] &= x_0 + x_N e^{j\pi n} \\ &+x_1 e^{j\frac{2\pi n}{N}} + x_2 e^{j\frac{4\pi n}{N}} + \ldots + x_{\frac{N}{2}-1} e^{j\frac{\pi(N-2)n}{N}} \\ &+x_1^* e^{-j\frac{2\pi n}{N}} + x_2^* e^{-j\frac{4\pi n}{N}} + \ldots + x_{\frac{N}{2}-1}^* e^{-j\frac{\pi(N-2)n}{N}}. \end{aligned} \tag{6.4}$$

These forms emphasize the similarity between the DTFS and CTFS, and one important difference: *the DTFS is finite.* This is not surprising: since $x[n]$ has period N, it is completely characterized by N numbers, so an infinite Fourier series should not be necessary. To summarize the DTFS expansion:

- The DTFS uses frequencies $\omega=\pm\frac{2\pi k}{N}$:
- If N is even: It uses indices $k=0\ldots\frac{N}{2}$
- If N is odd: It uses indices $k=0\ldots\frac{N-1}{2}$
- The CTFS uses $\omega=\pm\frac{2\pi k}{T}, k=0,1,\ldots\infty$.

If $x[n]$ is real-valued, the DTFS can, like the CTFS, be written in terms of sinusoids as $x[n]=$

$$x_0+2|x_1|\cos\left(\frac{2\pi}{N}n+\theta_1\right)+2|x_2|\cos\left(\frac{4\pi}{N}n+\theta_2\right)+\ldots \tag{6.5}$$

where $x_k=|x_k|e^{j\theta_k}$. Note the amplitudes of terms at $\omega=0$ and $\omega=\pi$ (if N is even) do *not* double, and

$$e^{j\pi n} = e^{-j\pi n} = \cos(\pi n) = (-1)^n. \tag{6.6}$$

The sinusoidal representation follows from

$$\begin{aligned} x_k e^{j\frac{2\pi n}{N}k} &+ x_k^* e^{j\frac{2\pi n}{N}(N-k)} = \\ |x_k|e^{j\theta_k}e^{j\frac{2\pi n}{N}k} &+ |x_k|e^{-j\theta_k}e^{j\frac{2\pi n}{N}(N-k)} \\ &= 2|x_k|\cos\left(\frac{2\pi n}{N}k+\theta_k\right) \end{aligned} \tag{6.7}$$

To emphasize the analogy between the CTFS and DTFS, we present the partial sums of the DTFS of a square wave. This should be compared to the analogous plot for CTFS. The difference is that the DTFS has a finite number of terms, so it sums to the square wave exactly if enough (here, eight) terms are used.

6.1.2 Computing DTFS Coefficients

The coefficients of the CTFS were computed using the orthogonality of the basis functions $\{e^{j2\pi\frac{k}{T}t}\}$ over the interval $0\le t<T$. The orthogonality of these basis functions had two important implications: (1) it enabled the existence of closed-form expressions for the coefficients; (2) it enabled Parseval's theorem.

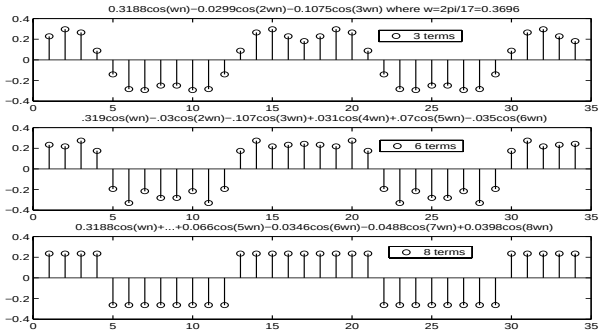


Figure 6.1: DTFS partial sums for square wave.

Orthogonality of $\{e^{j2\pi\frac{k}{N}n}\}$

The coefficients of the DTFS are computed using the orthogonality of the basis functions $\{e^{j2\pi\frac{k}{N}n}\}$ over the interval $0\le n<N$. The orthogonality of these basis functions has two important implications: (1) it enables the existence of closed-form expressions for the coefficients; and (2) it enables Parseval's theorem.

We now show that the basis functions $\{e^{j2\pi\frac{k}{N}n}\}$ are orthogonal over the interval $0\le n<N$. Setting $r=e^{j2\pi\frac{k-m}{N}}$ in the finite sum geometric series

$$\sum_{n=0}^{N-1} r^n = \frac{r^N-1}{r-1} \text{ if } r\neq 1 \tag{6.8}$$

shows that $\{e^{j2\pi\frac{k}{N}n}\}$ are orthogonal, since

$$\begin{aligned} \sum_{n=0}^{N-1} e^{j2\pi\frac{k}{N}n}e^{-j2\pi\frac{m}{N}n} &= \sum_{n=0}^{N-1} e^{j2\pi\frac{k-m}{N}n} \\ = \frac{e^{j2\pi\frac{k-m}{N}N}-1}{e^{j2\pi\frac{k-m}{N}(1)}-1} = 0 &\quad \text{if} \quad k\neq m \\ = \sum_{n=0}^{N-1} e^{j0} = N &\quad \text{if} \quad k=m. \end{aligned} \tag{6.9}$$

Using the discrete-time impulse $\delta[n]$, we can summa-

rize this result as the orthogonality relation

$$\sum_{n=0}^{N-1} e^{j2\pi\frac{k}{N}n} e^{-j2\pi\frac{m}{N}n} = N\delta[k-m]. \quad (6.10)$$

Formula for Computing DTFS Coefficients

Multiplying the DTFS (6.1) by $e^{-j2\pi\frac{m}{N}n}$ and summing over n from 0 to $N-1$ gives

$$\begin{aligned}\sum_{n=0}^{N-1} e^{-j2\pi\frac{m}{N}n} x[n] &= \sum_{n=0}^{N-1} e^{-j2\pi\frac{m}{N}n} \sum_{k=0}^{N-1} x_k e^{j2\pi\frac{k}{N}n} \\ &= \sum_{k=0}^{N-1} x_k \sum_{n=0}^{N-1} e^{j2\pi\frac{k}{N}n} e^{-j2\pi\frac{m}{N}n}\end{aligned}$$

where the final equality comes from exchanging the order of the (finite) summations. Using orthogonality (6.10) of the $\{e^{j2\pi\frac{k}{N}n}\}$ gives

$$\sum_{n=0}^{N-1} e^{-j2\pi\frac{m}{N}n} x[n] = \sum_{k=0}^{N-1} x_k N\delta[k-m] = Nx_m. \quad (6.11)$$

Note that all the sums are finite, so there are no convergence issues. Dividing by N shows that the DTFS coefficients x_k can be computed using

$$x_k = \frac{1}{N}\sum_{n=0}^{N-1} e^{-j2\pi\frac{k}{N}n} x[n], k = 0\ldots N-1. \quad (6.12)$$

Compare this to the CTFS formula

$$x_k = \frac{1}{T}\int_{t=0}^{T} e^{-j2\pi\frac{k}{T}t} x(t)dt. \quad (6.13)$$

Unlike the CTFS, which required integrals, this is a finite sum. We will see in the next chapter that a fast algorithm, the FFT, can be used to compute the $\{x_k\}$ very quickly from $\{x[n], n{=}0\ldots N{-}1\}$

In particular, the *mean* (dc value) of $x[n]$ is

$$x_0 = \frac{1}{N}\sum_{n=0}^{N-1} x[n]. \quad (6.14)$$

If N is even, the highest-frequency coefficient is

$$x_{N/2} = \frac{1}{N}(x[0] - x[1] + x[2] - x[3] + \ldots - x[N-1]). \quad (6.15)$$

Also, if $x[n]$ is real-valued, then $x_{N-k}{=}x_k^*$. This is conjugate symmetry in the DTFS.

Parseval's Theorem

Parseval's theorem for the DTFS states that the average power of $x[n]$ is the same when computed in either the time or the frequency domain:

$$\frac{1}{N}\sum_{n=0}^{N-1} |x[n]|^2 = \sum_{k=0}^{N-1} |x_k|^2. \quad (6.16)$$

Parseval's theorem may be derived as follows. First,

$$\sum_{n=0}^{N-1} |x[n]|^2 = \sum_{n=0}^{N-1} x[n]x[n]^*. \quad (6.17)$$

Substituting the DTFS for $x[n]$ and $x[n]^*$ gives

$$\sum_{n=0}^{N-1} |x[n]|^2 = \sum_{n=0}^{N-1} \left[\sum_{k_1=0}^{N-1} x_{k_1} e^{j2\pi\frac{k_1}{N}n}\right] \left[\sum_{k_2=0}^{N-1} x_{k_2} e^{j2\pi\frac{k_2}{N}n}\right]^* \quad (6.18)$$

Rearranging the order of (finite) summations gives

$$\sum_{n=0}^{N-1} |x[n]|^2 = \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} x_{k_1} x_{k_2}^* \sum_{n=0}^{N-1} e^{j2\pi\frac{k_1}{N}n} e^{-j2\pi\frac{k_2}{N}n}. \quad (6.19)$$

The orthogonality (6.10) of the $\{e^{j2\pi\frac{k}{N}n}\}$ gives

$$\begin{aligned}\sum_{n=0}^{N-1} |x[n]|^2 &= \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} x_{k_1} x_{k_2}^* N\delta[k_1-k_2] \\ &= N\sum_{k_1=0}^{N-1} |x_{k_1}|^2. \quad (6.20)\end{aligned}$$

Dividing by N gives Parseval's theorem.

Example: Computing DTFS coefficients. Compute the DTFS of the periodic signal

$$x[n] = \{\ldots \underline{24}, 8, 12, 16, 24, 8, 12, 16\ldots\} \quad (6.21)$$

Also confirm Parseval's theorem holds.

Solution:

The period of $x[n]$ is N=4. $e^{-j2\pi\frac{k}{4}n}$=$(-j)^{nk}$, so

$$\begin{aligned} x_0 &= \frac{1}{4}[24(1)+8(1)+12(1)+16(1)] = 15 \quad (6.22)\\ x_1 &= \frac{1}{4}[24(1)-8(j)-12(1)+16(j)] = 3+j2.\\ x_2 &= \frac{1}{4}[24(1)-8(1)+12(1)-16(1)] = 3.\\ x_3 &= \frac{1}{4}[24(1)+8(j)-12(1)-16(j)] = 3-j2. \end{aligned}$$

x_0 is the mean value of $x[n]$. Since $x[n]$ is real-valued, x_3=x^*_{4-3}=x^*_1 by conjugate symmetry. These coefficients can also be computed in Matlab using

```
fft([24 8 12 16])/4
```

The DTFS expansion of $x[n]$ is then

$$\begin{aligned} & 15+(3+2j)e^{j2\pi\frac{1}{4}n}+3e^{j\pi n}+(3-2j)e^{j2\pi\frac{3}{4}n}\\ = \;& 15+7.2\cos(\frac{\pi}{2}n+33.7^o)+3\cos(\pi n) \quad (6.23) \end{aligned}$$

since $3+j2$=$3.6e^{j33.7^o}$. Inserting n=0,1,2,3 into all expressions for $x[n]$ confirms they agree.

Parseval's theorem for this example is

- $\frac{1}{4}(|24|^2+|8|^2+|12|^2+|16|^2) = 260.$
- $|15|^2+|3+2j|^2+|3|^2+|3-2j|^2 = 260.$

Example: DTFS of a periodic sinusoid.
Compute DTFS of $x[n]$=$A\cos(2\pi\frac{M}{N}n+\theta)$ where $\frac{M}{N}$ has been reduced to lowest terms. Note that $x[n]$ is then periodic with period N.

Solution:

$$\begin{aligned} x[n] &= A\cos(2\pi\frac{M}{N}n+\theta) \quad (6.24)\\ &= \frac{A}{2}e^{j\theta}e^{j2\pi\frac{M}{N}n}+\frac{A}{2}e^{-j\theta}e^{-j2\pi\frac{M}{N}n}\\ &= \frac{A}{2}e^{j\theta}e^{j2\pi\frac{M}{N}n}+\frac{A}{2}e^{-j\theta}e^{j2\pi\frac{N-M}{N}n} \end{aligned}$$

and comparing with the definition (6.1) of DTFS, we can *read off* the DTFS coefficients as

$$x_k = \begin{cases} (A/2)e^{j\theta}, & k=M\\ (A/2)e^{-j\theta}, & k=N-M\\ 0, & \text{otherwise} \end{cases} \quad (6.25)$$

Note that this is only true for *periodic* sinusoids. It shows that the DTFS computes the line spectrum of a discrete-time *periodic* sinusoid.

Example: DTFS of $4\cos(0.15\pi n+1)$**.**

Solution:

$$4\cos(0.15\pi n+1) = 4\cos(2\pi\frac{3}{40}n+1) \quad (6.26)$$

So M=3 and the period N=40 and

$$x_k = \begin{cases} 2e^{j1}, & k=3\\ 2e^{-j1}, & k=37\\ 0, & \text{otherwise} \end{cases} \quad (6.27)$$

6.1.3 DTFS and Frequency Response

We've computed the DTFS; what do we do with it?

Let a periodic signal $x[n]$ with period N be input into an LTI system with impulse response $h[n]$. The goal is to compute the DTFS of the output $y[n]$ from the DTFS of the input $x[n]$.

We compute the response of the system to each term of the DTFS of $x[n]$ as follows:

- $H(e^{j\omega})$=$\sum_{n=-\infty}^{\infty}h[n]e^{j\omega n}$.
- $x_k e^{j2\pi\frac{k}{N}n} \rightarrow \boxed{h[n]} \rightarrow H(e^{j2\pi\frac{k}{N}})x_k e^{j2\pi\frac{k}{N}n}$
- $\sum_{k=0}^{N-1}x_k e^{j2\pi\frac{k}{N}n} \rightarrow \sum_{k=0}^{N-1}H(e^{j2\pi\frac{k}{N}})x_k e^{j2\pi\frac{k}{N}n}$
- DTFS of $x[n] \rightarrow \boxed{h[n]} \rightarrow$ DTFS of $y[n]$.

Each term of the DTFS of $x[n]$ is multiplied by $H(e^{j2\pi\frac{k}{N}})$ to get the corresponding term of the DTFS of $y[n]$. This is just like what happens for the CTFT, except that the DTFS is finite.

Example: Averaging a periodic signal.
The periodic signal with period N=6

$$x[n] = \{\ldots\underline{4},0,1,0,1,0,4,0,1,0,1,0\ldots\} \quad (6.28)$$

is input into the two-point averager

$$y[n] = \frac{1}{2}(x[n]+x[n-1]). \quad (6.29)$$

Compute the DTFS of the output $y[n]$.

Solution:

The frequency response function of the two-point averager was computed in a previous chapter. It was

$$H(e^{j\omega}) = \cos(\omega/2)e^{-j\omega/2}. \tag{6.30}$$

The DTFS of $x[n]$ can be computed using Matlab:

`fft([4 0 1 0 1 0])/6=[1 .5 .5 1 .5 .5].`

The DTFS of the input $x[n]$ is

$$\begin{aligned} x[n] &= 1e^{j2\pi\frac{0}{6}n} + .5e^{j2\pi\frac{1}{6}n} + .5e^{j2\pi\frac{2}{6}n} \\ &= 1e^{j2\pi\frac{3}{6}n} + .5e^{j2\pi\frac{4}{6}n} + .5e^{j2\pi\frac{5}{6}n} \\ &= 1 + \cos(\pi n/3) + \cos(2\pi n/3) + \cos(\pi n). \end{aligned} \tag{6.31}$$

As a check, we can use Parseval's theorem:

- $\frac{1}{6}(4^2+0^2+1^2+0^2+1^2+0^2)=3$.
- $(1)^2+(\frac{1}{2})^2+(\frac{1}{2})^2+(1)^2+(\frac{1}{2})^2+(\frac{1}{2})^2=3$.

The effect of the system on each term of the DTFS can be summarized in the following table:

ω :	0	$\pi/3$	$2\pi/3$	π
$H(e^{j\omega})$:	1	$0.866e^{-j\pi/6}$	$0.5e^{-j\pi/3}$	0
Gain:	1	0.866	0.5	0
Phase:	0	$-\pi/6$	$-\pi/3$	NA

So the DTFS of the output $y[n]$ is

$$\begin{aligned} & 1 + 0.866\cos(\frac{\pi}{3}n - \frac{\pi}{6}) + 0.5\cos(\frac{2\pi}{3}n - \frac{\pi}{3}) \\ = \;& 1 + 0.866\cos(\frac{\pi}{3}(n - \frac{1}{2})) + 0.5\cos(\frac{2\pi}{3}(n - \frac{1}{2})) \\ = \;& \{\ldots \underline{2}, 2, .5, .5, .5, .5, 2, 2, .5, .5, .5, .5 \ldots\} \end{aligned} \tag{6.32}$$

The averager has clearly smoothed $x[n]$ by reducing the high-frequency components (filtering) of $x[n]$.

Note that each input component has been "delayed" by $\frac{1}{2}$. Of course, a fractional delay in discrete time makes no sense, but the *effect* of the phase shifts is the same as what a fractional delay would do. This is a simple example of *linear phase*: the phase depends linearly on the frequency. It has the same effect that a (possibly fractional) time delay would have. There is no distortion of the input phase.

6.2 APPLICATION: Compute Spectra of Periodic Signals

Why should we care about the spectrum of a signal? The most important application is filtering out additive noise from the signal. The more we know about the spectrum of the signal (e.g., the frequencies of its largest components), the better a job we can do of removing the noise while leaving the signal. Other applications include determining the minimum sampling rate to avoid aliasing, and the modulation frequency needed to transmit multiple signals along a single channel (frequency domain multiplexing).

But unless the signal has a very simple analytic form, computation of its spectrum by analytically computing its Fourier transform or series is impossible. Instead, we sample the signal and compute its spectrum numerically from its samples. This section shows how to do this for a periodic signal. Spectra of non-periodic signals are covered in the DFT chapter.

6.2.1 Computation of Fourier Series Coefficients from Samples

Let $x(t)$ be a continuous-time periodic signal with period T seconds, so that $x(t)=x(t+T)$. Assume that $x(t)$ is bandlimited to F Hertz (its maximum frequency is F Hertz). The goal is to compute the complex coefficients x_k of the Fourier series expansion of $x(t)$. The $\{x_k\}$ define the *spectrum* of periodic $x(t)$.

Since $x(t)$ is periodic with period T, its spectrum consists of harmonics at frequencies $\{k/T\}$ Hertz for integers k. Since $x(t)$ is also bandlimited to F Hertz, there is an integer K such that $F=K/T$, or $K=TF$. For example, if $x(t)$ has period 1 msec, then it has harmonics at frequencies 1,2,3... kHz, so if it has a maximum frequency, that maximum frequency must be one of 1,2,3... kHz. It cannot be, say, 2.4 kHz. So the Fourier series is finite, with $2K+1$ terms:

$$x(t) = \sum_{k=-K}^{K} x_k e^{j2\pi kt/T}. \tag{6.33}$$

Now we sample $x(t)$ at N equally-spaced times in each period. Of course, sampling over more than a

single period would only give us the same samples. Setting $t=\{\frac{n}{N}T, n=0\ldots(N-1)\}$ in (6.33) gives

$$x\left(\frac{n}{N}T\right) = \sum_{k=-K}^{K} x_k e^{j2\pi k(\frac{n}{N}T)/T} = \sum_{k=-K}^{K} x_k e^{j2\pi nk/N}. \tag{6.34}$$

If $N \geq (2K+1)$, this is just the DTFS of $x(\frac{n}{N}T)$, which is a discrete-time periodic signal of period N! The condition $N \geq (2K+1)$ is needed since otherwise the number of terms being summed, $2K+1$, exceeds N. Furthermore, the number $2K+1$ of unknown coefficients x_k must not exceed the number N of known $x\left(\frac{n}{N}T\right)$ samples of $x(t)$. If $x(t)$ is real-valued, conjugate symmetry implies $x_{-k}=x_k^*$, but there are still $2K+1$ unknowns, since each $x_k, k \neq 0$ is complex, with a real part and an imaginary part (x_0 is real, with zero imaginary part, if $x(t)$ is real).

We know we can compute the coefficients x_k of the DTFS using the formula

$$x_k = \frac{1}{N}\sum_{n=0}^{N-1} x\left(\frac{n}{N}T\right) e^{-j2\pi nk/N} \tag{6.35}$$

Therefore, we can use this formula to compute the Fourier series coefficients x_k of $x(t)$ from the samples $x(\frac{n}{N}T)$ of a single period of $x(t)$.

The requirement $N \geq (2K+1)$ is actually a restatement of the sampling theorem. It is necessary to sample more than $2K$ times per period to ensure the number of samples $x(\frac{n}{N}T)$ of $x(t)$ exceeds the number $2K+1$ of unknowns x_k. A sampling rate of $2K=2FT$ samples per period of T sec. is equivalent to sampling $\frac{2FT}{T}=2F$ samples per sec., which is the Nyquist sampling rate; it is necessary to *exceed* this rate to ensure unique reconstruction.

6.2.2 APPLICATION: Interpreting Output of Matlab's "fft"

Suppose we are simply given the output of Matlab's `fft(X)` where `X` is a vector of samples of a signal $x(t)$ sampled at S samples per sec. How do we interpret a peak in `fft(X)` at a specific Matlab index `K`? This is a very common problem in engineering.

Matlab's `fft(X)` computes the DFT of the signal samples stored in `X`. As we will see in the next chapter, the DFT differs from the formula for DTFS coefficients by a factor of N=length of `X`. Let us regard `X` as a vector of samples of a single period of a periodic signal. Let `N=length(X)`. Then the period of the signal is $T=\frac{N}{S}$ sec., and the signal has a line spectrum of harmonics at $f=\frac{k}{T}=\frac{kS}{N}$ Hertz for integers k. The amplitudes and phases of the harmonics are `abs(fft(X))/N` and `angle(fft(X))`, respectively. But the second half of the output of `fft` is the complex conjugate of the first half, so *only the first half of the output of* `fft` *should be used to plot the one-sided spectrum.* The following example shows how to interpret the output of `fft`.

Example: Interpreting Output of `fft`.

A signal $x(t)$ is sampled at 1024 samples per sec. and stored in vector `X`. `fft(X)/4096` produces

$$[8, \underbrace{0\ldots0}_{31}, 2{+}2i, \underbrace{0\ldots0}_{95}, 6i, \underbrace{0\ldots0}_{3839}, -6i, \underbrace{0\ldots0}_{95}, 2{-}2i, \underbrace{0\ldots0}_{31}]$$

Assuming no aliasing has occurred, what is $x(t)$?

Solution:

The length is `N=length(X)`=4096 samples. The duration of the signal is $T=\frac{4096}{1024}$=4 sec. There are five peaks, at these indices: `K=[1 33 129 3969 4065]`. Note that indexing starts at `K=1`, but DFT indexing starts at k=0. Hence we must subtract one from the indices `K` to get DFT indices k. So x_k=0 except for these values:

- x_0=8;
- $x_{32}=x_{4064}^*=2+j2=2\sqrt{2}e^{j\pi/4}$.
- $x_{128}=x_{3968}^*=j6=6e^{j\pi/2}$.

The frequencies in Hertz are

$$\frac{k}{T} = k\frac{S}{N} = k\frac{1024}{4096} \rightarrow 32\frac{1024}{4096} = 8 \,\&\, 128\frac{1024}{4096} = 32. \tag{6.36}$$

The components at indices above 4096/2=2048 are discarded for the one-sided line spectrum.

The signal $x(t)$ is therefore

$$x(t) = 8 + 4\sqrt{2}\cos(2\pi 8t + \pi/4) + 12\cos(2\pi 32t + \pi/2). \quad (6.37)$$

The Matlab program for this example:

```
t=[0:1/1024:4-1/4096];
X=8+4*sqrt(2)*cos(16*pi*t+pi/4);
X=X+12*cos(64*pi*t+pi/2);
FX=fft(X)/4096;
K=find(abs(FX)>0.000001),FX(K)
```

Example: Computing Fourier series.

$x(t)$ is sampled at 50 samples/sec. We know:
(1) the signal is periodic with period 0.2 sec.;
(2) its maximum frequency is less than 25 Hertz;
(3) ten samples $\{x(0.02n), n = 0 \ldots 9\}$ in one period. Note that n=10 would give $x(0.02(10))$=$x(0.2)$=$x(0)$ since the signal has period 0.2 sec. and $(0.2)(50)$=10. The goal is to compute, entirely from its samples, the Fourier series expansion of the periodic signal $x(t)$

Solution:

T=0.2 sec. implies the signal has harmonics at 0,5,10,15,20 Hertz. There are no harmonics at higher frequencies, since these would not be less than 25 Hertz. There are 10 samples, so N=10. The DTFS coefficients formula is used to compute $\{x_k\}$ from the 10 samples. To list them, run the Matlab program.

The magnitude is plotted in the following plot. A stem plot is used since the $\{x_k\}$ are functions of integers k. This is the 2-sided magnitude line spectrum of $x(t)$ ($|x_k|$ vs. $\frac{k}{T}$). The actual computed $\{x_k\}$ are

```
[1 -2i 4+3i 0...]
```

These values give the magnitudes of the components as 1, $|-2i|$=2, $|4+3i|$=5, and 0. These are the heights of the lines in the figure. These values also give the phases of the components as $-\frac{\pi}{2}$ and $\arctan(\frac{3}{4})$=0.6435. The signal $x(t)$ is thus

$$x(t) = 1 + 4\sin(2\pi 5t) + 10\cos(2\pi 10t + 0.6435). \quad (6.38)$$

The signal is actually bandlimited to 10 Hertz, not to just less than 25 Hertz, but we did not know this.

The Matlab code used for this example:

```
N=10;T=0.2;t=[0:T/N:T-T/N];
```

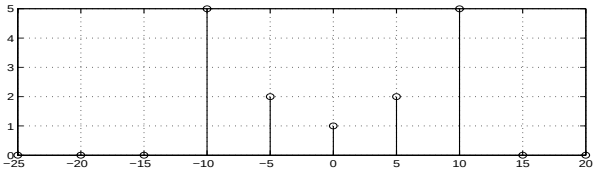


Figure 6.2: DTFS computation for example.

```
X=1+4*sin(2*pi*5*t);
X=X+10*cos(2*pi*10*t+0.6435);
F=[-(N/2)/T:1/T:(N/2-1)/T];
FX=fft(X)/N;stem(F,fftshift(abs(FX)))
```

Example: Spectrum of actual trumpet.

The file `trumpet.mat` contains the sound of an actual trumpet playing a single note sampled at the standard CD sampling rate of 44100 samples per sec. Compute its spectrum.

Solution:

The following Matlab program can be used for any sampled signal. It plots up to frequency $\mathtt{S}\frac{\mathtt{M}}{\mathtt{N}}$ Hertz.

```
load trumpet.mat;
S=44100;N=length(X);M=N/8;
F=[0:N-1]*S/N;FX=abs(fft(X));
plot(F(1:M),FX(1:M)/N)
```

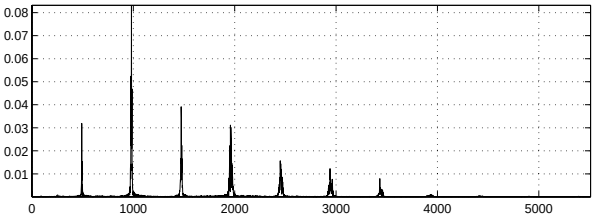


Figure 6.3: Spectrum of actual trumpet signal.

The fundamental frequency is just under 500 Hertz; it is actually note B at 494 Hertz. The line spectrum is quite evident, although there is some broadening of the peaks. This phenomenon, called *spectral leakage*, is explained in the chapter on data windows.

6.3 The Discrete-Time Fourier Transform (DTFT)

The Discrete-Time Fourier Transform (DTFT) is the discrete-time counterpart to the Continuous-Time Fourier Transform (CTFT). The major difference between the DTFT and CTFT is the DTFT is a periodic function of frequency ω, since in discrete time ω is periodic with period 2π. The DTFT can also be regarded as the Fourier dual of the CTFS.

The DTFT plays two major roles in DSP:

• The frequency response of an LTI system is the DTFT of its impulse response $h[n]$.

• The DTFT computes the *spectrum* of a discrete-time *non*-periodic signal.

6.3.1 DTFT Basics and Properties

DTFT Definition

The DTFT $X(e^{j\omega})$ of a discrete-time signal $x[n]$ is

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}; X(j\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt. \tag{6.39}$$

The analogy between the DTFT $X(e^{j\omega})$ and the CTFT $X(j\Omega)$ is evident. In fact, we have

$$\begin{aligned} x(t) &= \sum_{n=-\infty}^{\infty} x[n]\delta(t-n) \rightarrow \\ X(j\Omega) &= \sum_{n=-\infty}^{\infty} x[n]F[\delta(t-n)] \\ &= \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n} = X(e^{j\Omega}). \end{aligned} \tag{6.40}$$

DTFT is just the CTFT of the continuous-time sum-of-impulses $\sum_{n=-\infty}^{\infty} x[n]\delta(t-n)$.

DTFT	CTFT
$X(e^{j\omega})=\sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$	$X(j\Omega)=\int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt$
$x[n]=\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}\frac{d\omega}{2\pi}$	$x(t)=\int_{-\infty}^{\infty} X(j\Omega)e^{j\Omega t}\frac{d\Omega}{2\pi}$
$X(e^{j\omega}) = X(z)\|_{z=e^{j\omega}}$	$X(j\Omega) = X(s)\|_{s=j\Omega}$
dtft$(x[n-N])=X(e^{j\omega})e^{-j\omega N}$	$F(x(t-T))=X(j\Omega)e^{-j\Omega T}$
dtft$(x[n]e^{j\omega_o n})=X(e^{j(\omega-\omega_o)})$	$F(x(t)e^{j\Omega_o t})=X(j(\Omega-\Omega_o))$
$X(e^{j\omega})$ periodic	$X(j\Omega)$ not periodic

DTFT Properties

As a direct result of (6.40), most of the properties of the CTFT also hold for the DTFT. In particular:

- $X(e^{j\omega})$ defined if $x[n]$ is absolutely summable. $X(j\Omega)$ defined if $x(t)$ is absolutely integrable. We skip this to define the DTFT of a sinusoid.
- If $x[n]$ is real-valued, by conjugate symmetry: $X(e^{j\omega})=X(e^{-j\omega})^*$; $X(j\Omega)=X(-j\Omega)^*$.
- If $x[n]$ is real-valued and an even function, then $X(e^{j\omega})$ is also real-valued and even.
- $h[n]$ is impulse response of an LTI system, $H(e^{j\omega})$ is its frequency response function.
- dtft$(x[n]*y[n])=X(e^{j\omega})Y(e^{j\omega})$. F$(x(t)*y(t))=X(j\Omega)Y(j\Omega)$.
- dtft$(x[n-N]) = X(e^{j\omega})e^{-j\omega N}$. F$(x(t-T)) = X(j\Omega)e^{-j\Omega T}$. This time delay property will be used in the chapter on FIR filter design.
- dtft$(x[n]e^{j\omega_o n}) = X(e^{j(\omega-\omega_o)})$. Note that this is periodic in ω_o. F$(x(t)e^{j\Omega_o t}) = X(j(\Omega-\Omega_o))$.

6.3.2 Simple DTFT Computations

The easiest way to compute the DTFT is to compute the two-sided z-transform and then set $z=e^{j\omega}$. For finite duration signals, a useful formula is:

$$\begin{aligned} &Ae^{j\omega} + Be^{-j\omega} = \\ &(A+B)\cos(\omega) + j(A-B)\sin(\omega). \end{aligned} \tag{6.41}$$

Examples: Computing simple DTFTs

Compute the DTFT of the following:

- $\{1, 3, \underline{5}, 3, 1\}$; $\{3, 1, \underline{4}, 2, 5\}$; $(\frac{1}{2})^n u[n]$

Solution: Use $X(e^{j\omega})=X(z)|_{z=e^{j\omega}}$.

$$\begin{aligned} &\text{dtft}(\{1, 3, \underline{5}, 3, 1\}) = \\ &1e^{j2\omega} + 3e^{j\omega} + 5 + 3e^{-j\omega} + 1e^{-j2\omega} \\ &= 2\cos(2\omega) + 6\cos(\omega) + 5. \end{aligned} \tag{6.42}$$

Note that the DTFT of this real-valued and even function is real-valued and even, as expected.

$$\begin{aligned}&\text{dtft}(\{3,1,\underline{4},2,5\}) = \\ &3e^{j2\omega}+1e^{j\omega}+4+2e^{-j\omega}+5e^{-j2\omega} = \\ &4+8\cos(2\omega)+3\cos(\omega)-j2\sin(2\omega)-j\sin(\omega).\end{aligned} \tag{6.43}$$

Note that the real part is an even function, and the imaginary part is an odd function, as expected.

$$\begin{aligned}\text{dtft}((1/2)^n u[n]) &= \sum_{n=0}^{\infty}(1/2)^n e^{-j\omega n} \\ =\sum_{n=0}^{\infty}(\frac{1}{2e^{j\omega}})^n &= \frac{1}{1-\frac{1}{2e^{j\omega}}} = \frac{e^{j\omega}}{e^{j\omega}-\frac{1}{2}}\end{aligned} \tag{6.44}$$

For sinusoids, we have (compare to CTFT)

$$\begin{aligned}&\text{dtft}(A\cos(\omega_o n+\theta)) = \\ &A\pi e^{j\theta}\delta((\omega-\omega_o))+A\pi e^{-j\theta}\delta((\omega+\omega_o)) \\ &F(A\cos(\Omega_o t+\theta)) = \\ &A\pi e^{j\theta}\delta(\Omega-\Omega_o)+A\pi e^{-j\theta}\delta(\Omega+\Omega_o).\end{aligned} \tag{6.45}$$

$\delta((\omega))=\sum_{k=-\infty}^{\infty}\delta(\omega-2\pi k)$ (repeating every 2π).

Technically, both the CTFT and DTFT are undefined for pure sinusoids, since these are not absolutely integrable or summable. But in engineering, it is convenient to define these transforms using impulses.

Example: DTFT of $4\cos(0.15\pi n+1)$.

Solution: Plugging into the above formula, $4\pi e^{j1}\delta((\omega–0.15))+4\pi e^{-j1}\delta((\omega+0.15))$.

Example: DTFT of $4\sin(0.3n)$.

Solution: Writing this out in more detail

$$\begin{aligned}&\text{dtft}(4\sin(0.3n)) = \text{dtft}((2/j)e^{j0.3n}-(2/j)e^{-j0.3n}) \\ &=4\pi/j\sum_{k=-\infty}^{\infty}(\delta(\omega-0.3-2\pi k)-\delta(\omega+0.3-2\pi k)) \\ &=j4\pi\delta((\omega+0.3))-j4\pi\delta((\omega-0.3)).\end{aligned} \tag{6.46}$$

where $\delta((\omega))$ repeats in ω every 2π.

We could also set $\theta=-\frac{\pi}{2}$ in the formula above.

The relation between the DTFT and the two-sided z-transform is analogous to the relation between the CTFT and the two-sided Laplace transform:

$$X(e^{j\omega}) = X(z)|_{z=e^{j\omega}}; X(j\Omega) = X(s)|_{s=j\Omega}. \tag{6.47}$$

Both relations follow immediately from the definitions of the two-sided Z and Laplace transforms.

Both transforms also relate the frequency response function to the impulse response:

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}; H(j\Omega) = \int_{-\infty}^{\infty} h(t)e^{-j\Omega t}dt. \tag{6.48}$$

This is useful in computing impulse response directly from frequency response using inverse transforms.

6.3.3 Inverse DTFT

Still another connection between discrete time and continuous time comes from noting that the DTFT $X(e^{j\omega})$ is periodic with period 2π, since $e^{j(\omega+2\pi k)}=e^{j\omega}$ for any integer k in the definition (6.39) of the DTFT. Then (6.39) can be viewed as a CTFS expansion of the periodic function $X(e^{j\omega})$:

The DTFT is the Fourier *dual* of the CTFS.

So the inverse DTFT is simply the formula for computing the CTFS coefficients, with $T=2\pi$:

$$x[n] = \int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}\frac{d\omega}{2\pi}; x(t) = \int_{-\infty}^{\infty} X(j\Omega)e^{j\Omega t}\frac{d\Omega}{2\pi}. \tag{6.49}$$

Finite-Duration Signals

The following example shows how to compute the inverse DTFT if it has finite duration.

Example: Compute the inverse DTFT of $4\cos(2\omega)+6\cos(\omega)+j8\sin(2\omega)+j2\sin(\omega)$.

Solution:
Using the two useful identities

$$2\cos(\omega) = e^{j\omega}+e^{j\omega}; 2j\sin(\omega) = e^{j\omega}-e^{j\omega} \tag{6.50}$$

we can rewrite the above as

$$2[e^{j2\omega}+e^{-j2\omega}] \quad + \quad 3[e^{j\omega}+e^{-j\omega}]+$$

$$\begin{aligned} 4[e^{j2\omega} - e^{-j2\omega}] &+ 1[e^{j\omega} - e^{-j\omega}] = \\ (2+4)e^{j2\omega} + (2-4)e^{-j2\omega} &+ (3+1)e^{j\omega} + (3-1)e^{-j\omega} \\ \text{Inverse dtft} &\rightarrow \{6, 4, \underline{0}, 2, -2\}. \end{aligned} \quad (6.51)$$

No constant term, so the inverse DTFT is 0 at n=0.

Orthogonality of $\{e^{j\omega n}, n \in \text{integers}\}$

The inverse DTFT can also be derived using the orthogonality of the basis functions $\{e^{j\omega n}, n \in \text{integers}\}$ over the interval $-\pi < \omega < \pi$, as follows.

First, we derive the orthogonality of $\{e^{j\omega n}\}$.

$$\begin{aligned} &\int_{-\pi}^{\pi} e^{j\omega m} e^{-j\omega n} d\omega = \int_{-\pi}^{\pi} e^{j\omega(m-n)} d\omega \\ &= \frac{e^{j\omega(m-n)}}{j\omega(m-n)}|_{-\pi}^{\pi} = \frac{e^{j\pi(m-n)}}{j\pi(m-n)} - \frac{-e^{j\pi(m-n)}}{-j\pi(m-n)} \\ &= \frac{1}{j\pi(m-n)} - \frac{1}{j\pi(m-n)} = 0 \text{ if } m \neq n. \end{aligned} \quad (6.52)$$

If $m = n$ then

$$\int_{-\pi}^{\pi} e^{j\omega m} e^{-j\omega n} d\omega = \int_{-\pi}^{\pi} d\omega = 2\pi. \quad (6.53)$$

So the basis functions $\{e^{j\omega n}, n \in \text{integers}\}$ are orthogonal over the interval $-\pi < \omega < \pi$.

Multiplying the definition of DTFT (6.39) by $e^{j\omega m}$ and integrating over ω from $-\pi$ to π gives

$$\int_{-\pi}^{\pi} X(e^{j\omega}) e^{j\omega m} d\omega = \int_{-\pi}^{\pi} \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n} e^{j\omega m} d\omega. \quad (6.54)$$

Exchanging the order of summation and integration

$$\int_{-\pi}^{\pi} X(e^{j\omega}) e^{j\omega m} d\omega = \sum_{n=-\infty}^{\infty} x[n] \int_{-\pi}^{\pi} e^{-j\omega n} e^{j\omega m} d\omega \quad (6.55)$$

and using the orthogonality of the $\{e^{j\omega n}\}$ gives

$$\int_{-\pi}^{\pi} X(e^{j\omega}) e^{j\omega m} d\omega = \sum_{n=-\infty}^{\infty} x[n] 2\pi\delta[m-n] = 2\pi x_m. \quad (6.56)$$

Dividing by 2π gives (6.49).

Ideal Differentiator

Recall that in continuous time the ideal differentiator has frequency response

$$y = dx/dt \rightarrow H(j\Omega) = Y(j\Omega)/X(j\Omega) = j\Omega. \quad (6.57)$$

In discrete time, we use the periodic extension of this:

$$H(e^{j\omega}) = ((j\omega)). \quad (6.58)$$

The formula for inverse DTFT requires $H(e^{j\omega})$ only for $|\omega| < \pi$. The impulse response for the ideal differentiator in discrete time is, using (6.49),

$$\begin{aligned} h[n] &= \int_{-\pi}^{\pi} H(e^{j\omega}) e^{j\omega n} \frac{d\omega}{2\pi} \\ &= \int_{-\pi}^{\pi} (j\omega) e^{j\omega n} \frac{d\omega}{2\pi} \\ &= \begin{cases} \frac{(-1)^n}{n} & n \neq 0 \\ 0 & n = 0 \end{cases} \\ &= \{\ldots \frac{1}{3}, -\frac{1}{2}, 1, 0, -1, \frac{1}{2}, -\frac{1}{3} \ldots\} \end{aligned} \quad (6.59)$$

This means that the finite-difference approximation to differentiation can be written as

$$\begin{aligned} y[n] &= x[n+1] - x[n-1] \\ &- \frac{1}{2}(x[n+2] - x[n-2]) \\ &+ \frac{1}{3}(x[n+3] - x[n-3]) - \ldots \end{aligned} \quad (6.60)$$

The first two terms are the (doubled) central difference approximation to differentiation.

We will reuse this result in designing FIR filters. We have work to do there, since the above $h[n]$ is:

- Non-causal ($h[n] \neq 0$ for $n < 0$);
- Decays too slowly ($\simeq \frac{1}{n}$) to realize with delays;
- Not BIBO-stable (from (2.24), $h[n]$ is not absolutely summable).

Note that since $H(e^{j\omega})$=$((j\omega))$ is pure imaginary and odd, its inverse DTFT is real-valued and odd.

6.3.4 Discrete Sinc Functions

Discrete sinc functions arise as

- Impulse response of a brick-wall low-pass filter;
- Impulse response of a brick-wall band-pass filter;
- The DTFT of a rectangular pulse in time.

The first two are identical to continuous-time (except t becomes n). The third is now different.

Brick-Wall Low-Pass Filter

We define the discrete-time *brick-wall low-pass filter* with cutoff frequency ω_o to have frequency response

$$H(e^{j\omega}) = \begin{cases} 1, & 0 \le |\omega| \le \omega_o \\ 0, & \omega_o < |\omega| \le \pi \end{cases} \tag{6.61}$$

where ω is reduced mod 2π to interval $-\pi < \omega \le \pi$, since $H(e^{j\omega})$ is periodic in ω with period 2π.

The formula for inverse DTFT requires $H(e^{j\omega})$ only for $|\omega| < \pi$. The impulse response for the ideal low-pass filter in discrete time is, using (6.49),

$$h[n] = \frac{1}{2\pi}\int_{-\omega_o}^{\omega_o} e^{j\omega n} d\omega = \frac{\sin(\omega_o n)}{\pi n}. \tag{6.62}$$

Note that $h[n]$ has the same form as $h(t)=\frac{\sin(\omega_o t)}{\pi t}$ for the continuous-time brick-wall low-pass filter.

Brick-Wall Band-Pass Filter

The discrete-time *brick-wall band-pass filter* with cutoff frequencies ω_L and ω_H has frequency response

$$H(e^{j\omega}) = \begin{cases} 0, & 0 \le |\omega| < \omega_L \\ 1, & \omega_L < |\omega| < \omega_H \\ 0, & \omega_H < |\omega| \le \pi \end{cases} \tag{6.63}$$

where ω is reduced mod 2π to interval $-\pi < \omega \le \pi$, since $H(e^{j\omega})$ is periodic in ω with period 2π.

Using the modulation property of the DTFT, the impulse response for the ideal band-pass filter is

$$h[n] = \frac{\sin(\omega_d n)}{\pi n} 2\cos(\omega_c n); \begin{array}{l} \omega_c = (\omega_H + \omega_L)/2 \\ \omega_d = (\omega_H - \omega_L)/2 \end{array}. \tag{6.64}$$

$h[n]$ has the same form as the continuous-time brick-wall band-pass filter.

Rectangular Pulse

The spectrum $X(e^{j\omega})$ of the rectangular pulse

$$x[n] = \begin{cases} 1, & |n| \le N \\ 0, & |n| > N \end{cases} \tag{6.65}$$

which has length $2N+1$ can be computed as

$$X(e^{j\omega}) = \sum_{n=-N}^{N} e^{-j\omega n} = e^{j\omega N}\sum_{n=0}^{2N} e^{-j\omega n}. \tag{6.66}$$

Setting $r=e^{-j\omega}$ and $M=2N+1$ in the formula

$$\sum_{n=0}^{M-1} r^n = \frac{1-r^M}{1-r}, \quad r \ne 1 \tag{6.67}$$

shows that the summation simplifies to

$$\begin{aligned} X(e^{j\omega}) &= \left[\frac{e^{j\omega(N+\frac{1}{2})}}{e^{j\omega\frac{1}{2}}}\right]\frac{1-e^{-j\omega(2N+1)}}{1-e^{-j\omega}} \\ &= \frac{e^{j\omega(N+\frac{1}{2})} - e^{-j\omega(N+\frac{1}{2})}}{e^{j\omega\frac{1}{2}} - e^{-j\omega\frac{1}{2}}} \\ &= \frac{\sin[\omega(N+\frac{1}{2})]}{\sin[\omega(\frac{1}{2})]}. \end{aligned} \tag{6.68}$$

$X(e^{j\omega})$ is called a *discrete sinc* function. It is plotted

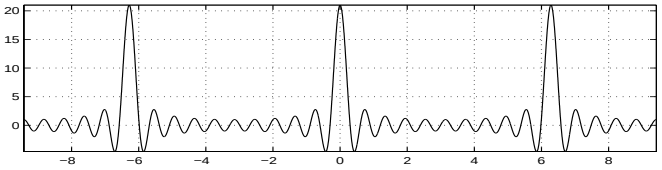


Figure 6.4: Discrete Sinc Function for $N = 10$.

in the above figure for N=10. Note that $X(e^{j\omega})$ looks like a sinc function for $\omega << 1$, but is still periodic with period 2π. As $\omega \to 0$, $\sin(\omega) \simeq \omega$ implies

$$X(e^{j\omega}) \simeq \frac{\omega(N+\frac{1}{2})}{\omega(\frac{1}{2})} = 2N+1 = \sum_{n=-\infty}^{\infty} x[n]. \tag{6.69}$$

Note the maximum value in the figure is 2(10)+1=21.

Chapter 7

Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT)

The Discrete Fourier Transform (DFT) is related to both the DTFS and DTFT, but it has no continuous-time counterpart. It can be regarded as either: (1) the result of sampling the DTFT in ω; or (2) a single period of the DTFS. It can also be regarded as the result of sampling the z-transform at equally-spaced points on the unit circle. All of these interpretations will be used in this chapter.

7.1 Definition of DFT

The *N-point discrete Fourier transform (DFT)* and its inverse (IDFT) are defined as

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n}, k = 0 \ldots N-1. \\ x[n] &= \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j2\pi \frac{k}{N} n}, n = 0 \ldots N-1 \end{aligned} \quad (7.1)$$

The N-point DFT is also called a DFT of order N. If $x[n]$ has duration $L < N$, it is *zero-padded* to length N by appending N-L zeros to it, to give

$$x[n] = \begin{cases} x[n], & 0 \le n \le L-1 \\ 0, & L \le n \le N-1 \end{cases} \quad (7.2)$$

Comparing (7.1) to the definition (6.1) of the DTFS and the formula (6.12) for computing its coefficients x_k shows that they are identical, except for the placement of the factor $\frac{1}{N}$. So $X_k = N x_k$.

This begs the question of why we need yet another transform. The answer is: (1) *interpretation* of the transform: the DFT is a linear transformation from a set of numbers $\{x[0] \ldots x[N-1]\}$ to another set $\{X_0 \ldots X_{N-1}\}$; and (2) *computation* of the transform (see the section on FFT). (7.1) can be viewed as a matrix-times-vector computation and its inverse.

More significant is the relation between the DFT and the DTFT and z-transform, which is:

$$X_k = X(e^{j\omega})|_{\omega = \frac{2\pi k}{N}} = X(z)|_{z = e^{j2\pi k/N}}. \quad (7.3)$$

So the DFT is the result of sampling the DTFT at N equally-spaced frequencies between 0 and 2π. This makes the DFT ideal for computing the DTFT numerically. The DFT is also the result of sampling the z-transform at N equally-spaced points around the unit circle $|z|=1$. This allows the DFT to be computed quickly using a divide-and-conquer algorithm called the fast Fourier transform (FFT), which we derive below. The forms (7.1) of the DFT and its inverse also make it ideal for numerical computation of the continuous-time Fourier transform, which we discuss below. To summarize:

Relation between DFT and DTFS: $X_k = N x_k$.
Relation between DFT, DTFT, z-transform: $X_k = X(e^{j\omega})\|_{\omega = \frac{2\pi k}{N}} = X(z)\|_{z = e^{j2\pi k/N}}$.

7.2 Properties of DFT

Properties of the DFT are almost identical to those of the DTFS (they are identical to a factor of $\frac{1}{N}$):

- If $x[n]$ is real-valued, then $X_{N-k}=X_k^*$. This is conjugate symmetry in the DFT.
- $X_0=\sum_{n=0}^{N-1} x[n]$ is the *sum* of the $x[n]$.
- If N is even, then the component at $\omega=\pi$ is $X_{N/2}=x[0]–x[1]+x[2]–x[3]+\ldots–x[n–1]$.
- *Parseval's theorem* for the DFT states that the *energy* of $x[n]$ is the same when computed in either the time or the frequency domain, i.e., $\sum_{n=0}^{N-1}|x[n]|^2=\frac{1}{N}\sum_{k=0}^{N-1}|X_k|^2$.

The following properties of the DFT are true since the DFT is the DTFT computed at samples. Throughout, $((n))$ means n reduced mod(N)

- If $h[n]$ is the impulse response of an LTI system, H_k is $H(e^{j\omega})$ sampled at $\omega=\frac{2\pi k}{N}$.
- $\mathrm{DFT}(x[((n–n_o))])=X_k e^{-j2\pi k n_o/N}$ for any n_o. This is a *circular shift* of $x[n]$. For example, $x[n]=\{1,2,3,4\}\rightarrow x[((n–2))]=\{3,4,1,2\}$.
- $\mathrm{DFT}(x[n]e^{j2\pi k_o n/N})=X_{((k-k_o))}$ as expected, since $X(e^{j\omega})$ is periodic with period 2π implies X_k is periodic with period N, since $\omega=\frac{2\pi}{N}k$. This modulation property will be important in deriving the FFT later in this chapter.

7.2.1 DFT of Periodic Sinusoids

$$\begin{aligned}\mathrm{DFT}(e^{j2\pi\frac{k_o}{N}n}) &= \sum_{n=0}^{N-1} e^{j2\pi\frac{k_o}{N}n}e^{-j2\pi\frac{k}{N}n}\\ =\sum_{n=0}^{N-1} e^{-j2\pi\frac{k-k_o}{N}n} &= N\delta[k-k_o]. \qquad (7.4)\end{aligned}$$

Then the DFT of a periodic sinusoid is

$$\begin{aligned}&\mathrm{DFT}(A\cos(2\pi\frac{k_o}{N}n+\theta)) \qquad (7.5)\\ &=\frac{A}{2}e^{j\theta}\mathrm{DFT}(e^{j2\pi\frac{k_o}{N}n}+\frac{A}{2}e^{-j\theta}\mathrm{DFT}(e^{-j2\pi\frac{k_o}{N}n}\\ &=\frac{A}{2}e^{j\theta}N\delta[((k-k_o))]+\frac{A}{2}e^{-j\theta}N\delta[((k+k_o))]\end{aligned}$$

where $((k\pm k_o))$ means $(k\pm k_o)$ reduced mod N.

The DFT of a periodic sinusoid is the DTFS of a periodic sinusoid multiplied by N. As $N\rightarrow\infty$, the pair of discrete impulses becomes a pair of continuous impulses, which is the DTFT of a sinusoid.

7.2.2 DFT of Convolution

To see what time-domain operation the DFT maps to multiplication, we compute the inverse DFT of X_kY_k.

Substituting the definition (7.1) of the DFT gives

$$\begin{aligned}\mathrm{DFT}^{-1}[X_kY_k] &= \frac{1}{N}\sum_{k=0}^{N-1}(X_kY_k)e^{\frac{j2\pi kn}{N}} \qquad (7.6)\\ &=\frac{1}{N}\sum_{k=0}^{N-1}e^{\frac{j2\pi kn}{N}}\sum_{n_1=0}^{N-1}x[n_1]e^{-\frac{j2\pi kn_1}{N}}\left[\sum_{n_2=0}^{N-1}y[n_2]e^{-\frac{j2\pi kn_2}{N}}\right]\end{aligned}$$

Rearranging the order of (finite) summations gives

$$\mathrm{DFT}^{-1}[X_kY_k]=\frac{1}{N}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}x[n_1]y[n_2]\sum_{k=0}^{N-1}e^{\frac{j2\pi k}{N}(n-n_1-n_2)}. \qquad (7.7)$$

The orthogonality (6.10) of the $\{e^{j2\pi\frac{k}{N}n}\}$ gives

$$\begin{aligned}\mathrm{DFT}^{-1}[X_kY_k] &= \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}x[n_1]y[n_2]\delta[((n-n_1-n_2))]\\ &=\sum_{n_1=0}^{N-1}x[n_1]y[((n-n_1))]. \qquad (7.8)\end{aligned}$$

where $((n-n_1))$ means $(n-n_1)$ reduced mod N.

This is *not* the convolution of $x[n]$ and $y[n]$, due to the reduction mod N. This is called a *circular* or *cyclic* convolution. We can convert this into a convolution by choosing the transform order N to be the duration of the convolution $x[n]*y[n]$ (not the duration of $x[n]$ or $y[n]$) and *zero-padding* $x[n]$ and $y[n]$ by extending them with zeros so they both have duration N. Then the *zero-padded* DFT can be used to compute convolutions. Using the FFT, this is a way to compute convolutions quickly.

7.2.3 DFT Examples

Example: DFT Computation.

Compute the 4-point DFT of $\{24, 8, 12, 16\}$.

Solution:
Noting that $e^{-j2\pi nk/4}=(-j)^{nk}$, (7.1) gives

$$\begin{aligned} X_0 &= 24(1)+8(1)+12(1)+16(1)=60. \quad (7.9)\\ X_1 &= 24(1)-8(j)-12(1)+16(j)=12+j8.\\ X_2 &= 24(1)-8(1)+12(1)-16(1)=12.\\ X_3 &= 24(1)+8(j)-12(1)-16(j)=12-j8. \end{aligned}$$

This is the result of a DTFS example multiplied by N=4. This result can be checked using Matlab:

```
fft([24 8 12 16])
```

Example: DFT of sinusoid.
Compute the 8-point DFT of $\{4\cos(\frac{\pi}{2}n+1)\}$. Note this sinusoid has period N=8.

Solution:
Using the formula (7.5) for the DFT of a sinusoid,

$$X_k = \begin{cases} 16e^{j1}, & k=2\\ 16e^{-j1}, & k=6\\ 0 & \text{otherwise} \end{cases}. \quad (7.10)$$

This result can be checked using Matlab:

```
fft(4*cos(pi/2*[0:7]+1))
```

Example: Convolution using DFT
Use the DFT to compute $\{\underline{4}, 5\} * \{\underline{1}, 2, 3\}$.

Solution:
The duration of the result of the convolution is 2+3–1=4, so we use N=4 and zero-pad the two signals to be convolved to have durations four each.

The 4-point DFT of $x[n]$=$\{4, 5, 0, 0\}$ is

$$\begin{aligned} X_0 &= 4(1)+5(1)+0(1)+0(1)=9. \quad (7.11)\\ X_1 &= 4(1)-5(j)-0(1)+0(j)=4-j5.\\ X_2 &= 4(1)-5(1)+0(1)-0(1)=-1.\\ X_3 &= 4(1)+5(j)-0(1)-0(j)=4+j5. \end{aligned}$$

The 4-point DFT of $y[n]$=$\{1, 2, 3, 0\}$ is

$$\begin{aligned} Y_0 &= 1(1)+2(1)+3(1)+0(1)=6. \quad (7.12)\\ Y_1 &= 1(1)-2(j)-3(1)+0(j)=-2-j2.\\ Y_2 &= 1(1)-2(1)+3(1)-0(1)=2.\\ Y_3 &= 1(1)+2(j)-3(1)-0(j)=-2+j2. \end{aligned}$$

Multiplying these gives

$$\begin{aligned} X_0Y_0 &= (9)(6)=54. \quad (7.13)\\ X_1Y_1 &= (4-j5)(-2-2j)=-18+j2.\\ X_2Y_2 &= (-1)(2)=-2.\\ X_3Y_3 &= (4+j5)(-2+j2)=-18-j2. \end{aligned}$$

Finally, the inverse DFT of $\{54,$–18+$j2, 2,$–18–$j2\}$ is $\{4, 13, 22, 15\}$ which is indeed $\{\underline{4}, 5\} * \{\underline{1}, 2, 3\}$.

This result can be checked using Matlab:

```
ifft(fft([1 2 3 0]).* fft([4 5 0 0]))
```

This is a lot of work to compute a trivial convolution. But in fact the DFT can be computed rapidly using the FFT, as we show next.

7.3 Fast Fourier Transform (FFT): Radix 2

7.3.1 Terminology

The fast Fourier transform (FFT) is a divide-and-conquer fast *algorithm* for computing the DFT, which is a *transform*. This should be specified explicitly as:

Fast Fourier Transform (FFT) is an algorithm to compute Discrete Fourier Transforms (DFT).

Sometimes people incorrectly say they wish to "compute the FFT of a given data set." This is like saying one wishes to "Gaussian eliminate a linear system of equations." Correct statements are: (1) one wishes to compute the solution to a system of equations *using* Gaussian elimination; (2) one wishes to compute the DFT of a data set *using* the FFT.

We now show how to compute an 16-point DFT using two 8-point DFTs and 8 multiplications and additions (MADs). This *divides* the 16-point DFT into two 8-point DFTs, which in turn can be *divided* into four 4-point DFTs, which are just additions and subtractions. This *conquers* the 16-point DFT by *dividing* it into 4-point DFTs and additional MADs.

7.3.2 Dividing a 16-Point DFT

Recall that the 16-point DFT is the z-transform evaluated at 16 equally-spaced points on the unit circle. This is shown in the figure below, in which 8 points are depicted by O's and the other 8 points are depicted by X's.

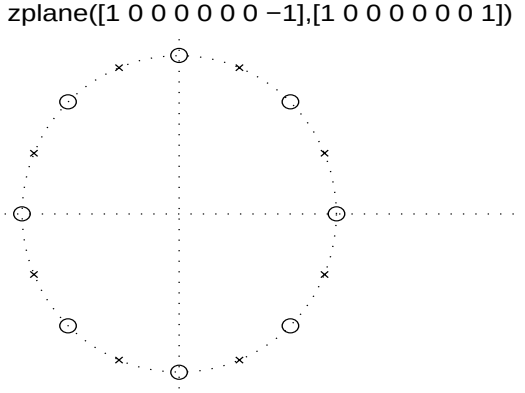


Figure 7.1: An 16-point DFT is $X(z)$ evaluated at 16 points equally spaced on the unit circle.

But the 8 points depicted by O's are the points at which $X(z)$ is evaluated for an 8-point DFT. And the 8 points depicted by X's are the points at which $X(z)$ is evaluated for a *modulated* 8-point DFT. So the 16-point DFT can be computed as an 8-point DFT (for even k) and as a modulated 8-point DFT (for odd k).

Computation at Even Indices

We consider even and odd indices k separately.

For even k, we can write k=$2k'$ and split the 16-point DFT summation into two summations:

$$\begin{aligned} X_{2k'} &= \sum_{n=0}^{15} x[n]e^{-j2\pi\frac{2k'}{16}n} \qquad (7.14)\\ &= \sum_{n=0}^{7} x[n]e^{-j2\pi\frac{2k'}{16}n} + \sum_{n=8}^{15} x[n]e^{-j2\pi\frac{2k'}{16}n}. \end{aligned}$$

Changing variables from n to n'=n–8 in the second summation, and recognizing $\frac{2k'}{16}$=$\frac{k'}{8}$ gives

$$\begin{aligned} X_{2k'} &= \sum_{n=0}^{7} x[n]e^{-j2\pi\frac{k'}{8}n} + \sum_{n=0}^{7} x[n+8]e^{-j2\pi\frac{k'}{8}(n+8)}\\ &= \sum_{n=0}^{7}(x[n]+x[n+8])e^{-j2\pi\frac{k}{8}n}\\ &= \mathrm{DFT}(\{x[n]+x[n+8], n=0,\ldots 7\}). \qquad (7.15) \end{aligned}$$

So for even k the 16-point DFT of $x[n]$ is the 8-point DFT of $\{x[n]$+$x[n$+$4]$, n=$0,\ldots 7\}$. The z-transform is computed at the O's of the figure above.

Computation at Odd Indices

For odd k, we can write k=$2k'$+1 and split the 16-point DFT summation into two summations:

$$\begin{aligned} X_{2k'+1} &= \sum_{n=0}^{15} x[n]e^{-j2\pi\frac{2k'+1}{16}n} \qquad (7.16)\\ &= \sum_{n=0}^{7} x[n]e^{-j2\pi\frac{2k'+1}{16}n} + \sum_{n=8}^{15} x[n]e^{-j2\pi\frac{2k'+1}{16}n} \end{aligned}$$

Changing variables from n to n'=n–8 in the second summation, and recognizing that $e^{-j2\pi\frac{8}{16}}$=–1 and $\frac{2k'+1}{16}$=$\frac{k'}{8}$+$\frac{1}{16}$ gives

$$\begin{aligned} X_{2k'+1} &= \sum_{n=0}^{7}(x[n]e^{-j2\pi\frac{1}{16}n})e^{-j2\pi\frac{k'}{8}n} \qquad (7.17)\\ &+ \sum_{n=0}^{7}(x[n+8]e^{-j2\pi\frac{1}{16}(n+8)})e^{-j2\pi\frac{k'}{8}(n+8)}\\ &= \sum_{n=0}^{7} e^{-j2\pi\frac{1}{16}n}(x[n]-x[n+8])e^{-j2\pi\frac{k'}{8}n}\\ &= \mathrm{DFT}(\{e^{-j2\pi\frac{1}{16}n}(x[n]-x[n+8])\}). \end{aligned}$$

So for odd k the 16-point DFT of $x[n]$ is the 8-point DFT of $\{e^{-j\frac{2\pi}{16}n}(x[n]$–$x[n$+$8])$,$n$=$0,\ldots 7\}$. The signal $\{x[n]$–$x[n$+$8]$,n=$0,\ldots 7\}$ has been *modulated* by multiplication by $e^{-j\frac{2\pi}{16}n}$. The modulation shifts the frequencies, so that the z-transform is computed at the X's of the figure above. The multiplications by $e^{-j\frac{2\pi}{8}n}$ are known as *twiddle multiplications* (mults) by the *twiddle factors* $e^{-j\frac{2\pi}{8}n}$.

Example: Dividing an 8-point DFT into 2 4-point DFTs.

Divide the 8-point DFT of $\{7, 1, 4, 2, 8, 5, 3, 6\}$ into two 4-point DFTs and twiddle mults.

Solution:

- For even indices k we have $X_{0,2,4,6}$=DFT($\{$7+8,1+5,4+3,2+6$\}$)= $\{36, 8+j2, 8, 8–j2\}$. We are half done! For odd indices, we need twiddle mults.
- The twiddle mults are by $\{e^{-j2\pi n/8}\}$ which are $\{1, \frac{\sqrt{2}}{2}(1–j), -j, \frac{\sqrt{2}}{2}(-1–j)\}$.
- Implementing the twiddle mults gives: $\{7–8, 1–5, 4–3, 2–6\}\times$ $\{1, \frac{\sqrt{2}}{2}(1–j), -j, \frac{\sqrt{2}}{2}(-1–j)\}=$ $\{-1, 2\sqrt{2}(-1+j), -j, 2\sqrt{2}(1+j)\}$.
- For odd indices k we have $X_{1,3,5,7}=$ DFT($\{-1, 2\sqrt{2}(-1+j), -j, 2\sqrt{2}(1+j)\}$)= $\{–1+j4.66, –1+j6.66, –1–j6.66, –1–j4.66\}$.

Combining these results for even and odd k gives

- DFT($\{7, 1, 4, 2, 8, 5, 3, 6\}$)=$\{36,$ $–1+j4.7, 8+j2, –1+j6.7, 8, –1–j6.7, 8–j2, –1–j4.7\}$. Note the conjugate symmetry in the second line.
- This agrees with direct computation using `fft([7 1 4 2 8 5 3 6])`

7.3.3 Dividing up a 2N-point DFT

Now we wish to compute a 2N-point DFT by dividing it into two N-point DFTs and N twiddle mults.

The above derivation generalizes directly–replace 8 with N and 16 with $2N$. The results are as follows:

For even indices $k = 2k'$ we have

$$\begin{aligned} X_{2k'} &= \sum_{n=0}^{N-1}(x[n] + x[n+N])e^{-j2\pi\frac{k'}{N}n} \qquad (7.18)\\ &= \text{DFT}\{x[n] + x[n+N], n = 0, 1 \ldots N-1\}. \end{aligned}$$

For odd indices $k = 2k' + 1$ we have

$$\begin{aligned} X_{2k'+1} &= \sum_{n=0}^{N-1} e^{-j2\pi\frac{1}{2N}n}(x[n] - x[n+N])e^{-j2\pi\frac{k'}{N}n}\\ &= \text{DFT}\{e^{-j2\pi\frac{1}{2N}n}(x[n] - x[n+N])\} \qquad (7.19) \end{aligned}$$

So a $2N$-point DFT can be divided into:

- Two N-point DFTs
- N multiplications by twiddle factors $e^{-j2\pi\frac{1}{2N}n}$
- $2N$ additions and subtractions.

7.3.4 Dividing and Conquering

Now suppose N *is a power of two*, e.g., N=1024=2^{10}. Then we can apply the above algorithm to divide an N-point DFT into two $\frac{N}{2}$-point DFTs, four $\frac{N}{4}$-point DFTs, eight $\frac{N}{8}$-point DFTs, and so on until we reach 4-point DFTs, which are just

$$\begin{aligned} X_0 &= x[0] + x[1] + x[2] + x[3]\\ X_1 &= x[0] - jx[1] - x[2] + jx[3]\\ X_2 &= x[0] - x[1] + x[2] - x[3]\\ X_3 &= x[0] + jx[1] - x[2] - jx[3] \qquad (7.20) \end{aligned}$$

At each stage, half of the DFTs are modulated, requiring $\frac{N}{2}$ multiplications. So if N is a power of two, then an N-point DFT computed using the FFT will require roughly $\frac{N}{2}\log_2(N)$ multiplications and $N\log_2(N)$ additions. These can be reduced slightly by recognizing that some mults are by ± 1 and $\pm j$.

To show the significance of the FFT, suppose we wish to compute a 32768-point DFT. Direct computation using the formula (7.1) would require $(32768)^2 \approx$1.1 billion MADS. But computation using the FFT would require less than $\frac{32768}{2}\log_2(32768) \approx$ 250 thousand MADs, a savings of a factor of 4000!

7.4 FFT: Cooley-Tukey

The Cooley-Tukey FFT was originally derived by Gauss. But its first implementation on computers (using punch cards and tape drives!) is due to Cooley and Tukey at IBM in 1965. The Cooley-Tukey FFT divides an (N_1N_2)-point DFT into:

- N_1 (N_2-point) DFTs, and
- N_2 (N_1-point) DFTs, and
- $(N_1–1)(N_2–1)$ twiddle mults.

The goal is to compute the N=N_1N_2-point DFT

$$X_k = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N}, k = 0, 1 \ldots N-1. \qquad (7.21)$$

7.4.1 Coarse and Vernier Indices

Define the *coarse and vernier indices*

- Coarse indices n_2 and k_1
- Vernier indices n_1 and k_2

as the quotients and remainders, respectively, after dividing n by N_1 and k by N_2:

$$\begin{aligned} n = n_1 + N_1 n_2 \quad &\text{for} \quad \begin{array}{l} n_1 = 0, 1 \ldots N_1 - 1 \\ n_2 = 0, 1 \ldots N_2 - 1 \end{array} \\ k = k_2 + N_2 k_1 \quad &\text{for} \quad \begin{array}{l} k_1 = 0, 1 \ldots N_1 - 1 \\ k_2 = 0, 1 \ldots N_2 - 1 \end{array} \end{aligned} \tag{7.22}$$

These names come from coarse and fine (vernier) adjustments for gauges and microscopes.

For example, let N_1=3 and N_2=2. Then

n	0	1	2	3	4	5
n_1	0	1	2	0	1	2
n_2	0	0	0	1	1	1
k	0	1	2	3	4	5
k_1	0	0	0	1	1	1
k_2	0	1	0	1	0	1

The DFT can be rewritten as

$$X_{k_2+N_2k_1} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1 + N_1 n_2] e^{-j\frac{2\pi}{N}nk}. \tag{7.23}$$

Now write nk in the DFT exponent as

$$\begin{aligned} nk &= (n_1 + N_1 n_2)(k_2 + N_2 k_1) \\ &= n_1 k_2 + N_1 n_2 k_2 + N_2 n_1 k_1 + N_1 N_2 n_2 k_1 \end{aligned} \tag{7.24}$$

and using N=N_1N_2 in the DFT exponent gives

$$\begin{aligned} e^{-j\frac{2\pi}{N}nk} &= e^{-j\frac{2\pi}{N}n_1k_2} e^{-j\frac{2\pi}{N}N_1n_2k_2} \\ &\times\ e^{-j\frac{2\pi}{N}N_2n_1k_1} e^{-j\frac{2\pi}{N}N_1N_2n_2k_1} \\ &= e^{-j\frac{2\pi}{N}n_1k_2} e^{-j\frac{2\pi}{N_2}n_2k_2} e^{-j\frac{2\pi}{N_1}n_1k_1}. \end{aligned} \tag{7.25}$$

Then the DFT can be rewritten as

$$\begin{aligned} X_{k_2+N_2k_1} &= \underbrace{\sum_{n_1=0}^{N_1-1} e^{-j\frac{2\pi}{N_1}n_1k_1}}_{\mathrm{N_1-point\ DFT}} [e^{-j\frac{2\pi}{N_1}n_1k_2} \\ &\times\ \underbrace{\sum_{n_2=0}^{N_2-1} e^{-j\frac{2\pi}{N_2}n_2k_2} x[n_1 + N_1 n_2]}_{\mathrm{N_2-point\ DFT\ for\ each\ n_1}}]. \end{aligned} \tag{7.26}$$

We have rewritten the N-point DFT as N_2 N_1-point DFTs, N_1 N_2-point DFTs, and twiddle mults.

7.4.2 Procedure

1. Compute N_1 (for each n_1) N_2-point DFTs of $x[n_1 + N_1 n_2]$.
2. Multiply result by *twiddle factors* $e^{-j\frac{2\pi}{N}n_1k_2}$.
3. Compute N_2 (for each k_2) N_1-point DFTs of the result. Done!

Note that if either n_1=0 or k_2=0, then $e^{-j\frac{2\pi}{N}n_1k_2}$=1. So the number of twiddle mults is not N_1N_2, but (N_1–1)(N_2–1). This saves a factor of two below.

7.4.3 2-D Visualization

We can visualize this by writing both $x[n]$ and X_k as $(N_1 \times N_2)$ arrays. This leads to this procedure:

1. Map $x[n]$ to x_{n_1,n_2} using x_{n_1,n_2}=x[n_1+N_1n_2];
2. Take N_2–point DFT of each row (fixed n_1).
3. Multiply this array point-by-point by the array of twiddle factors $e^{-j\frac{2\pi}{N}n_1k_2}$.
4. Take N_1–point DFT of each column (fixed k_2). The result of this is X_{k_1,k_2}.
5. Map X_{k_1,k_2}=$X_{k_2+N_2k_1}$.

Note that the mappings used for $x[n]$ and X_k are different. But they can be made identical by taking the transpose (exchanging each column for the correspondingly numbered row) of X_{k_1,k_2} before the final mapping. This is called *index shuffling*.

Example: 4-point DFT to 2×2 array.
Use the 2-D visualization to map the general 4-point DFT of $\{a, b, c, d\}$ to 4 2-point DFTs and twiddle mults. Recall that 2-point DFTs are just additions and subtractions.

Solution:

Map $\{a, b, c, d\}$ to a 2×2 array:

$$\{a,b,c,d\} \rightarrow \boxed{\begin{matrix} a & b \\ c & d \end{matrix}} = \boxed{\begin{matrix} a \\ c \end{matrix}} \boxed{\begin{matrix} b \\ d \end{matrix}}$$

Take 2-point DFTs of each column:

$$\boxed{\begin{matrix} (a+c) \\ (a-c) \end{matrix}} \boxed{\begin{matrix} (b+d) \\ (b-d) \end{matrix}} = \boxed{\begin{matrix} (a+c) & (b+d) \\ (a-c) & (b-d) \end{matrix}}$$

Multiply by twiddle factors:

$$\times \boxed{\begin{matrix} 1 & 1 \\ 1 & -j \end{matrix}} = \boxed{\begin{matrix} (a+c) & (b+d) \\ (a-c) & -j(b-d) \end{matrix}}$$

Take 2-point DFTs of each row:

$$\boxed{\begin{matrix} [(a+c)+(b+d)] & [(a+c)-(b+d)] \\ [(a-c)-j(b-d)] & [(a-c)+j(b-d)] \end{matrix}}$$

Read off $\{X_0, X_1, X_2, X_3\}$. Note *index shuffling*:

$$\boxed{\begin{matrix} X_0 & X_2 \\ X_1 & X_3 \end{matrix}} \rightarrow \{X_0, X_1, X_2, X_3\}.$$

7.4.4 Radix-2 Cooley-Tukey FFTs

Let N be even, so we can write $N{=}2\frac{N}{2}$. We can set $N_1{=}2$ and $N_2{=}\frac{N}{2}$, or vice-versa. This gives two different radix-2 Cooley-Tukey FFTs.

Decimation in Time

Let $N_1{=}2$ and $N_2{=}\frac{N}{2}$. This gives

$$\begin{aligned} X_{k_2} &= 1 \underbrace{\sum_{n_2=0}^{N/2-1} e^{-j2\pi\frac{n_2 k_2}{N/2}} x[2n_2]}_{\text{N/2−point DFT}} \qquad (7.27) \\ &+ e^{-j2\pi\frac{1k_2}{N}} \underbrace{\sum_{n_2=0}^{N/2-1} e^{-j2\pi\frac{n_2 k_2}{N/2}} x[2n_2+1]}_{\text{N/2−point DFT}}. \\ X_{k_2+N/2} &= 1 \underbrace{\sum_{n_2=0}^{N/2-1} e^{-j2\pi\frac{n_2 k_2}{N/2}} x[2n_2]}_{\text{N/2−point DFT}} \\ &- e^{-j2\pi\frac{1k_2}{N}} \underbrace{\sum_{n_2=0}^{N/2-1} e^{-j2\pi\frac{n_2 k_2}{N/2}} x[2n_2+1]}_{\text{N/2−point DFT}} \end{aligned}$$

We take separate $\frac{N}{2}$-point DFTs of $x[n]$ at even times and $x[n]$ at odd times, then add and subtract them.

Only half the terms are multiplied by twiddle factors, so the number of twiddle mults is $\frac{N}{2}\log_2 N$. This agrees with $(N_1{-}1)(N_2{-}1){=}(2{-}1)(\frac{N}{2}{-}1){\approx}\frac{N}{2}$.

Decimation in Frequency

Let $N_1{=}\frac{N}{2}$ and $N_2{=}2$. This gives

$$X_{2k_1} = \underbrace{\sum_{n_1=0}^{N/2-1} e^{-j2\pi\frac{n_1 k_1}{N/2}}}_{\text{N/2−point DFT}} \underbrace{[x[n_1]+x[n_1+N/2]]}_{\text{Half of 2−point DFT}} 1. \qquad (7.28)$$

$$X_{2k_1+1} = \underbrace{\sum_{n_1=0}^{N/2-1} e^{-j2\pi\frac{n_1 k_1}{N/2}}}_{\text{N/2−point DFT}} \underbrace{[x[n_1]-x[n_1+N/2]]}_{\text{Half of 2−point DFT}} e^{-j2\pi\frac{1n_1}{N}}.$$

We take separate $\frac{N}{2}$-point DFTs of the sums and differences $(x[n] \pm x[n{+}\frac{N}{2}])$.

Only one set of DFTs are multiplied by twiddle factors, so the number of twiddle mults is again $\frac{N}{2}\log_2 N$. The first FFT derived in this chapter (using z-transforms evaluated on the unit circle) is the decimation-in-frequency radix-2 Cooley-Tukey FFT.

7.5 Real-Time vs. Batch Signal Processing

Dereverberation is a good example of deconvolution of a minimum phase system using its inverse system. However, many systems of interest are not minimum phase, so there is no stable and causal inverse system.

So far in these notes, all signal processing has been performed in *real time*. This means that a signal is processed by passing it through an LTI system. In discrete time, the system is a difference equation, implemented on a computer or on a DSP chip.

For real-time signal processing, a stable and causal system is required. But there is another approach: Wait until the *entire* signal is known and stored, and then process it in its entirety. This is called *batch* signal processing. The advantage of batch signal processing is that the LTI system no longer needs to be

either stable or causal. This greatly expands the class of systems that can be used for signal processing. Batch signal processing is usually performed using the FFT algorithm to compute DFTs.

7.6 APPLICATION: Deconvolution using DFT

7.6.1 Deconvolution using DFT

Let $x[n]$ have duration M and $h[n]$ have duration L. Then $y[n]$ has duration N=L+M–1. Define the *zero-padded* versions $\tilde{h}[n]$ of $h[n]$ and $\tilde{x}[n]$ of $x[n]$ as

$$\begin{aligned} \tilde{h}[n] &= \{h[n], \underbrace{0 \ldots 0}_{N-L}\} \quad H_k = \mathrm{DFT}(\tilde{h}[n]) \\ \tilde{x}[n] &= \{x[n], \underbrace{0 \ldots 0}_{N-M}\} \quad X_k = \mathrm{DFT}(\tilde{x}[n]) \end{aligned} \tag{7.29}$$

and Y_k of $y[n]$. Then $Y_k = H_k X_k$. Dividing by H_k and taking an N-point inverse DFT gives

$$\tilde{x}[n] = \mathrm{DFT}^{-1}(\mathrm{DFT}(y[n])/\mathrm{DFT}(\tilde{h}[n])). \tag{7.30}$$

In summary, divide the DFT of the output $y[n]$ by the DFT of the zero-padded impulse response $h[n]$ and take the inverse DFT of the quotient. The result is the zero-padded input $\tilde{x}[n]$. Discarding the N–M final zeros gives $x[n]$.

Example: Illustrative deconvolution.
We are given the response to an input $x[n]$:

- $x[n] \rightarrow \boxed{h[n]=\{\underline{1}, 2, 3\}} \rightarrow \{\underline{6}, 19, 32, 21\}$.

The goal is to compute the input $x[n]$.

Solution:
The duration of $y[n]$ is $N = 4$.
The 4-point DFT of $y[n]$ is:

$$\begin{aligned} Y_0 &= 6(1) + 19(1) + 32(1) + 21(1) = 78 \\ Y_1 &= 6(1) - 19(j) - 32(1) + 21(j) = -26 + j2 \\ Y_2 &= 6(1) - 19(1) + 32(1) - 21(1) = -2 \\ Y_3 &= 6(1) + 19(j) - 32(1) - 21(j) = -26 - j2. \end{aligned} \tag{7.31}$$

The 4-point DFT of $h[n]$ is:

$$\begin{aligned} H_0 &= 1(1) + 2(1) + 3(1) + 0(1) = 6 \\ H_1 &= 1(1) - 2(j) - 3(1) + 0(j) = -2 - j2 \\ H_2 &= 1(1) - 2(1) + 3(1) - 0(1) = 2 \\ H_3 &= 1(1) + 2(j) - 3(1) - 0(j) = -2 + j2. \end{aligned} \tag{7.32}$$

So the 4-point DFT of $x[n]$ is

$$\begin{aligned} X_0 &= \frac{Y_0}{H_0} = \frac{78}{6} = 13. \\ X_1 &= \frac{Y_1}{H_1} = \frac{-26 + j2}{-2 - j2} = 6 - j7. \\ X_2 &= \frac{Y_2}{H_2} = \frac{-2}{2} = -1. \\ X_3 &= \frac{Y_3}{H_3} = \frac{-26 - j2}{-2 + j2} = 6 + j7. \end{aligned} \tag{7.33}$$

The inverse DFT of {13,6–j7,–1,6+j7} is $\{\underline{6}, 7, 0, 0\}$. Therefore, the input $x[n] = \{\underline{6}, 7\}$.

7.6.2 Deconvolution: Z-Transforms

This problem is small enough that we could have used polynomial division. Taking the z-transform of $y[n] = h[n] * x[n]$ gives $Y(z) = H(z)X(z)$, so that

$$X(z) = \frac{Y(z)}{H(z)} = \frac{6 + 19z^{-1} + 32z^{-2} + 21z^{-3}}{1 + 2z^{-1} + 3z^{-2}} = 6 + 7z^{-1} \tag{7.34}$$

However, polynomial divison does not work on larger problems, due to roundoff error.

Another way to perform deconvolution is to examine the zeros of $Y(z)$. Some of these zeros come from $H(z)$ and the rest come from $X(z)$. Since $H(z)$ is known, the zeros of $Y(z)$ coming from $H(z)$ can be identified. The remaining zeros all come from $X(z)$, and they determine $X(z)$ to a scale factor. However, this method does not work unless the problem is very small, again due to roundoff error. The locations of zeros of a polynomial are very sensitive to perturbations of the coefficients of the polynomial, and in practice the zeros of $H(z)$ do not match some of the zeros of $Y(z)$, so this approach will not work.

7.6.3 FFT Implementation Issues

N can be rounded up to the next power of two to make computation of the three DFTs using the FFT algorithm even faster. If any of the H_k values are zero, which would require dividing by zero, changing N often solves this problem. To see why this works, suppose $H_{k_o} = 0$ for some index k_o. This means that the z-transform $H(z)$ of $h[n]$ has a zero at $e^{j2\pi k_o/N}$, since the DFT is the z-transform sampled on the unit circle at $z = e^{j2\pi k/N}$ for integers k. Changing N to, say, $N+1$ means that the DFT is now the z-transform sampled at $z = e^{j2\pi k/(N+1)}$, so the zero at $e^{j2\pi k_o/N}$ will now be missed, and the $(N+1)$-point DFT of $h[n]$ will no longer be zero at index $k = k_o$.

A more serious issue is what to do if the DTFT $H(e^{j\omega}) \approx 0$ for a range of ω, so changing N will not help. This is a serious problem in image processing, so we defer discussion of it to that chapter.

7.7 APPLICATIONS: Filtering Signals using the DFT

7.7.1 Brick-Wall Low-Pass Filtering

We have seen that real-time noise filtering is relatively difficult computationally. Even in discrete time, the half-band filter designed in a previous chapter using five zeros and five poles did not have a very selective frequency response function.

We can do much better using batch signal processing. Implementing a brick-wall low-pass filter is trivial–just set the undesired frequency components to zero! Specifically, to eliminate all frequency components above `F` Hertz in a signal sampled at `S` samples per sec. and stored in Matlab vector `X`, use

```
N=length(X);FX=fft(X);K=ceil(N*F/S)+1;
FX(K:N+2-K)=0;Y=real(ifft(FX));
```

Note that one must be added to $\frac{NF}{S}$ since Matlab indexing starts at one, not zero. The values of `FX` set to zero must maintain the conjugate symmetry of `FX`. Even so, the slight roundoff error inherent in any computation requires that `Y` be computed as the *real part* of `ifft(FX)`, since the imaginary part, while small ($\approx 10^{-18}$), is not zero.

We do not demonstrate this with examples, since we can do much better job of eliminating noise than using brick-wall low-pass filtering, as we now show.

7.7.2 APPLICATION: Noise Filtering of Signals with Line Spectra

If the signal is periodic, so that it has a line spectrum, we can do *much* better than a brick-wall low-pass filter. Since we know that the signal power is entirely represented in the spectral lines, we can set *all other* frequency components to zero without affecting the signal! This virtually eliminates the noise; only the noise added to the spectral lines themselves remains.

We present two examples: (1) a small illustrative example that demonstrates the procedure; (2) filtering the noisy trumpet signal.

Example: Illustration of noise filtering a periodic signal.

The goal is to filter out noise added to the signal

$$x(t) = 1 + 4\sin(2\pi 5t) + 10\cos(2\pi 10t + 0.6435).$$

Note this is a periodic signal with period 0.1 sec.

The formula for $x(t)$ is not known to the user. All that is given to the user is a segment of the signal, 1 sec. long, sampled at 1000 samples per sec. The signal-to-noise ratio is –2.56 dB. This means that the noise power is greater than the signal power!

Solution:

The signal $x(t)$ is plotted for $0 \le t \le 1$, both without and with noise, in the next two figures.

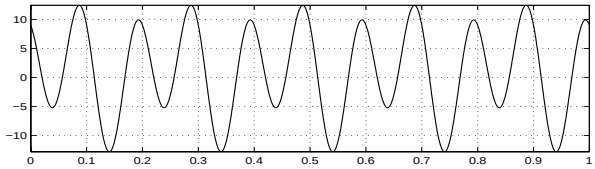


Figure 7.2: Noiseless periodic signal.

The *two*-sided line spectrum of the noisy signal is computed using the DFT, and plotted in the next figure. Negative frequencies are omitted in the plot,

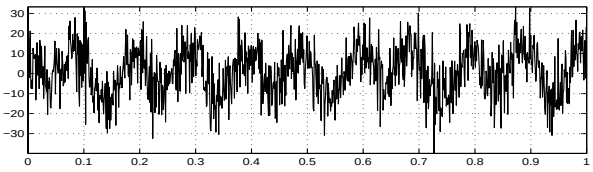


Figure 7.3: Noisy periodic signal.

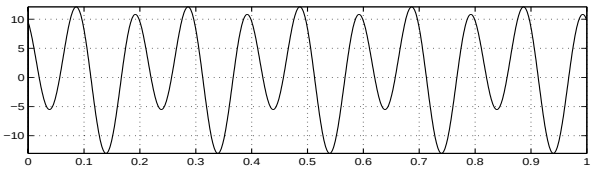


Figure 7.6: Inverse DFT of thresholded spectrum.

since the *two*-sided spectrum is known to be symmetric about dc (zero frequency). Since the components are computed only at discrete frequencies, specifically, integer numbers of Hertz (the segment is 1 sec. long), a stem plot is used to display it.

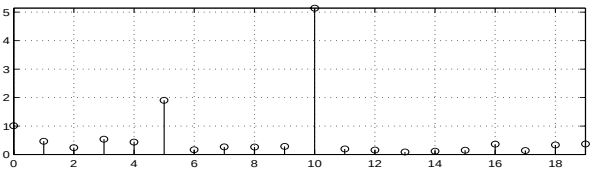


Figure 7.4: Spectrum of noisy signal.

The amplitudes of the signal spectral lines are much larger than the amplitudes of the noise. This suggests *thresholding* the spectrum: Set all frequency components with amplitudes less than 0.9 to 0. The thresholded spectrum is shown in the next figure.

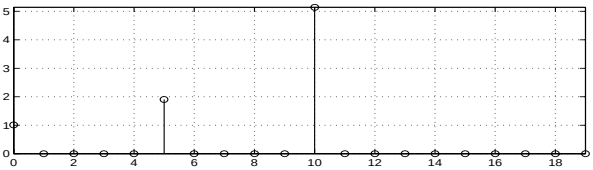


Figure 7.5: Thresholded spectrum of noisy signal.

The inverse DFT of the thresholded spectrum is shown in the next figure. Comparing the original, noisy and filtered signals shows that the noise has been virtually eliminated. The only remaining noise is the noise added to the signal spectral lines; the effect of this noise is not apparent in the figure.

The most remarkable aspect of this example is that we did not use any knowledge of the signal or of the noise strength. The only assumption was that the harmonics of the signal were larger than any components of the noise, so that a simple thresholding strategy would eliminate the noise while leaving the signal. The value 0.9 of the threshold was chosen by trial-and-error, without using any knowledge of the signal and noise strengths. An important clue is that since the signal is known to be periodic, the harmonics occur at integer multiples of some fundamental frequency; the threshold should be chosen to preserve this structure in the filtered signal. Statistics of the signal and noise, if known, can be used to compute a value of the threshold, but this is far beyond the scope of these notes.

The Matlab code for this example:

```
clear;rand('seed',0);N=10*randn(1,1000);
T=linspace(0,1,1000);X=1+4*sin(2*pi*5*T);
X=X+10*cos(2*pi*10*T+0.6435);Y=X+N;
SNR=10*log10(sum(X.^2)/sum(N.^2))
subplot(211),plot(T,X)
subplot(212),plot(T,Y),figure
FX=fft(X)/1000;FY=fft(Y)/1000;
FZ=FY;FZ(find(abs(FY)<0.9))=0;
subplot(211),stem(0:19,abs(FY(1:20)))
subplot(212),stem(0:19,abs(FZ(1:20)))
Z=1000*real(ifft(FZ));figure
subplot(211),plot(T,Z)
```

Example: Noise filtering trumpet signal.
Noise is added to the actual trumpet signal, which is sampled at 44100 samples per sec. The duration of the noisy trumpet signal is 32768 samples, or $\frac{32768}{44100}$=0.743 s. The signal-to-noise ratio is 8.54 dB. Threshold its spectrum to eliminate the noise.

Solution:
The procedure is identical to the previous example.

The threshold is set (by trial-and-error) at 0.0015.

Small segments of the trumpet signal, without and with noise added, are plotted in the next two figures.

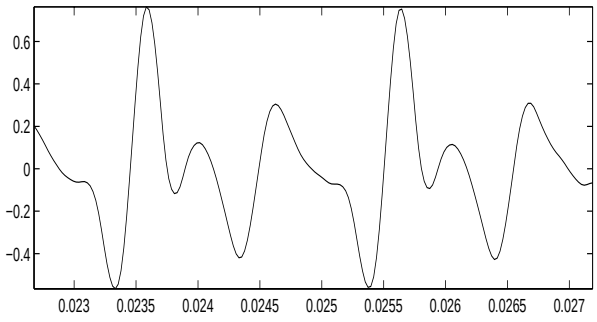


Figure 7.7: Noiseless trumpet signal.

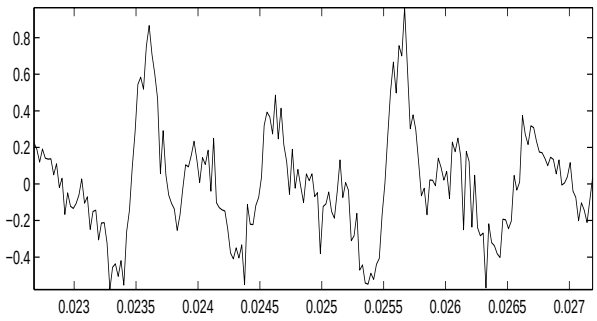


Figure 7.8: Noisy trumpet signal.

The two-sided line spectrum of the noisy trumpet signal, computed using the DFT, is plotted in the next figure. Due to the much finer discretization of frequency, a continuous plot is used to display it.

Setting all frequency components with amplitudes less than 0.0015 to zero gives the thresholded spectrum shown in the next figure.

The inverse DFT of the thresholded spectrum is shown in the next figure. Comparing the original, noisy and filtered signals shows that the noise has been virtually eliminated. The only remaining noise is the noise added to the trumpet signal spectral lines.

The following program should be run to confirm that the noise has been almost eliminated from the

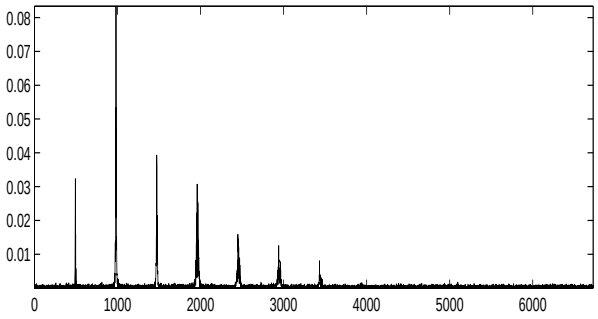


Figure 7.9: Spectrum of noisy trumpet signal.

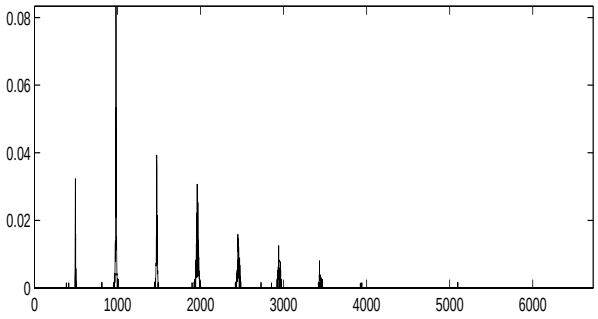


Figure 7.10: Thresholded spectrum of noisy trumpet.

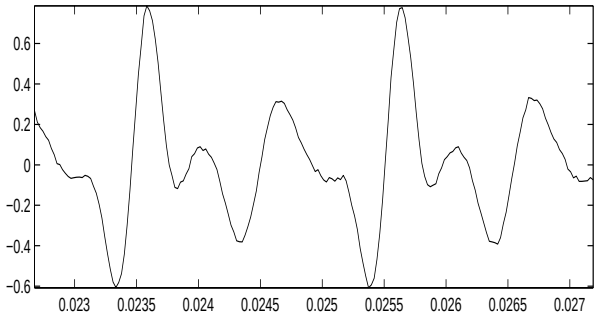


Figure 7.11: Inverse DFT of thresholded spectrum.

sound of the trumpet. Note that the additive white noise sounds like hissing.

The Matlab code for this example:

```
clear;rand('seed',0);N=.1*randn(1,32768);
```

```
NN=[1000:1199];T=NN/44100;
F=[0:4999]*44100/32768;
load trumpet.mat;Y=X+N;
figure,subplot(211),plot(T,X(NN))
figure,subplot(211),plot(T,Y(NN))
FY=fft(Y)/32768;
FZ=FY;FZ(find(abs(FY)<0.0015))=0;
Z=32768*real(ifft(FZ));
figure,subplot(211),plot(F,abs(FY(1:5000)))
figure,subplot(211),plot(F,abs(FZ(1:5000)))
figure,subplot(211),plot(T,Z(NN))
soundsc(Y,44100),pause,soundsc(Z,44100)
SNR=10*log10(sum(X.^2)/sum(N.^2))
```

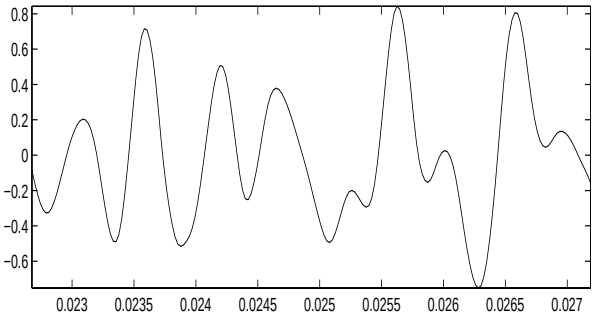


Figure 7.12: Time waveform of two-trumpets signal.

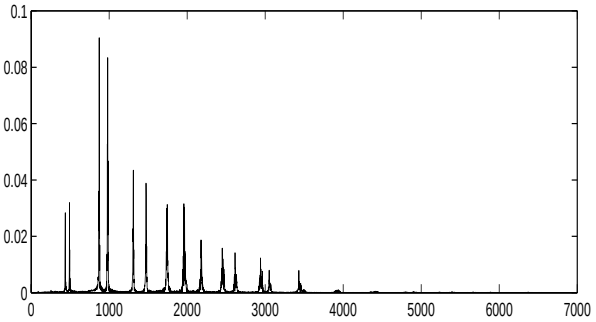


Figure 7.13: Spectrum of two-trumpets signal.

7.7.3 APPLICATION: Removal of Periodic Interference

Periodic *interference* can be removed easily by simply setting the harmonics of the interference to zero. No thresholding is necessary if the period of the interference is known. We show how effective this can be by revisiting the two-trumpets signal to which we applied a comb filter in a previous chapter.

Example: Two-trumpets signal, revisited.

We are given the signal of two actual trumpets playing simultaneously. One trumpet is playing note A (which has a period of $\frac{1}{440}$ sec.) and the other trumpet is playing note B (which has a period of $\frac{1}{494}$ sec.). The signal was sampled at the standard CD sampling rate of 44100 samples per sec. Use batch filtering and the DFT to eliminate the sound of the trumpet playing note A, while keeping the trumpet playing note B. We need only know that note B is at a higher frequency than note A.

Solution:

The time waveform of the two-trumpets signal is shown in the next figure. It is difficult to make sense.

The spectrum of the two-trumpets signal is shown in the next figure. Note the pairs of harmonics. The lower harmonic of each pair is associated with note A, and the higher harmonic of each pair is associated with note B.

All we need to do is set the harmonics of the *lower* fundamental frequency to zero. We set the lower frequencies to zero, since we wish to eliminate note A and note A is at a lower frequency than note B. We don't even need to interpret the output of the DFT!

The resulting spectrum, with harmonics of one trumpet eliminated, is shown in the next figure.

The time waveform of the filtered two-trumpets signal is shown in the next figure. This signal is very similar to, but not identical to, the noiseless trumpet signal. Hence we have eliminated one trumpet while leaving the other almost unaffected. The filter does not work perfectly since some of the higher-frequency harmonics of the two trumpets overlap, so that one cannot be eliminated without affecting the other.

The reader should run the following program and confirm that that one of the two trumpets has almost been eliminated.

The Matlab code for this example:

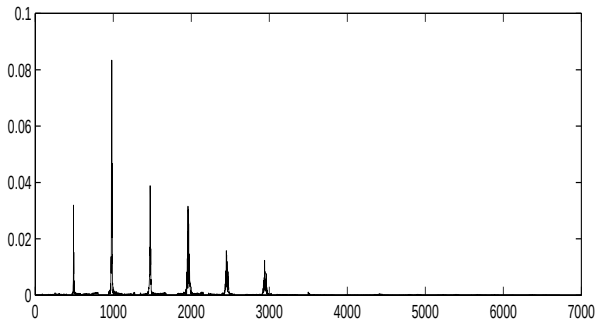


Figure 7.14: Spectrum of filtered two-trumpets.

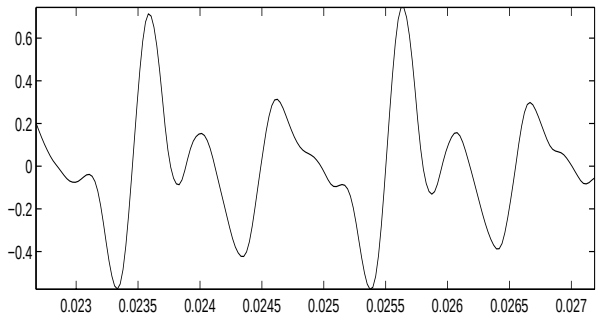


Figure 7.15: Time waveform of filtered two-trumpets.

```
load twotrumpets.mat;S=44100;N=length(X);
F=[0:4999]*S/N;FX=fft(X);FX=FX/S;
subplot(211),plot(F,abs(FX(1:5000)))
Z=[301:350 601:700 951:1000];
Z=[Z 1251:1350 1601:1651 1901:2000];
Z=[Z 2250:2300 2501:2600 2851:2950];
FX([Z N+2-Z])=0;NN=[1000:1199];T=NN/S;
subplot(212),plot(F,abs(FX(1:5000)))
Y=real(ifft(FX));figure
subplot(211),plot(T,X(NN))
subplot(212),plot(T,Y(NN));
soundsc(X,S),pause,soundsc(Y,S)
```

7.8 APPLICATION: Spectra of Non-Periodic Signals using the DFT

In the previous chapter, we computed the Fourier series expansion of a periodic and bandlimited signal by sampling it in time and using the DTFS. Now we compute the Fourier transform of a finite-duration and bandlimited signal by sampling it in time and in frequency and using the DFT. In addition to its obvious use in numerically computing continuous Fourier transforms, this section shows the power of the sampling theorem.

7.8.1 Problem Statement

The goal is to compute the continuous-time Fourier transform (CTFT) and its inverse

$$\begin{aligned} X(j2\pi f) &= \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \\ x(t) &= \int_{-\infty}^{\infty} X(j2\pi f)e^{+j2\pi ft}df \end{aligned} \quad (7.35)$$

where $\Omega{=}2\pi f$ and $X(j2\pi f){=}X(j\Omega)$. Using f in Hertz instead of Ω in radians per sec. saves some factors of 2π, and makes the results easier to follow.

For convenience we refer to $x(t)$ as the signal and $X(j2\pi f)$ as the spectrum. We assume that:

- $X(j2\pi f)$ is bandlimited to $F/2$ Hertz: $X(j2\pi f) = 0$ unless $|f| < F/2$ (bandwidth F).
- $x(t)$ has *duration* T: $x(t) = 0$ unless $|t| < T/2$: $x(t)$ is *time*-limited to $T/2$ sec. (timewidth T).
- T and F are rounded up so that their dimensionless product $N{=}TF$ is an odd integer. Then we define $M{=}(N{-}1)/2{=}(TF{-}1)/2$.

If $x(t)$ has a range of non-zero values $a < t < b$ instead of $|t| < T/2$, define $t_c = (b + a)/2$ and $T = b - a$. Then $y(t) = x(t + t_c) = 0$ unless $|t| < T/2$. So we can compute the Fourier transform $Y(j2\pi f)$ of $y(t)$ using the procedure to follow. Then, using the time delay property of Fourier transforms, we have

$$X(j2\pi f) = Y(j2\pi f)e^{-j2\pi ft_c}. \quad (7.36)$$

In theory, no finite-duration signal can be perfectly bandlimited. In practice, most signals of interest are effectively bandlimited, so there exists a frequency F Hertz above which $X(j2\pi f)$ is negligible. We will see several examples of this throughout this chapter. Discrete-time signal processing would be of only theoretical interest were this not true.

7.8.2 Sampling in Time and Frequency

The plan is to recall that sampling $x(t)$ at $t = n/F$ makes its spectrum periodic with period F. Since $x(t)$ is band-limited to $F/2$, there is no aliasing.

But we also exchange the time and frequency domains, so sampling $X(j2\pi f)$ at $f{=}k/T$ makes its inverse Fourier transform periodic with period T. Since $x(t)$ is time-limited to $T/2$, there is no aliasing.

As a result of these facts, samples of $X(j2\pi f)$ can be computed from samples of $x(t)$, and vice-versa.

Since $x(t)$ is bandlimited to $F/2$ Hertz, we may sample signal $x(t)$ at a sampling rate of F samples per sec. without the sampled spectra overlapping. The result is the continuous-time signal $\tilde{x}(t)$:

- $\tilde{x}(t){=}\sum_{n=-\infty}^{\infty} x(t)\delta(t{-}\frac{n}{F}){=}\sum_{n=-\infty}^{\infty} x[n]\delta(t{-}\frac{n}{F})$ where the discrete-time signal $x[n]{=}x(n/F)$.
- $F[\tilde{x}(t)] = F\sum_{k=-\infty}^{\infty} X(j2\pi(f{-}kF))$. The spectrum of $\tilde{x}(t)$ is the periodic (with period F) repetition of the spectrum $X(j2\pi f)$ of $x(t)$.

But we can also write $F[\tilde{x}(t)]$ as

$$\begin{aligned} F[\tilde{x}(t)] &= \int_{-\infty}^{\infty} \tilde{x}(t)e^{-j2\pi ft}dt \qquad (7.37)\\ &= \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[n]\delta(t-\frac{n}{F})e^{-j2\pi ft}dt. \end{aligned}$$

Exchanging the order of the integral and summation,

$$\begin{aligned} F[\tilde{x}(t)] &= \sum_{n=-\infty}^{\infty} x[n]\int_{-\infty}^{\infty} \delta(t-\frac{n}{F})e^{-j2\pi ft}dt \qquad (7.38)\\ &= \sum_{n=-\infty}^{\infty} x[n]e^{-j2\pi nf/F} = \sum_{n=-M}^{M} x[n]e^{-j2\pi nf/F}. \end{aligned}$$

$x(t)$ is time-limited to $|t| < T/2$, so $x[n]$ is time-limited to $|n| < TF/2$, so $x[n]{=}0$ unless $|n| \le M$.

Equating these two expressions for $F[\tilde{x}(t)]$ at the samples $f = k/T$ for $|k| \le M$ gives

$$X_k = X(j2\pi k/T) = \frac{1}{F}\sum_{n=-M}^{M} x[n]e^{-j2\pi nk/(TF)}, |k| \le M. \qquad (7.39)$$

This is the DFT of $x[n]$ with $N{=}TF$, except for the scale factor $1/F$ and ranges of indices n and k.

7.8.3 Exchange the Time and Frequency Domains

Now exchange, using Fourier duality:

- Signal $x(t)$ and spectrum $X(j2\pi f)$;
- Time t in sec. and frequency f in Hertz;
- Signal duration T and bandwidth F.

The following derivation should be compared to the derivation in the previous subsection.

Since $X(j2\pi f)$ is time-limited to a maximum time of $T/2$ seconds, we may sample spectrum $X(j2\pi f)$ in frequency at a sampling rate of T samples per Hertz without the sampled signals overlapping. The result is the continuous-frequency spectrum $\tilde{X}(j2\pi f)$:

- $\sum_{k=-\infty}^{\infty} X(j2\pi f)\delta(f{-}\frac{k}{T}){=}\sum_{k=-\infty}^{\infty} X_k\delta(f{-}\frac{k}{T})$ Discrete-frequency spectrum $X_k{=}X(j2\pi\frac{k}{T})$.
- $F^{-1}[\tilde{X}(j2\pi f)] = T\sum_{n=-\infty}^{\infty} x(t{-}nT)$.

This is the periodic (with period T) repetition of $x(t)$.

But we can also write $F^{-1}[\tilde{X}(j2\pi f)]$ as

$$\begin{aligned} F^{-1}[\tilde{X}(j2\pi f)] &= \int_{-\infty}^{\infty} \tilde{X}(j2\pi f)e^{j2\pi ft}df \qquad (7.40)\\ &= \int_{-\infty}^{\infty} \sum_{k=-\infty}^{\infty} X_k\delta(f-\frac{k}{T})e^{j2\pi ft}df. \end{aligned}$$

Exchanging order of integration and summation,

$$\begin{aligned} F^{-1}[\tilde{X}(j2\pi f)] &= \sum_{k=-\infty}^{\infty} X_k\int_{-\infty}^{\infty} \delta(f-\frac{k}{T})e^{j2\pi ft}df \qquad (7.41)\\ &= \sum_{k=-\infty}^{\infty} X_k e^{j2\pi kt/T} = \sum_{k=-M}^{M} X_k e^{j2\pi kt/T}. \end{aligned}$$

$X(j2\pi f)$ is band-limited to $|f| < F/2$, so X_k is band-limited to $|k| < FT/2$, so X_k=0 unless $|k| \leq M$.

Equating these two expressions for $F^{-1}[\tilde{X}(j2\pi f)]$ at the samples t=n/F for $|n| \leq M$ gives

$$x[n] = x\left(\frac{n}{F}\right) = \frac{1}{T}\sum_{k=-M}^{M} X_k e^{j2\pi nk/(TF)}, |n| \leq M. \tag{7.42}$$

This is the inverse DFT of $x[n]$ with N=TF, except for scale factor $1/T$ and the range of indices n and k.

7.8.4 Ranges of Indices

The above two formulae are the N-point DFT and inverse DFT, except for scale factors $\frac{1}{F}$ and $\frac{1}{T}$, and the ranges of indices:

- $|n| \leq M$ instead of $0 \leq n \leq (N-1)$.
- $|k| \leq M$ instead of $0 \leq k \leq (N-1)$.

But note that $e^{\pm j2\pi nk/N}$ is periodic in both n and k with period N, and the sampled signals and spectra are periodic in the other domain with period N. So any range of indices of length N can be used here.

7.8.5 Summary

- $X(j2\pi f)$ can be computed at frequencies f=k/T from samples $x[n]$ of $x(t)$ at t=n/F.
- $x(t)$ can be computed at times t=n/F from samples X_k of $X(j2\pi f)$ at f=k/T.
- Use an N=TF-point DFT and scale factors:
- X_k=$\frac{1}{F}\sum_{n=-M}^{M} x[n]e^{-j2\pi nk/(TF)}$, M=$\frac{N-1}{2}$.
- $x[n]$=$\frac{1}{T}\sum_{k=-M}^{M} X_k e^{+j2\pi kn/(TF)}$, M=$\frac{N-1}{2}$.
- Note the combined scale factor $\frac{1}{T}\frac{1}{F}$=$\frac{1}{N}$.

7.8.6 Comparison with Discretization

This is the same result that would be obtained by discretizing the CTFT as follows:

$$X(j2\pi k\Delta f) = \sum_{n=-M}^{M} x(n\Delta t)e^{-j2\pi(k\Delta f)(n\Delta t)}\Delta t \tag{7.43}$$

$$x(t) = \sum_{k=-M}^{M} X(j2\pi k\Delta f)e^{+j2\pi(k\Delta f)(n\Delta t)}\Delta f.$$

- t=$n\Delta t$ where Δt=$1/F$.
- f=$k\Delta f$ where Δf=$1/T$.
- $\Delta t\Delta f$=$1/N$ and N=TF.

However, the DFT computation is *exact* if the signal has duration T seconds and bandwidth B Hertz!

Example: Using DFT to compute CTFT.
Use the DFT to compute $F[e^{-|t|}]$=$\frac{2}{\Omega^2+1}$=$\frac{2}{4\pi^2 f^2+1}$.
Assume for computational purposes that

- $e^{-|t|} \approx 0$ for $|t| > 6$ since e^{-6}=0.0025.
- $\frac{2}{4\pi^2 f^2+1} \approx 0$ for $|f| > 8$: $\frac{2}{4\pi^2 8^2+1}$=0.0008.

Solution:
Using the above assumptions, we have

- T=2(6)=12→ Δf=1/12.
- F=2(8)=16→ Δt=1/16.
- N=TF=(12)(16)=192.

Sampling in both time and frequency and using the DFT gives the next plot.

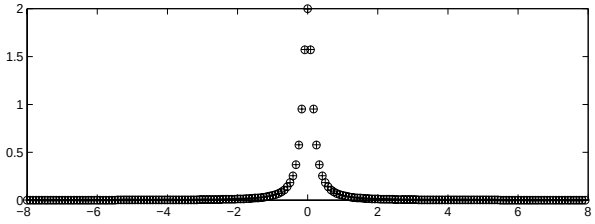


Figure 7.16: Actual values 'o' and DFT computation '+' of $F[e^{-|t|}]$ using $N = 192$.

The Matlab code for this example:

```
T=[-6:1/16:6-1/16];X=exp(-abs(T));
F=[-8:1/12:8-1/12];FX=2./(4*pi*pi*F.*F+1);
FY=fftshift(abs(fft(X)))/16;
subplot(211),plot(F,FX,'o',F,FY,'+r')
```

Although Δt=$\frac{1}{16}$ and Δf=$\frac{1}{12}$ are very coarse discretizations, the DFT has done an excellent job of computing $F[e^{-|t|}]$.

Example: Using DFT to compute CTFT.

Use the DFT for $F[\frac{e^{-t^2/2}}{\sqrt{2\pi}}]=e^{-\Omega^2/2}=e^{-2\pi^2 f^2}$.

Assume for computational purposes that

- $\frac{e^{-t^2/2}}{\sqrt{2\pi}} \approx 0$ for $|t|>4$ since $\frac{e^{-4^2/2}}{\sqrt{2\pi}}$=0.00013.
- $e^{-2\pi^2 f^2} \approx 0$ for $|f|>0.6$: $e^{-2\pi^2 0.6^2}$=0.00082.

Solution:

Using the above assumptions, we have

- T=2(4)=8→ Δf=1/8=0.125.
- F=2(0.6)=1.2→ Δt=1/1.2.
- N=TF=(8)(1.2)=9.6.
- To round N up to 10, we must round F up to 1.25 and Δt down to 1/1.25=0.8.

Sampling in both time and frequency and using the DFT gives the next plot.

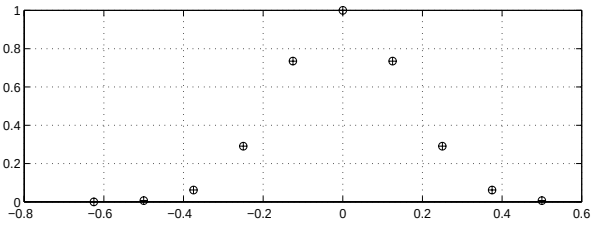


Figure 7.17: Actual values 'o' and DFT computation '+' of $F[\frac{1}{\sqrt{2\pi}}e^{-t^2/2}]$ using N=10.

The Matlab code for this example:

```
T=[-4:0.8:3.2];X=exp(-T.*T/2)/sqrt(2*pi);
F=[-0.625:0.125:0.5];
FX=exp(-2*pi*pi*F.*F);
FY=fftshift(abs(fft(X)))/1.25;
subplot(211),plot(F,FX,'o',F,FY,'+r')
```

Using just **10** discretization points has done an excellent job of computing $F[\frac{1}{\sqrt{2\pi}}e^{-t^2/2}]$! This is certainly not obvious from simply discretizing the continuous-time Fourier transform. This example shows the power of the sampling theorem.

Example: Using DFT to compute spectra.

We use the DFT to compute the spectrum of the train whistle signal built into Matlab. It was plotted in Chapter 1, so we do not repeat that plot here.

The following code plots its one-sided spectrum:

```
load train.mat;Ly2=length(y)/2;
f=[0:Ly2-1]*8192/(2*Ly2);
Fy=fft(y);Fy2=Fy(1:Ly2);
subplot(211),plot(f,abs(Fy2))
```

The one-sided spectrum of the train whistle:

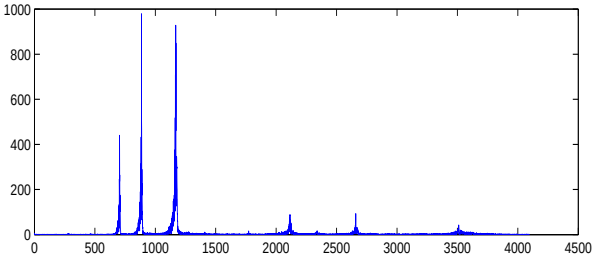


Figure 7.18: One-sided spectrum of train whistle.

- There are only 6 significant nonzero components, the peaks centered at the following frequencies: 705, 886, 1170, 2115, 2658, 3511 Hertz.
- The 1^{st} 3 frequencies are in 5:3 and 5:4 ratios. This is suggestive of chords in the whistle sound.
- The $4^{th}-6^{th}$ frequencies are triple the $1^{st}-3^{rd}$ frequencies. Are these harmonics of $1^{st}-3^{rd}$?
- Recall that the DFT scales the DTFS by N: X_k=Nx_k. The amplitudes of the sinusoids are `abs(Fy2)/Ly2`. The phases are `angle(Fy2)`. Remember this for the DFT of sinusoids!
- The train whistle signal is approximately $\frac{1}{Ly2}\times$:

$$\begin{aligned} 440\cos(2\pi 705t-1.70) &+ 979\cos(2\pi 886t+2.93)+ \\ 928\cos(2\pi 1170t+1.35) &+ 88\cos(2\pi 2115t+1.41)+ \\ 93\cos(2\pi 2658t+0.84) &+ 43\cos(2\pi 3511t+3.03) \end{aligned}$$

The "line spectrum" consists of peaks, not lines. This *spectral leakage* is covered in the next chapter. Note the train whistle signal is not exactly periodic; its amplitude dips at 0.5 sec., then increases again.

Chapter 8

Data Windows and Spectrograms

The previous chapter included several computed spectra of an actual trumpet signal, for which the spectral lines had broad bases. This is due in part to the trumpet signal not being perfectly periodic, but it is also due in part to *spectral leakage*, so called because the spectral lines seem to be "leaking."

Spectral leakage occurs because the spectrum is being computed from only a finite number of samples of the signal. While it cannot be eliminated (except in special cases), its effects can be mitigated using *data windows*. Multiplying the finite number of samples by a data window reduces the *sidelobes*, artifacts present in the computed spectrum, making it easier to interpret, and to spot small spectral peaks.

Although it is usually treated as a different subject, *spectrograms* are computed by using a series of data windows, with varying delays, and plotting the computed spectra alongside each other. A spectrogram provides a *time-varying* spectrum which tracks how various spectral components change over time.

8.1 Spectral Leakage

8.1.1 Examples of Spectral Leakage

The following example shows spectral leakage is not due to the complexity of the signal spectrum.

Example: Spectral Leakage.

Use the DFT to compute the spectra of the following two signals, each of which is a one-second-long segment of a sinusoid.

- $x(t) = 2\cos(2\pi 440t), 0 \leq t < 1$
- $y(t) = 2\cos(2\pi 440.5t), 0 \leq t < 1$

Both sinusoids are sampled at 1024 samples per sec., so each sampled sinusoid is 1024 samples long.

Solution:

The computed spectrum of $x(t)$ shows two spectral lines, both of height one, at ± 440 Hertz, as expected for a pure sinusoid. The two lines are not spread out at their bases. There is no spectral leakage.

The computed spectrum of $y(t)$ shows two spectral lines centered at ± 440.5 Hertz, but the two lines are spread out at their bases. Despite the simplicity of the signal spectrum, there is spectral leakage.

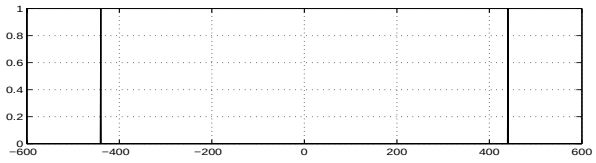


Figure 8.1: Spectrum of 440 Hertz sinusoid.

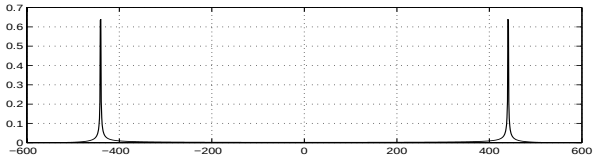


Figure 8.2: Spectrum of 440.5 Hertz sinusoid.

The Matlab code for this example:

```
N=0:1023;F=-512:511;
X=2*cos(2*pi*440*N/1024);
Y=2*cos(2*pi*440.5*N/1024);
FX=fft(X)/1024;FY=fft(Y)/1024;
plot(F,fftshift(abs(FX)))
plot(F,fftshift(abs(FY)))
```

The next subsection analyzes, using the concept of *periodic extension*, why the computed spectrum of the 440 Hertz sinusoid does not display spectral leakage, while the computed spectrum of the 440.5 Hertz sinusoid does display spectral leakage.

8.1.2 Periodic Extension of a Signal Segment

The key to understanding spectral leakage is to recall that: (1) the DFT actually computes the DTFS of the sampled signal; and (2) the DTFS comes from sampling a continuous-time Fourier series. So use of the DFT implicitly assumes the sampled signal is a single period of a periodic signal. The problem is the sampled signal is usually *not* a single period of a periodic signal. Instead, it consists of several periods and a fraction of a period of the signal.

Consider signal $x(t)=\{2\cos(2\pi f_o t), 0 \leq t < T\}$. Note that $x(t)$ is *not* a sinusoid; it is a *segment*, T sec. long, of the sinusoid $\{2\cos(2\pi f_o t), -\infty < t < \infty\}$. The segment $x(t)$ may be several periods long. The distinction between a sinusoid and a segment of the sinusoid is significant to understanding leakage.

Now define the *periodic extension* $\tilde{x}(t)$ of $x(t)$ by repeating $x(t)$, which has duration T, over and over in time t. Equivalently, adjoin to $x(t)$ copies of $x(t)$ delayed and advanced by integer multiples of T. So

$$\tilde{x}(t) = \{\ldots x(t+T), x(t), x(t-T), x(t-2T)\ldots\} \quad (8.1)$$

This is illustrated in the next two figures for two different cases analyzed in the next two paragraphs.

In the first figure, the length of the segment is an integral number of periods. Then the periodic extension of the segment is the true sinusoid, extending from $-\infty$ to ∞, since the endpoints of the segment match up. The DFT computes the spectrum of the sinusoid, extending from $-\infty$ to ∞. This is a line spectrum; there is no spectral leakage.

The sinusoid $2\cos(2\pi f_o t)$ has period $\frac{1}{f_o}$. If $T=\frac{M}{f_o}$ for some integer M, or equivalently if Tf_o is an integer, then $x(t)$ consists of exactly M periods of the sinusoid. In this case, the periodic extension $\tilde{x}(t)$ is precisely the sinusoid $\{2\cos(2\pi f_o t), -\infty < t < \infty\}$, since the endpoints of each segment match up.

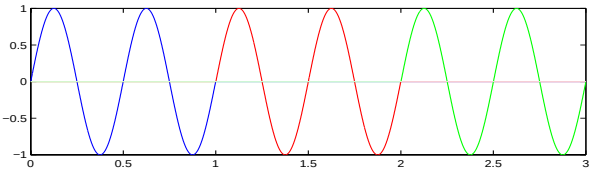


Figure 8.3: Periodic extension of a segment of a sinusoid. Each copy of the segment is a different color. Segment length is an integral number of periods.

In the second figure, the length of the segment is *not* an integral number of periods. Then the periodic extension of the segment is *not* the true sinusoid, since the endpoints of the segment do not match up. The DFT computes the spectrum of the repeated *segment* of the sinusoid. This is not a pure line spectrum; there is spectral leakage from the lines.

If Tf_o is not an integer, then $x(t)$ does not consist of an integral number of periods of the sinusoid, and the periodic extension $\tilde{x}(t)$ is not the same as the sinusoid, since the endpoints of each segment do not match up. There will be spectral leakage.

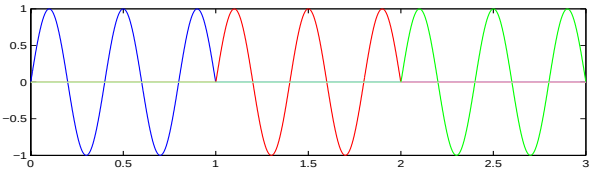


Figure 8.4: Periodic extensions of a segment of a sinusoid. Each copy of the segment is a different color. Segment length is a non-integral number of periods.

The Matlab code for these figures:

```
clear;T=linspace(0,3,3000);
Z=zeros(1,1000);T1=linspace(0,1,1000);
X=sin(4*pi*T1);Y=sin(5*pi*T1);
subplot(211),plot(T,[X Z Z]),hold on
```

```
subplot(211),plot(T,[Z X Z],'r'),hold on
subplot(211),plot(T,[Z Z X],'g'),hold on
subplot(211),plot(T,[Z Z Z],'w'),figure
subplot(211),plot(T,[Y Z Z]),hold on
subplot(211),plot(T,[Z Y Z],'r'),hold on
subplot(211),plot(T,[Z Z Y],'g'),hold on
subplot(211),plot(T,[Z Z Z],'w')
```

This explains why the computed spectrum is not a pure line spectrum unless the segment of the sinusoid contains an integral number of periods of the sinusoid. It does not explain why the spectrum of the segment consists of lines with bases. We analyze this in the next subsection.

8.1.3 Discrete Sinc Function

In fact, the broad base of each line is a *discrete sinc function* that replaces each line in the line spectrum. The discrete sinc function was defined previously as the DTFT of an even rectangular pulse. Here, let

$$w[n] = \begin{cases} 1, & n = 0, 1 \ldots N-1 \\ 0, & \text{otherwise} \end{cases} \tag{8.2}$$

The DTFT $W(e^{j\omega})$ of the rectangular pulse $w[n]$ is

$$W(e^{j\omega}) = \sum_{n=-\infty}^{\infty} w[n]e^{-j\omega n} = \sum_{n=0}^{N-1} 1e^{-j\omega n}. \tag{8.3}$$

Setting $r{=}e^{-j\omega}$ in the sum of a finite geometric series

$$\sum_{n=0}^{N-1} r^n = \frac{1-r^N}{1-r} \tag{8.4}$$

shows that the summation simplifies to

$$\begin{aligned} W(e^{j\omega}) &= \frac{1-e^{-j\omega N}}{1-e^{-j\omega}} \\ &= \frac{e^{j\omega N/2} - e^{-j\omega N/2}}{e^{j\omega/2} - e^{-j\omega/2}} \frac{e^{-j\omega N/2}}{e^{-j\omega/2}} \\ &= \frac{\sin(\omega N/2)}{\sin(\omega/2)} e^{-j\omega(N-1)/2}. \end{aligned} \tag{8.5}$$

The additional factor of $e^{-j\omega(N-1)/2}$ is due to the pulse being delayed by (*N*–1)/2 to make it causal. Recall that delaying a signal by D multiplies its DTFT by $e^{-j\omega D}$.

Omitting the factor $e^{-j\omega(N-1)/2}$, $W(e^{j\omega})$ is plotted in the next figure for $N{=}21$. $W(e^{j\omega})$ does look like a continuous-time sinc function (multiplied by N) for small ω. However, $W(e^{j\omega})$ is periodic with period 2π, unlike the continuous-time sinc function.

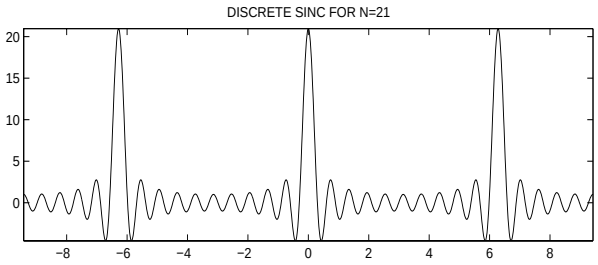


Figure 8.5: Discrete sinc function with $N{=}21$ plotted vs. ω. The phase factor is omitted.

The Matlab code used for this plot:

```
W=linspace(-pi,pi,1000);
X=sin(21*W/2)./sin(W/2);
subplot(211),plot(W,X),hold on
W2=[-20*pi:2*pi:20*pi]/21;
ZZ=[zeros(1,10) 21 zeros(1,10)];
subplot(211),stem(W2,ZZ,'r')
```

8.1.4 Spectrum of Sinusoidal Segment: Leakage

Now consider the segment $x[n]$ of the discrete-time sinusoid $\{2\cos(\omega_o n), -\infty < n < \infty\}$, defined as

$$\begin{aligned} x[n] &= \begin{cases} 2\cos(\omega_o n), & n = 0, 1 \ldots N-1 \\ 0, & \text{otherwise} \end{cases} \\ &= 2\cos(\omega_o n)w[n] = e^{j\omega_o n}w[n] + e^{-j\omega_o n}w[n]. \end{aligned} \tag{8.6}$$

Using the modulation property of the DTFT, the DTFT of $x[n]$ is found to be

$$X(e^{j\omega}) = W(e^{j(\omega-\omega_o)}) + W(e^{j(\omega+\omega_o)}). \tag{8.7}$$

So the spectrum of the sinusoidal segment $x[n]$ consists of two discrete sinc functions centered at $\pm\omega_o$.

Spectral leakage is actually due to each line in a line spectrum being replaced with a discrete sinc function. The broadening of the bases is actually the magnitude of a discrete sinc function.

8.1.5 Spectrum of Sinusoidal Segment: No Leakage

Since the spectrum of $x[n]$ computed using the DFT will consist of two discrete sinc functions, it might seem as though there will always be spectral leakage. Yet we know that if the duration N of $x[n]$ is an integral number of periods of the sinusoid, there will be no leakage. We now show why the discrete sinc function does not appear in the computed spectrum of an integral number of periods.

Recall the sinusoid $\{2\cos(\omega_o n), -\infty < n < \infty\}$ is periodic with period N if and only if $\omega_o{=}2\pi\frac{M}{N}$ for some integer M. Also recall the DFT computes $X(e^{j\omega})$ at frequencies $\omega{=}2\pi\frac{k}{N}$. Substituting both of these in (8.7) gives

$$X(e^{j2\pi k/N}) = W(e^{j2\pi(\frac{k}{N}-\frac{M}{N})}) + W(e^{j2\pi(\frac{k}{N}+\frac{M}{N})}). \tag{8.8}$$

We analyze each term separately. From the definition (8.5) of the discrete sinc function, the first term is

$$\begin{aligned} & W(e^{j2\pi(\frac{k}{N}-\frac{M}{N})}) \qquad (8.9)\\ &= \frac{\sin(2\pi(\frac{k}{N}-\frac{M}{N})\frac{N}{2})}{\sin(2\pi(\frac{k}{N}-\frac{M}{N})\frac{1}{2})]} e^{-j2\pi(\frac{k}{N}-\frac{M}{N})(N-1)/2} \\ &= \frac{\sin(\pi(k-M))}{\sin(\pi(k-M)/N)} e^{-j\pi(k-M)\frac{N-1}{N}} = N\delta[k-M] \end{aligned}$$

since $\sin(\pi(k{-}M)){=}0$ for integers k and M unless $k{=}M$. If $k{=}M$, this is indeterminate, but we have

$$W(e^{j2\pi(\frac{k}{N}-\frac{M}{N})}) = W(e^{j0}) = \sum_{n=0}^{N-1} 1e^{-j0n} = N. \tag{8.10}$$

This explains why the impulse is multiplied by N.

A visual picture of (8.9) is given in the next figure. The discrete sinc function with $N{=}21$ is sampled at frequencies $\omega{=}2\pi\frac{k}{N}$. Except at $k{=}0$, sampling is at the zero crossings of the discrete sinc function!

A similar analysis applied to the second term gives $N\delta[k{+}M]$. But since $k = 0, 1 \ldots N{-}1$ this should be changed to $N\delta[k{+}M{-}N]$, since $W(e^{j2\pi(\frac{k}{N}-\frac{M}{N})})$ is periodic in k with period N.

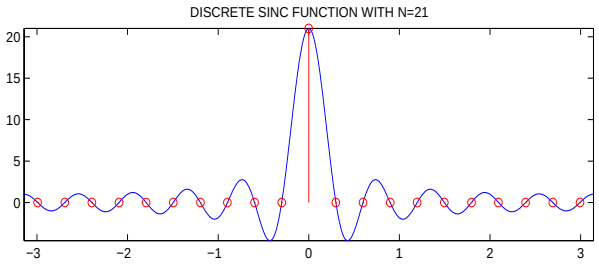


Figure 8.6: Discrete sinc function for $N{=}21$ sampled at $\omega{=}2\pi\frac{k}{N}$. The samples are at the zero crossings.

So if $\omega_o{=}2\pi\frac{M}{N}$, the N-point DFT computes

$$X(e^{j2\pi k/N}) = N\delta[k-M] + N\delta[k+M-N], \tag{8.11}$$

two lines at $k{=}M$ and $k{=}N{-}M$. There is no spectral leakage. This is illustrated in the figure above.

Example: No spectral leakage.
The spectrum of $\{\cos(0.3\pi n), n = 0 \ldots N-1\}$ is to be computed using the DFT. For what values of N will there be no spectral leakage?

Solution:
$\frac{\omega_o}{2\pi}{=}\frac{0.3\pi}{2\pi}{=}\frac{3}{20}{=}\frac{M}{N}$. The period=20 must divide N.

8.2 Data Windows

In practice, we will not be so fortunate as to have for our data segment an integer number of periods. Indeed, almost all discrete-time sinusoids are not periodic at all. What can we do in this situation?

The answer actually was given in (8.7), in which the spectrum of the data segment was found to be two discrete sinc functions at the two frequencies $\pm\omega_o$. The discrete sinc function came from multiplying the sinusoid by $w[n]$, which zeroed out its unobserved parts and left its observed parts unaltered. But what if we alter the observed parts, and use

$$w[n] = \begin{cases} \neq 1, & n = 0, 1 \ldots N-1 \\ 0, & \text{otherwise} \end{cases} \tag{8.12}$$

The DTFT of $x[n]$=$2\cos(\omega_o n)w[n]$ is then

$$X(e^{j\omega}) = W(e^{j(\omega-\omega_o)}) + W(e^{j(\omega+\omega_o)}). \quad (8.13)$$

But now we can choose the nonzero values of $w[n]$ to make $X(e^{j\omega})$ a less messy function. This is the idea behind using a *data window* $w[n]$.

8.2.1 Need for Data Windows

It may seem as though some base spreading of spectral lines is not a big deal. But the above examples used long segments (1024 samples) of sinusoids. Shorter segments cause much greater problems, as the following examples show.

Example: Sidelobes
Use a 256-point DFT to compute the spectrum of $\{\cos(0.2\pi n), 0 \le n \le 24\}$.

Solution:

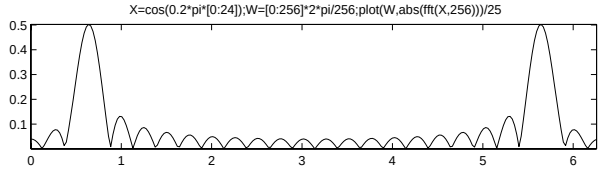


Figure 8.7: Computed spectrum of data segment $\{\cos(0.2\pi n), 0 \le n \le 24\}$.

- The *main lobe* is centered at ω=0.2π=0.628. It has the proper height=0.5 and width≈0.5.
- The *side lobes* are smaller bumps centered at ω=1,1.25,1.5,1.75,2... They are local peaks in the magnitude of the discrete sinc function.
- The *first side lobe* is the largest side lobe, next to the main lobe, centered at ω=1.
- This resembles a plot of antenna gain vs. angle for a linear array of point sources (which is also a discrete sinc function).

```
X=cos(0.2*pi*[0:24]);w=[0:255]*2*pi/256;
FX=abs(fft(X,256))/25;
subplot(211),plot(w,FX)
```

Another issue that arises is *resolution*. Resolution is the ability to detect two closely-spaced sinusoids in a short data segment. To demonstrate this issue, we use an N-point DFT to compute the spectrum of a segment of length L+1 of the sum of two sinusoids at frequencies ω=0.2π and ω=0.22π:

$$\begin{aligned} x[n] &= \{\cos(0.2\pi n) + \cos(0.22\pi n), 0 \le n \le L\}. \\ X_k &= \sum_{n=0}^{L} x[n]e^{-j2\pi nk/N}. \end{aligned} \quad (8.14)$$

We present three plots:
- L=50 and N=256
- L=50 and N=1024
- L=100 and N=256

Solution:

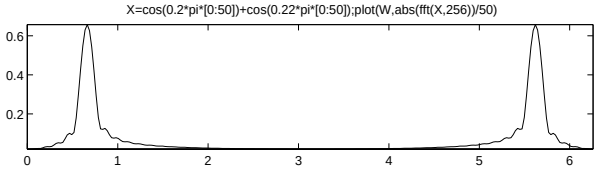


Figure 8.8: Computed spectrum of data segment of sum of two sinusoids for L=50 and N=256.

Only a single peak is present (the second peak is the peak at the conjugate frequency). We have failed to resolve the two sinusoids. Recall that the N-point DFT computes the DTFT $X(e^{j\omega})$ at ω=$2\pi\frac{k}{N}$.

Perhaps using a finer discretization of the DTFT by increasing N (with zero padding) will help:

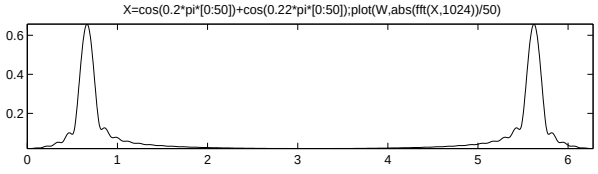


Figure 8.9: Computed spectrum of data segment of sum of two sinusoids for L=50 and N=1024.

Using a finer discretization of the DTFT did not help-the two peaks are fused together in $X(e^{j\omega})$ itself.

Now increase the length of the data segment by using L=100 instead of L=50:

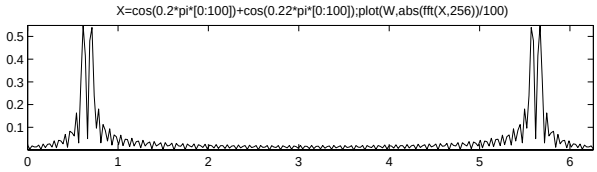


Figure 8.10: Computed spectrum of data segment of sum of two sinusoids for L=100 and N=256.

Now the two peaks have been resolved! Sidelobes are now more apparent. These would be smoother if a larger N were used.

In optics, the resolution of an imaging system is inversely proportional to the aperture size. To resolve a double star, increasing magnification does not help. Increasing the diameter of the telescope lens or mirror does help. The following formula is well-known in optics: To resolve angles ω_1 and ω_2 (measured in radians) requires an aperture size of about L, where

$$|\omega_1 - \omega_2| > \frac{2\pi}{L}. \tag{8.15}$$

This value of L comes from the first zero-crossing of the discrete sinc function, which is at ω=$\frac{2\pi}{L}$. This is only a very rough measure. In practice, L can be smaller than this formula suggests. Here, we have

$$0.22\pi - 0.2\pi = \frac{2\pi}{L} \rightarrow L = 100. \tag{8.16}$$

The above figure shows that the minimum L required to resolve the two peaks is between 50 and 100.

8.2.2 Choices of Data Windows

In all of the above examples, a *rectangular window* was used: $w[n]$=1 for $0 \leq n \leq L$–1. Although the notion of continuity has no place in discrete time, the two sharp jumps at the endpoints of this $w[n]$ cause $W(e^{j\omega})$ to oscillate and decay slowly. To see this, recall that the DTFT is the Fourier dual of the continuous-time Fourier series: $x[n]$ are (roughly) the coefficients of the Fourier series expansion of $W(e^{j\omega})$. This suggests that $w[n]$ should taper gradually to zero at its endpoints. All data windows do this.

The two main measures for data windows are:

- The ratio of the height of the first (and highest) sidelobe to the height of the main lobe. This measures how well the window reduces sidelobes.
- The width of the main lobe. This measures the resolution of the window: the wider the main lobe, the poorer the resolution.
- There is a tradoff between these.

The following table summarizes features of some commonly-used windows. In this table:

- "Formula" is the formula for the window $w[n]$ for $0 \leq n \leq L$. $w[n]$=0 otherwise.
- "Main" is the width of the main lobe.
- "Side" is the ratio $20\log_{10}\frac{1^{\text{st}}\text{ SIDE LOBE}}{\text{MAIN LOBE}}$ (in dB).

Name	Formula	Main	Side
Rectangle	1	$4\pi/L$	–13
Bartlett	$1–2\lvert n-\frac{L}{2}\rvert/L$	$8\pi/L$	–27
Hanning	$0.50–0.50\cos(\frac{2\pi n}{L})$	$8\pi/L$	–32
Hamming	$0.54–0.46\cos(\frac{2\pi n}{L})$	$8\pi/L$	–43
Blackman	$0.42^{-0.50\cos(\frac{2\pi n}{L})}_{+0.08\cos(\frac{4\pi n}{L})}$	$12\pi/L$	–58

The tradeoff between resolution and sidelobe suppression is apparent. A Hamming window is a common choice. And yes, two people named Hanning and Hamming both worked on this problem!

The data segment is multiplied by, not convolved with, the window function.

The following example demonstrates window use.

Example: Using a data window.
Use a 256-point DFT to compute the spectrum of $\{\cos(0.2\pi n), 0 \leq n \leq 24\}$, using (1) a rectangular window; and (2) a Hamming window.

Solution:
The first plot, using a rectangular window, is the same as the first plot in this section. Note the ratio

of the heights of the first sidelobe to the main lobe is about 0.2≈–13 dB.

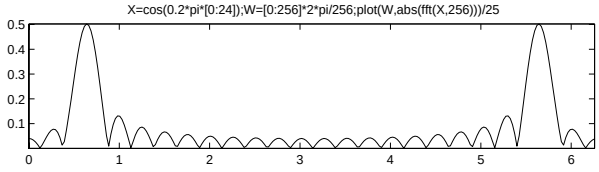


Figure 8.11: Computed spectrum using rectangular.

The second plot, using a Hamming window, has a main peak twice as wide as the first plot. But the sidelobes are now barely apparent.

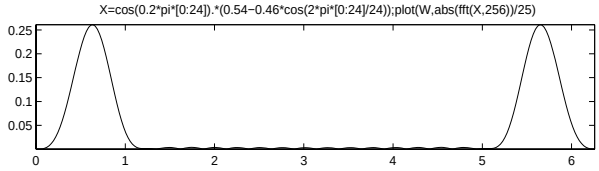


Figure 8.12: Computed spectrum using Hamming.

The Matlab code used for this example:

```
X=cos(0.2*pi*[0:24]);
w=[0:255]*2*pi/256;
FX=abs(fft(X,256))/25;
subplot(211),plot(w,FX)
W=0.54-0.46*cos(2*pi*[0:24]/24);
Y=X.*W;%MULTIPLY by data window.
FY=abs(fft(Y,256))/25;
subplot(212),plot(w,FY)
```

Example: How a data window helps.
A 256-point DFT is used to compute the spectrum of a data segment of length 75. How many sinusoids are present? To determine this, try using both: (1) a rectangular window; and (2) a Hamming window.

Solution:
The left plot, using a rectangular window, seems to suggest that 3 sinusoids are present. The right plot, using a Hamming window, shows that there are actually 4 sinusoids present. The 4^{th} peak was lost in the sidelobes when a rectangular window was used.

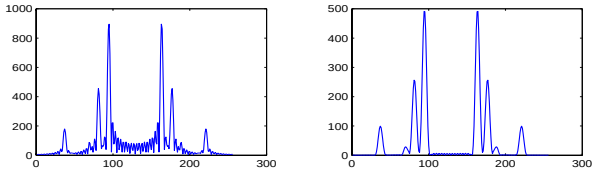


Figure 8.13: Rectangular vs. Hamming.

```
load p8.mat%Includes X1.
FX=abs(fft(X1,256));
Y1=X1.*hamming(75)';
FY=abs(fft(Y1,256));
subplot(221),plot(FX)
subplot(222),plot(FY)
```

- Multiply the data segment and the window function point-by-point. Don't convolve them!
- Increasing the DFT order N does not help to resolve two spectral peaks close together.
- Increasing the data segment length L does help to resolve two spectral peaks close together.
- Using a data window does not help resolve two spectral peaks close together; in fact, it reduces resolution, since the main peak is wider.
- But using a data window reduces sidelobes. This makes the computed spectrum easier to interpret, especially if small peaks are present in it.

8.3 Spectrograms

8.3.1 Problem Statement

Let $x(t)$ have the *time-varying spectrum*

$$\begin{aligned} x(t) &= \sum_{k=1}^{\infty} A_{k1} \cos\left(2\pi \frac{k}{P_1} t + \theta_1\right), T_0 < t < T_1 \\ &= \sum_{k=1}^{\infty} A_{k2} \cos\left(2\pi \frac{k}{P_2} t + \theta_2\right), T_1 < t < T_2 \\ &= \sum_{k=1}^{\infty} A_{k3} \cos\left(2\pi \frac{k}{P_3} t + \theta_3\right), T_2 < t < T_3 \end{aligned}$$

$$\vdots \quad \vdots \quad \vdots \tag{8.17}$$

That is, the line spectrum of $x(t)$ **changes** every so often. The study of signals whose spectra changes with time is *time-frequency analysis*.

Musical signals have this form:

- The *durations* T_{i+1}–T_i are known. All whole notes have the same duration; all half notes have half the duration of whole notes, etc. So we do not have the problem of *segmenting* the signal into different intervals;
- The *frequencies* $\frac{k}{P_i}$ can only take on twelve different values in an octave. So we are only *choosing* from several possible frequencies;

This section introduces the lowest-level form of the *spectrogram*, which is a way of visualizing the time-varying line spectrum of a signal. The spectrogram is a means of interpreting data.

8.3.2 Spectrogram: Motivation

For example, we are given a music signal in the file `victorstone.mat`. All we know about it is that it was sampled at 8192 samples per sec., and that it consists of a sequence of musical notes. We wish to intepret this signal.

The next figure shows its line spectrum. This is not much help. In fact, it constitutes a histogram of the musical notes-the varying heights of the spectral lines provide a measure of how often that note was played, but not when it was played.

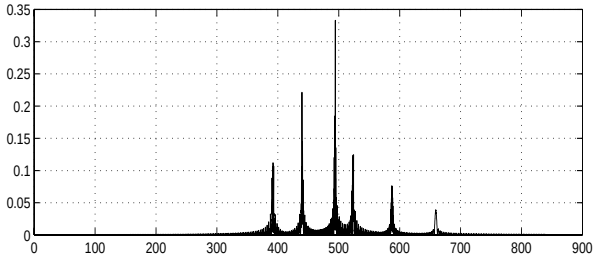


Figure 8.14: Line spectrum of music signal.

The Matlab code for this plot:

```
load victorstone.mat;N=length(X);S=8192;
FX=2/N*abs(fft(X));F=[0:N-1]*S/N;
plot(F(1:8000),FX(1:8000))
```

The next figure shows its *spectrogram*:

- The *height* of each line is the frequency kP_i in Hertz of that time segment of the signal;
- The *length* of each line is the duration T_{i+1}–T_i in seconds of that time segment;
- The *brightness* of each line is the amplitude A_{ki} of that time segment of the signal.

The line heights are at the frequency values used by musical notes. The notes change as time progresses from left to right. You can now recognize this as "The Victors," the fight song of the University of Michigan. Indeed, the spectrogram can function as a crude type of musical notation, indicating the pitch and duration of notes played in succession. It could also function like a player piano roll, moving from right to left.

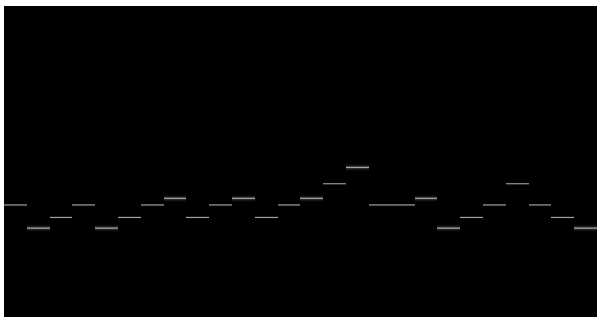


Figure 8.15: Spectrogram of music signal.

8.3.3 Spectrogram: Computation

The Matlab code for this plot:

```
load victorstone.mat;LX=length(X);
L=26;N=LX/L;XX=reshape(X',N,L);
FXX=abs(fft(XX));FXX=FXX(5*N/6:N,:);
subplot(211),imagesc(FXX)
colormap(gray),axis off
```

Details of the spectrogram for this signal:

- `X` had length `LX`=78000. At 8192 samples per sec. this is a duration of $\frac{78000}{8192}$=9.5215 secs.
- `X` was segmented into `L`=26 segments of length `N`=3000 samples each. So each segment has a duration of $\frac{3000}{8192}$=0.3662 secs.
- `X` was laid out by column in the array `XX`:
  `X(1:3000)` in the 1^{st} column of `XX`;
  `X(3001:6000)` in the 2^{nd} column of `XX`;
  `X(75001:78000)` in the 26^{th} column.
- `fft` when applied to an array computes the DFT of each column of the array. So `FXX` is the array of line spectra of each segment of `X`, only laid out vertically, rather than horizontally.
- Only the bottom sixth of FXX is kept. The top half is the mirror image of the lower half, so it should not be shown.
- `imagesc` displays this array as an image. The brightness of each image pixel is `FXX` at that frequency and time.

How did we know to segment `X` into 26 segments? If we know the signal is a musical signal consisting of whole notes with durations 0.3662 secs., we know that the number of notes is $\frac{9.5215}{0.3662}$=26 since the duration of the signal is 9.5215 secs.

One can plot a crude version of the spectrogram of a signal consisting of `L` segments of lengths `N` each, where `N=length(X)/L`, using

```
imagesc(abs(fft(reshape(X',N,L))))
```

To get a 3-D plot of the spectrogram, use

```
waterfall(abs(fft(reshape(X',N,L))))
```

Using this command, the segments do not overlap, and a rectangular data window is used. The Matlab command `specgram` allows the segments to overlap and other data windows to be used.

8.3.4 Time vs. Frequency Resolution Tradeoff

We don't have to know how many segments the signal actually contains. In fact, varying

- The number of segments `L` and
- The length `N` of each segment so
- `L*N=LX`=total length of the signal.

trades off time and frequency resolution. The larger `L` is, the shorter the length `N` of each segment, so changes in the spectrum as it changes in time can be tracked faster. However, `fft` computes line spectra at frequencies f at Matlab indices `K`, where

$$f = (K-1)/P = (K-1)(S/N). \tag{8.18}$$

P=$\frac{N}{S}$ is now the duration of each interval. So the discretization (resolution) of frequency is coarser.

To illustrate this, recompute the above spectrogram using different values of `L` and hence `N`:

- `L`=13 and `N=length(X)/L`=6000;
- `L`=104 and `N=length(X)/L`=750.

Figure 8.16: Spectrogram with L=13 segments.

Each segment now contains two notes, and the spectrogram plots both of them. If you were playing "The Victors" from this, you would have to guess which note to play in each segment!

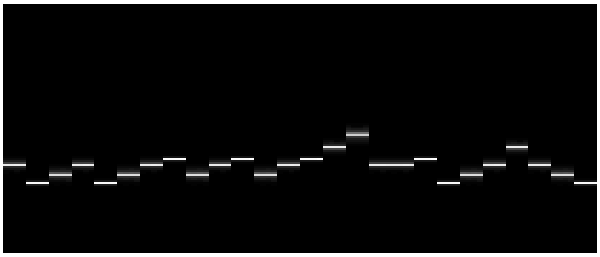


Figure 8.17: Spectrogram with L=104 segments.

Each segment is only 750 samples long, and the

frequencies are smeared out. This improves visibility, but is not so good for actually computing the frequencies of each segment.

8.3.5 Chirp Signal

A common test signal for time-frequency analysis is the *chirp*, which does indeed sound like a bird chirp (dolphin clicks are also chirps).

$$x(t) = A\cos(2\pi F t^2) \text{ for } t > 0. \tag{8.19}$$

The chirp $\cos(t^2)$ is plotted in the next figure.

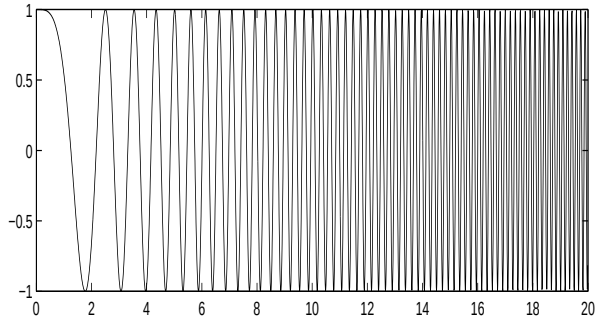


Figure 8.18: Chirp Signal $\cos(t^2)$.

The chirp signal looks like a sinusoid whose frequency is steadily increasing in time.

Its spectrogram is plotted in the next figure.

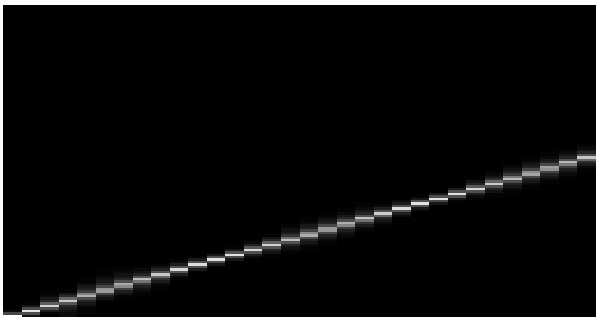


Figure 8.19: Spectrogram of chirp.

This clearly indicates a signal whose frequency is increasing *linearly* with time. The spectrogram makes interpretation of the signal easy.

The Matlab code for these plots:

```
X=cos([0:8191].*[0:8191]/10000);
T=linspace(0,19.99,2000);
subplot(211),plot(T,X(1:2000))
FXX=abs(fft(reshape(X',256,32)));
FXX=FXX(129:256,:);
figure,subplot(211),imagesc(FXX)
axis off,colormap(gray)
```

8.3.6 Interpretation of Chirp

How do we interpret exactly what is going on in the spectrogram of the chirp?

- $\cos(t^2)$ is sampled at 100 samples per sec. Setting $t{=}\frac{n}{100}$ in $\cos(t^2)$ gives $\cos(n^2/10000)$.
- The length of `X` is 8192 samples, so its duration is $\frac{8192}{100}$=81.92 sec (almost 1.5 minutes!).
- The height of the right-most line in the spectrogram is index `K`=67. The frequency at this time is $f{=}(K{-}1)\frac{S}{N}{=}(67{-}1)\frac{100}{256}$=25.8 Hertz since each segment is 256 samples long.
- *Instantaneous frequency* of $\cos(2\pi f t^2)$ is $2ft$, not ft as $\cos(2\pi f t^2){=}\cos(2\pi(ft)t)$ suggests. Here, $f{=}\frac{1}{2\pi}$, so the instantaneous frequency at t=81.92 is $\frac{2(81.92)}{2\pi}$=26.1 Hertz. This is higher than the spectrogram value of 25.8 Hertz since the spectrogram averages over the final segment, instead of using the value at its end of the segment.

8.3.7 Removing Interference

To show how a spectrogram can help in removing an interfering signal, run the following Matlab code:

```
load victorstone.mat;LX=length(X);S=8192;
X=X+cos(2*pi*700*[1:LX]/S);soundsc(X,S)
%Plot its spectrum and spectrogram using:
F=[0:LX-1]*S/LX;FX=2/LX*abs(fft(X));
plot(F(1:9000),FX(1:9000))
L=26;N=LX/L;XX=reshape(X',N,L);
FXX=abs(fft(XX));FXX=FXX(5*N/6:N,:);
subplot(211),imagesc(FXX)
colormap(gray),axis off
```

The spectrum and spectrogram are plotted next:

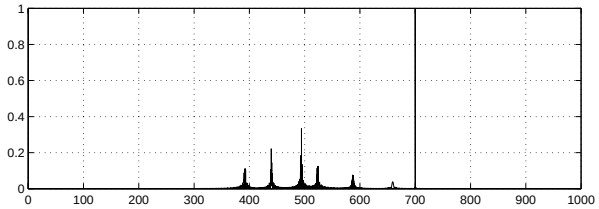


Figure 8.20: Spectrum: Victors+interference.

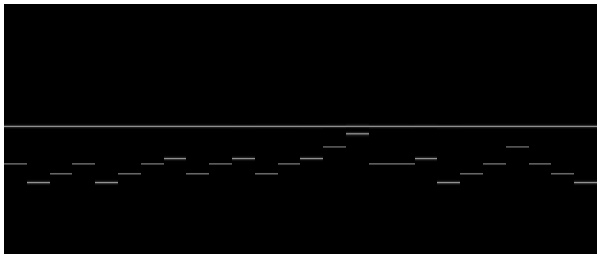


Figure 8.21: Spectrogram: Victors+interference.

The spectrum and spectrogram *together* show that the interference is a tone at 700 Hertz that has been added to the tonal version of "The Victors."

To eliminate this interference, run this code:

```
K=1+round(LX*700/S);KK=[K-100:K+100];
FX=fft(X);FX(KK)=0;FX(LX+2-KK)=0;
Y=real(ifft(FX));soundsc(Y,S)
```

- `K=1+round(LX*700/S);` comes from solving `f=(K-1)S/LX` for the Matlab index `K`;
- We set not only `FX(K)` to zero, but also all `FX` values whose indices are within 100 of `K`. We also set to zero their complex conjugate values;
- Listen to `Y`: The interference is now gone!

Example: Two songs together.

For a more dramatic example of removing more noxious interference, run the following Matlab code:

```
load victorsmsu.mat;LZ=length(Z);S=8192;
F=[0:LZ-1]*S/LZ;FZ=2/LZ*abs(fft(Z));
subplot(211),plot(F(1:9000),FZ(1:9000))
L=130;N=LZ/L;ZZ=reshape(Z',N,L);
FZZ=abs(fft(ZZ));FZZ=FZZ(5*N/6:N,:);
figure,subplot(211),imagesc(FZZ)
colormap(gray),axis off
```

The spectrum and spectrogram are plotted next:

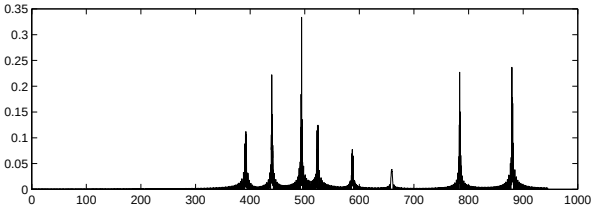


Figure 8.22: Spectrum: Victors+interference.

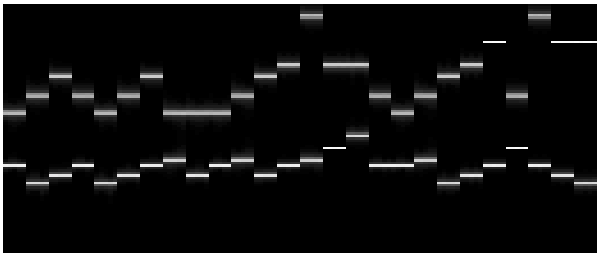


Figure 8.23: Spectrogram: Victors+interference.

The signal sounds like a calliope. Listening carefully, the signal is "The Victors" and another song being played simultaneously. The spectrum is of little help here. But the spectrogram shows that the other song is one octave above "The Victors." It is the MSU fight song, which is noxious interference!

To eliminate this interference, run this code:

```
K=1+round(LZ*700/S);
KK=[K:LZ+2-K];FZ=fft(Z);FZ(KK)=0;
Y=real(ifft(FZ));soundsc(Y,S)
```

"The Victors" spectrum is below 700 Hertz, so now we set all of the spectrum above 700 Hertz to zero. Listen to `Y`: The interference is now gone!

Chapter 9

Discrete-Time Filter Design

In a previous chapter, we showed how placing poles and zeros can be used to design notch and comb filters. But an attempt to design a half-band lowpass filter showed that placing poles and zeros is not the best way to design other types of filters.

This chapter covers design procedures for discrete-time filters. There are two major types of filters:

- FIR filters, in which $h[n]$ has finite duration;
- IIR filters, in which $h[n]$ has infinite duration.

9.1 FIR Filter Design

9.1.1 Overview

An FIR filter has the form

$$y[n] = b_0x[n] + b_1x[n-1] + \ldots + b_Nx[n-N]. \quad (9.1)$$

The advantages of using an FIR filter are as follows:

- FIR filters are always stable, unlike IIR filters;
- FIR filters have a transient response that is zero after a finite time, unlike IIR filters;
- FIR filters have no phase distortion.

There are 3 major approaches to FIR filter design:

- *Windowing* the ideal filter computed using the inverse DTFT, using (say) a Hamming window;
- *Frequency sampling*, in which the desired frequency response is attained exactly at a finite number of frequencies, usually equally spaced;
- *Equiripple*, in which the iterative Parks-McClellan algorithm is used to minimize the maximum (minimax) absolute weighted error.

Before presenting these approaches, we present some forms of the desired (ideal) filter frequency response.

9.1.2 Forms of Desired Frequency Response Functions

$H_D(e^{j\omega})$ is the Desired frequency response function: For a low-pass filter with cutoff frequency ω_o:

$$H_D(e^{j\omega}) = \begin{cases} 1 & \text{for } 0 \le |\omega| < \omega_o \\ 0 & \text{for } \omega_o < |\omega| < \pi \end{cases} \quad (9.2)$$

For a band-pass filter with cutoff frequencies ω_L, ω_H:

$$H_D(e^{j\omega}) = \begin{cases} 0 & \text{for } 0 \le |\omega| < \omega_L \\ 1 & \text{for } \omega_L < |\omega| < \omega_H \\ 0 & \text{for } \omega_H < |\omega| < \pi \end{cases} \quad (9.3)$$

For an ideal differentiator:

$$H_D(e^{j\omega}) = j\omega \text{ for } |\omega| < \pi \quad (9.4)$$

For a Hilbert transform:

$$H_D(e^{j\omega}) = \begin{cases} -j & \text{for } 0 < \omega < \pi \\ j & \text{for } -\pi < \omega < 0 \end{cases} \quad (9.5)$$

- $H_D(e^{j\omega})$ for the low-pass and band-pass filters is real and even. So $h[n]$ should be real and even.
- $H_D(e^{j\omega})$ for the differentiator and Hilbert transform is pure imaginary and odd. So $h[n]$ should be real and odd.

- $H_D(e^{j\omega})$ is periodic with period 2π, as always.
- We will design $h[n]$ to be real, even or odd, and noncausal. Then we delay it by half its length to make it causal.

Specifically, we design $h[n]$ to have the form

- $h[n] = \{h[-L], \ldots, h[-1], \underline{h[0]}, h[1], \ldots h[L]\}$
- OR $= \{-h[-L], \ldots, -h[-1], \underline{0}, h[1], \ldots h[L]\}$

and then implement not the noncausal $h[n]$ but the causal $\tilde{h}[n]$, which is $h[n]$ delayed by L:

$$\tilde{h}[n] = h[n - L] \tag{9.6}$$

since by time-invariance we have

- $x[n] \rightarrow \boxed{h[n]} \rightarrow y[n]$ implies
- $x[n] \rightarrow \boxed{h[n–L]} \rightarrow y[n–L]$.

The causal $\tilde{h}[n]$ can be implemented as in (9.1) with N=2L. The output will be delayed by L, but this won't matter. At the standard CD sampling rate of 44100 samples per sec., L=100 is only $\frac{1}{441} \approx 2$ msec. We also used a delayed impulse response for non-minimum-phase inverse systems. Also note we can halve the number of multiplications by adding $x[n]$ and $\pm x[N–n]$ before multiplying by b_n=$\pm b_{N-n}$.

9.1.3 Linear Phase

Designing the filter this way avoids the issue of linear phase, but it should be mentioned here. Recall that the DTFT of a signal delayed by D is the DTFT of the signal multiplied by $e^{-j\omega D}$:

$$\text{dtft}(x[n - D]) = X(e^{j\omega})e^{-j\omega D}. \tag{9.7}$$

The phase of the DTFT becomes

$$\arg[X(e^{j\omega})e^{-j\omega D}] = \arg[X(e^{j\omega})] - \omega D \tag{9.8}$$

so that the phase is altered by a term proportional to the frequency ω; hence the term *linear phase*. This is a bit of a misnomer–the phase is actually linear with occasional jumps of $\pm\pi$ in it, since if $X(e^{j\omega})$ is

$$\begin{matrix}\text{REAL}\\ \text{EVEN}\end{matrix} \rightarrow \arg[X(e^{j\omega})] = \begin{cases} 0 & \text{if } X(e^{j\omega}) > 0 \\ \pi & \text{if } X(e^{j\omega}) < 0 \end{cases} \tag{9.9}$$

$$\begin{matrix}\text{IMAG}\\ \text{ODD}\end{matrix} \rightarrow \arg[X(e^{j\omega})] = \begin{cases} \frac{\pi}{2} & \text{if } \frac{1}{j}X(e^{j\omega}) > 0 \\ -\frac{\pi}{2} & \text{if } \frac{1}{j}X(e^{j\omega}) < 0 \end{cases}$$

So at frequencies where $X(e^{j\omega})$ changes sign, which happens when $X(e^{j\omega})$=0, the phase jumps by $\pm\pi$. While phase angles $\pi \equiv -\pi$, the sign should be chosen to keep $\arg[X(e^{j\omega})]$ an odd function. These jumps in phase do not constitute phase distortion, since they are just $X(e^{j\omega})$ changing sign.

9.1.4 Forms of Linear Phase Filters

There are 4 types of linear phase filters, depending on whether $h[n]$ is symmetric or antisymmetric, and depending on whether its length is odd or even:

Type	Form	Restriction
I	$\{b, a, c, a, b\}$	None
II	$\{b, a, a, b\}$	$H(e^{j\pi})$=0
III	$\{b, a, 0, -a, -b\}$	$H(e^{j0,j\pi})$=0
IV	$\{b, a, -a, -b\}$	$H(e^{j0})$=0

You do *not* have to memorize this utterly ridiculous "Type I-IV" notation!

The restrictions are of some interest, since they follow automatically from the forms of $h[n]$:

- $h[n]$=$\{b, a, a, b\} \rightarrow H(e^{j\pi})$=$b$–$a$+$a$–$b$=0.
- $h[n]$=$\{b, a, 0, -a, -b\} \rightarrow H(e^{j\pi})$=$b$–$a$+$a$–$b$=0.
- $h[n]$ antisymmetric$\rightarrow H(e^{j0})$=0.

So using a type II filter automatically satisfies the low-pass filter criterion of rejecting ω=π, and using a type IV filter automatically satisfies the high-pass filter criterion of rejecting ω=0. But the savings here are only minimal. We will consider only types I and III (which have odd lengths) in these notes.

9.1.5 Design by Windowing

The idea behind designing an FIR filter using windowing is to use a data window on the ideal impulse

response. That is,

1. Compute $h_D[n]=\frac{1}{2\pi}\int_{-\pi}^{\pi} H_D(e^{j\omega})e^{j\omega n}d\omega$.
2. Compute $h[n]=h_D[n]w[n]$ for data window $w[n]$.
3. $w[n]$ should be advanced in time so that $w[n]\neq 0$ for $|n|\leq L$, not the usual $0\leq n\leq 2L$.
4. Implement the causal $\tilde{h}[n]=h[n–L]$.

Example: Window design of differentiator
Design an FIR differentiator using:

1. A 5-point rectangular window.
2. A 5-point Hamming window.

Solution:
From the DTFT chaper, we have

$$H_D(e^{j\omega}) = j\omega, |\omega| < \pi. \tag{9.10}$$

So $h_D[n]$ is computed from $H_D(e^{j\omega})$ as

$$\begin{aligned} h_D[n] &= \int_{-\pi}^{\pi} H_D(e^{j\omega})e^{j\omega n}\frac{d\omega}{2\pi} \\ &= \int_{-\pi}^{\pi} j\omega e^{j\omega n}\frac{d\omega}{2\pi} \\ &= \begin{cases} \frac{(-1)^n}{n} & n\neq 0 \\ 0 & n=0 \end{cases} \\ &= \{\ldots\frac{1}{3},-\frac{1}{2},1,0,-1,\frac{1}{2},-\frac{1}{3}\ldots\} \end{aligned} \tag{9.11}$$

The rectangular $w_R[n]$ and Hamming $w_H[n]$ windows, advanced in time to be centered at n=0, are

$$\begin{aligned} w_R[n] &= \{1,1,\underline{1},1,1\} \\ w_H[n] &= 0.54+0.46\cos\left(\frac{2\pi n}{5-1}\right) \\ &= \{0.08,0.54,\underline{1},0.54,0.08\} \end{aligned} \tag{9.12}$$

Then the two FIR differentiators are:

$$\begin{aligned} h_R[n] &= \{-1/2,1,\underline{0},-1,1/2\} \\ \tilde{h}_R[n] &= \left\{\underline{-1/2},1,0,-1,1/2\right\} \\ h_H[n] &= \{-0.04,0.54,\underline{0},-0.54,0.04\} \\ \tilde{h}_H[n] &= \{\underline{-0.04},0.54,0,-0.54,0.04\} \end{aligned} \tag{9.13}$$

9.1.6 Design by Frequency Sampling

The idea behind designing an FIR filter using frequency sampling is to choose $h[n]$ so that the DTFT of $h[n]$ equals $H_D(e^{j\omega})$ at the $2L$+1 points $\omega=\frac{2\pi k}{2L+1}$ for k=$-L,\ldots L$. Specifically, we solve the equations

$$\sum_{n=-L}^{L} h[n]e^{-j\frac{2\pi nk}{2L+1}} = H_D(e^{j\frac{2\pi k}{2L+1}}) \tag{9.14}$$

for k=$-L,\ldots L$. By conjugate symmetry, this is actually L+1 equations in L+1 unknowns. The following example demonstrates frequency sampling.

Example: Design of lowpass filter using frequency sampling.
Design an FIR lowpass filter of length 5 using frequency sampling and a Type I filter. Use these specs:

- $\underline{\omega=0}: H(e^{j0})=1$;
- $\underline{\omega=\pi}: H(e^{j\pi})=0$;
- $\underline{\omega=\frac{\pi}{2}}: H(e^{j\frac{\pi}{2}})=\frac{3}{4}$.

Solution:
A type I filter has the general form
$h[n]$=$\{a,b,\underline{c},b,a\}$. The DTFT of $h[n]$ is

$$H(e^{j\omega}) = c+2b\cos(\omega)+2a\cos(2\omega). \tag{9.15}$$

$H(e^{j\omega})$ evaluated at ω=$0,\pm\frac{\pi}{2},\pm\frac{2\pi}{2}$ is

- $\underline{\omega=0}: H(e^{j0})=1=c+2b+2a$
- $\underline{\omega=\pi}: H(e^{j\pi})=0=c-2b+2a$
- $\underline{\omega=\frac{\pi}{2}}: H(e^{j\frac{\pi}{2}})=\frac{3}{4}=c-2a$

Solving these three equations in three unknowns gives

$$a=-1/16; \quad b=1/4; \quad c=5/8 \tag{9.16}$$

so the answer is

$$\begin{aligned} h[n] &= \left\{-\frac{1}{16},\frac{1}{4},\frac{5}{\underline{8}},\frac{1}{4},-\frac{1}{16}\right\} \\ \tilde{h}[n] &= \left\{\underline{-\frac{1}{16}},\frac{1}{4},\frac{5}{8},\frac{1}{4},-\frac{1}{16}\right\} \end{aligned} \tag{9.17}$$

By choosing the frequencies to exclude ω=π, and some reordering, the solution to the linear system of equations can be replaced with an inverse DFT. The following example illustrates how this can be done.

Example: Design of lowpass filter using frequency sampling, revisited.
Redo the previous example, now using the specs:

- $\underline{\omega = 0} : H(e^{j0}) = 1;$
- $\underline{\omega = \frac{2\pi}{5}} : H(e^{j\frac{2\pi}{5}}) = 1;$
- $\underline{\omega = \frac{4\pi}{5}} : H(e^{j\frac{4\pi}{5}}) = 0.$

Solution:
The DTFT of $h[n]$ is still

$$H(e^{j\omega}) = c + 2b\cos(\omega) + 2a\cos(2\omega). \qquad (9.18)$$

$H(e^{j\omega})$ evaluated at ω=0,$\pm\frac{2\pi}{5}, \pm\frac{4\pi}{5}$ now becomes

- $\underline{\omega = 0} : H(e^{j0}) = 1 = c + 2b + 2a$
- $\underline{\omega = \frac{2\pi}{5}} : H(e^{j\frac{2\pi}{5}}) = 1 = c + 2(0.309)b - 2(0.809)a$
- $\underline{\omega = \frac{4\pi}{5}} : H(e^{j\frac{4\pi}{5}}) = 0 = c - 2(0.809)b + 2(0.309)a$

Solving these three equations in three unknowns gives

$$a = -0.1236; \quad b = 0.3236; \quad c = 0.6000. \qquad (9.19)$$

But the system of equations is now a 5-point DFT. So they can be solved using an inverse 5-point DFT:

`real(ifft([1 1 0 0 1]))` gives $\{c, b, a, a, b\}$

as computed above. Note the ordering:

- We must compute the inverse DFT of $\{X_0, X_1, X_2, X_3 = X_2^*, X_4 = X_1^*\}$.
- The result is a single period of the periodic extension of $h[n]$: $\{h[0], h[1], h[2], h[3] = h[-2], h[4] = h[-1]\}$.
- This is *not* the same as the desired filter $\{h[-2], h[-1], h[0], h[1], h[2]\}$ but a circular shift of it by half its length.
- This can be performed using `fftshift`. So:
- $h[n] = \{-0.1236, 0.3236, \underline{0.6}, 0.3236, -0.1236\}$
- $\tilde{h}[n] = \{\underline{-0.1236}, 0.3236, 0.6, 0.3236, -0.1236\}$.

In the previous example. the design specs constituted a 4-point DFT, so an inverse DFT gives the *aliased* solution (a is doubled from its correct value)

$$a = -1/8; \quad b = 1/4; \quad c = 5/8. \qquad (9.20)$$

9.1.7 Design using Equiripple

The idea behind designing an equiripple FIR filter is to compute the $h[n]$ satisfying the *minimax* criterion

$$\underset{\text{h[n]}}{\text{MIN}}\ \underset{\omega}{\text{MAX}}\ \left\{|E(e^{j\omega})|W(e^{j\omega})\right\} \qquad (9.21)$$

where the error $E(e^{j\omega})$ and weight $W(e^{j\omega})$ are

$$\begin{aligned} E(e^{j\omega}) &= H_D(e^{j\omega}) - \sum_{n=0}^{N-1} h[n]e^{-j\omega n} \qquad (9.22)\\ W(e^{j\omega}) &= \text{weight to penalize error at } \omega \end{aligned}$$

The way to interpret this minimax criterion is as a game between you, the designer, and the math of discrete-time filter performance;

- Math plays 2^{nd}, and will choose the ω that makes the weighted error as large as possible.
- You play 1^{st}, and know that math will take the worst case (value of ω) for your design.
- So you play (choose $h[n]$) to minimize the worst case (minimum maximum error magnitude).

This can be done using an iterative algorithm developed in 1972 by Parks and McClellan at Georgia Tech. We will skip the approximation theory behind this algorithm (Remez exchange and alternation theorems, among other things) and simply show how to implement it using the Matlab command `firpm`.

Example: Equiripple design using `firpm`.

Design a bandpass filter having the specs:

$$H(e^{j\omega}) = \begin{cases} 0 & \text{for } 0.0\pi \le \omega < 0.2\pi \\ dc & \text{for } 0.2\pi < \omega < 0.3\pi \\ 1 & \text{for } 0.3\pi < \omega < 0.7\pi \\ dc & \text{for } 0.7\pi < \omega < 0.8\pi \\ 0 & \text{for } 0.8\pi < \omega < 1.0\pi \end{cases} \quad (9.23)$$

where dc="don't care," and having duration 21.

Solution:
The "don't cares" are *transition bands* between the *passband* (gain=1) and the two *stopbands* (gain=0). The order=20 of the filter is one less than its length. The Matlab program for designing this filter is

```
F=[0.0 0.2 0.3 0.7 0.8 1.0];
G=[0.0 0.0 1.0 1.0 0.0 0.0];
H=firpm(20,F,G);freqz(H)
```

- `F` is a vector of pairs of frequencies $\frac{\omega}{\pi}$.
  `F` must have even length, and include 0 and 1.
- `G` is a vector of gains $H(e^{j\omega})$ at frequencies `F`.
- `firpm` designs a filter whose gain varies linearly: between `(F(1)`π`,G(1))` and `(F(2)`π`,G(2))`; and between `(F(3)`π`,G(3))` and `(F(4)`π`,G(4))`; etc. The computed impulse response is stored in `H`.

The default design is a symmetric $h[n]$, suitable for filters. To design an antisymmetric $h[n]$, use either

- `H=firpm(20,[0 1],[1 1],'hilbert');` or
- `firpm(20,[0 1],[0 pi],'differentiator')`
  The latter uses $W(e^{j\omega})=\frac{1}{\omega}$ to penalize heavily errors at low frequencies, to minimize $\frac{H(e^{j\omega})-j\omega}{j\omega}$.

9.2 IIR Filter Design

9.2.1 Overview

An IIR filter has infinite duration, by definition.

The philosophy behind IIR filter design is to take a continuous-time filter, such as a Butterworth filter, and transform it to a discrete-time filter.

The advantages of using an IIR filter are as follows:

- IIR filters can attain sharper transitions than FIR filters using the same number of coefficients, e.g., MA (FIR) vs. ARMA (IIR) notch filters.
- While IIR filters can be unstable, the design procedures to follow guarantee a stable IIR filter.
- IIR filters, unlike FIR filters, need not be delayed to make them causal.

There are 2 major approaches to IIR filter design:

- *Impulse invariance*: the continuous-time impulse response is sampled to a discrete-time one;
- *Bilinear transform*: the continuous-time transfer function is mapped to a discrete-time one.

9.2.2 Impulse Invariance

The idea behind designing an IIR filter using impulse invariance is to sample the impulse response of a suitable continuous-time filter. Specifically,

- Select a suitable continuous-time filter, such as a Butterworth filter, with transfer function $H_a(s)$.
- Compute its continous-time impulse response $h_a(t)$=$\mathcal{L}^{-1}\{H_a(s)\}$ (inverse Laplace transform).
- Let $h[n]$=$Th_a(nT)$ for sampling interval T.

Multiplication by T is required dimensionally: $h_a(t)$ has units of $\frac{1}{\text{TIME}}$, since $\int_{-\infty}^{\infty}\delta(t)dt$=1 and dt has units of time. For example, the impulse response of an RC circuit is $h_a(t)$=$\frac{1}{RC}e^{-\frac{t}{RC}}u(t)$.

To show that $h[n]$ is stable if $h_a(t)$ is stable, let $H_a(s)$ be stable and a rational function. Then

$$h_a(t) = \sum_{i=1}^{N} A_i e^{p_i t} u(t) \quad (9.24)$$

where $p_i = -a_i + jb_i$ are the poles of $H_a(s)$.

Since $H_a(s)$ is stable, Re$[p_i] < 0$ so $a_i > 0$. Then

$$h[n] = Th_a(nT) = \sum_{i=1}^{N} (A_i T)(e^{p_i T})^n u[n] \quad (9.25)$$

and the discrete-time poles have

$$|e^{p_i T}| = |e^{-a_i T}| \cdot |e^{jb_i T}| = e^{-a_i T} < 1 \qquad (9.26)$$

The discrete-time system is stable since $a_i > 0$.

Example: Impulse Invariance IIR Design
Given T=0.001 and the one-pole lowpass filter

- $H_a(s)=\frac{1000}{s+1000} \rightarrow |H_a(j\Omega)|=\frac{1000}{\sqrt{\Omega^2+10^6}}$.

Use impulse invariance to design a filter.

Solution:

- $h_a(t)=\mathcal{L}^{-1}\{H_a(s)\}=1000e^{-1000t}u(t)$.
- $h[n]=(0.001)(1000)e^{-(1000)(0.001)n}=e^{-n}u[n]$.
- $H(z)=Z(e^{-n}u[n])=\frac{z}{z-0.368}$ (a crude low-pass).

Example: Impulse Invariance IIR Design
Given the filter with a sharp resonant peak

- $H_a(s)=\frac{s+0.1}{(s+0.1)^2+9} \rightarrow$ poles $\{-0.1 \pm j3\}$.
- $H_a(j\Omega)$ has a sharp peak at Ω=3.

Use impulse invariance to design a filter.

Solution:

- $h_a(t)=\mathcal{L}^{-1}\{H_a(s)\}=e^{-0.1t}\cos(3t)u(t)$.
- $h[n]=Te^{(-0.1T)n}\cos((3T)n)u[n]$ for any T.
- For small T, $H(z)$ has a pole close to $|z|$=1 so $|H(e^{j\omega})|$ also has a sharp resonant peak.

9.2.3 Bilinear Transformation

The idea behind designing an IIR filter using bilinear transformation is to map the given continuous-time transfer function $H_a(s)$ to the discrete-time transfer function $H(z)$ using the bilinear transformation

$$\begin{aligned} s &= \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} = \frac{2}{T}\frac{z-1}{z+1}. \qquad (9.27) \\ H(z) &= H_a\left(\frac{2}{T}\frac{z-1}{z+1}\right). \end{aligned}$$

Example: Bilinear Transform IIR Design
Given T=0.001 and the one-pole lowpass filter

- $H_a(s)=\frac{1000}{s+1000} \rightarrow |H_a(j\Omega)|=\frac{1000}{\sqrt{\Omega^2+10^6}}$.

Use bilinear transformation to design a filter.

Solution:

$$\begin{aligned} s &= \frac{2}{0.001}\frac{z-1}{z+1} = 2000\frac{z-1}{z+1}. \qquad (9.28) \\ H(z) &= H_a\left(2000\frac{z-1}{z+1}\right) \\ &= \frac{1000}{2000\frac{z-1}{z+1}+1000}\left[\frac{z+1}{z+1}\right] \\ &= \frac{z+1}{2(z-1)+(z+1)} = \frac{z+1}{3z-1} \end{aligned}$$

which is a (crude) lowpass filter.

Bilinear transformation gets its name from being the ratio of two linear functions. It has the properties:

- Maps Re[s] < 0 to $|z| < 1$ and vice-versa;
  Maps Re[s]=0 to $|z|$=1 and vice-versa.

The first property shows that the discrete-time filter is stable if the continuous-time filter is stable. To derive this property, solve (9.27) for z in terms of s:

$$s = \frac{2}{T}\frac{z-1}{z+1} \rightarrow z = \frac{1+sT/2}{1-sT/2} \qquad (9.29)$$

and let s=$-a$+jb where $a > 0$. This gives

$$z = \frac{(1-aT/2)+j(bT/2)}{(1+aT/2)-j(bT/2)} \qquad (9.30)$$

and the squared magnitude of z is

$$|z|^2 = \frac{(1-aT/2)^2+(bT/2)^2}{(1+aT/2)^2+(bT/2)^2}. \qquad (9.31)$$

Now, $a, T > 0$ implies $-aT < aT$. Adding 1+$(\frac{aT}{2})^2$ shows (1–$aT/2)^2 <$ (1+$aT/2)^2$. So the numerator of (9.31) is smaller than the denominator of (9.31), and $|z|^2 < 1$. So the left half-plane of s is mapped to the interior of the unit circle $|z|$=1, and bilinear transformation, like impulse invariance, preserves stability.

The second property follows from setting a=0 in the above derivation, so that s=jb. This gives

$$|z|^2 = \frac{1+(bT/2)^2}{1+(bT/2)^2} = 1 \qquad (9.32)$$

and the imaginary axis of s is mapped to $|z|$=1.

Going the other way, rewrite (9.27) as

$$\begin{aligned} s &= \frac{2}{T}\frac{z-1}{z+1}\left[\frac{z^*+1}{z^*+1}\right] = \frac{2}{T}\frac{zz^*+z-z^*-1}{(z+1)(z^*+1)} \\ &= \frac{2}{T}\frac{|z|^2-1}{|z+1|^2} + j\frac{4}{T}\frac{\text{Imag}[z]}{|z+1|^2} \end{aligned} \tag{9.33}$$

since xx^*=$|x|^2$ and x–x^*=$j2\cdot$Imag$[x]$ for any complex number x. Then, in particular, we have

$$\text{Real}[s] = \frac{2}{T}\frac{|z|^2-1}{|z+1|^2} < 0 \tag{9.34}$$

if and only if $|z| < 1$. So bilinear transformation also maps the inside of the unit circle $|z|$=1 to the left half-plane Real$[s] < 0$, thus preserving stability.

For the second property, let z=$e^{j\omega}$. Then

$$\begin{aligned} s &= \frac{2}{T}\frac{e^{j\omega}-1}{e^{j\omega}+1} \\ &= \frac{2}{T}\frac{e^{j\omega/2}-e^{-j\omega/2}}{e^{j\omega/2}+e^{-j\omega/2}}\left[\frac{e^{j\omega/2}}{e^{j\omega/2}}\right] \\ &= \frac{2}{T}\frac{2j\sin(\omega/2)}{2\cos(\omega/2)} = j\frac{2}{T}\tan(\omega/2) = j\Omega. \end{aligned} \tag{9.35}$$

on Re$[s]$=0. This is the *(pre)warping* formula

$$\Omega = (2/T)\tan(\omega/2) \tag{9.36}$$

which shows that bilinear transformation maps

- Continuous-time frequency $0 \le \Omega < \infty$ to discrete-time frequency $0 \le \omega < \pi$.

This allows us to choose a continuous-time frequency Ω_o to map to a desired discrete-time frequency ω_o.

Example: IIR Design using prewarping.
Given the filter with a sharp resonant peak

- $H_a(s)$=$\frac{s+0.1}{(s+0.1)^2+16}$ → poles $\{-0.1 \pm j4\}$. $H_a(j\Omega)$ has a sharp peak at Ω=4.
- Use impulse invariance to design a filter with a sharp resonant peak at $\omega = \frac{\pi}{2}$.

Solution:

We must choose T to map Ω=4 to ω=$\frac{\pi}{2}$ using the prewarping formula (9.36):

$$\begin{aligned} \Omega &= (2/T)\tan(\omega/2) \\ 4 &= (2/T)\tan(\pi/2/2) = (2/T). \end{aligned} \tag{9.37}$$

which has the solution T=$\frac{1}{2}$. Then we use

$$\begin{aligned} s &= \frac{2}{1/2}\frac{z-1}{z+1} = 4\frac{z-1}{z+1}. \\ H(z) &= H_a\left(4\frac{z-1}{z+1}\right) \\ &= \frac{4\frac{z-1}{z+1}+0.1}{\left(4\frac{z-1}{z+1}+0.1\right)^2+16}\left[\frac{100(z+1)^2}{100(z+1)^2}\right] \\ &= \frac{\frac{1}{8}z^2+0.0061z-0.1189}{z^2+0.0006z+0.9512} \end{aligned} \tag{9.38}$$

$H(z)$ has poles at $\pm j0.9753$=$0.9753e^{\pm j\pi/2}$.
$H(z)$ has zeros at ± 0.952=$0.976e^{j0,j\pi}$.
$H(e^{j\omega})$ has sharp peaks at $\pm\frac{\pi}{2}$, as desired, and dips to zero at ω=0 and π, mimicing $H(s)$.

The algebra can be performed in Matlab using

```
[B A]=bilinear([1 0.1],[1 0.2 16.01],2);
```

which gives the output

```
B=[0.125 0.0061 -0.1189]
A=[1.000 0.0006 0.9512]
```

which are the coefficients in $H(z)$.
Note `bilinear` uses $\frac{1}{T}$=2, not T=$\frac{1}{2}$ directly.

9.3 Results of Different Approaches to Filter Design

Example: Lowpass filter designs.
Design a half-band (cutoff frequency=$\frac{\pi}{2}$) lowpass filter of length 21 using the following approaches:

- Windowing using a Hamming window;
- Frequency sampling using inverse DFT;
- Equiripple with transition 0.4π to 0.6π;

- Discrete-time 10^{th}-order Butterworth filter. This is the bilinear transformation of the continuous-time Butterworth filter, and it has 21 coefficients, matching the other designs.

Solution:
The following Matlab code

```
clear;W=2*pi*[0:104]/210;
H1=0.5*sinc(0.5*[-10:10]);
H1=H1.*hamming(21)';
G1=abs(fft(H1,210));figure
subplot(221),plot(W,G1(1:105)),
axis tight,title('WINDOWING')
subplot(222),stem(H1),
axis tight,title('WINDOWING')
F=[1 1 1 1 1 .5 zeros(1,10) .5 1 1 1 1];
H2=fftshift(real(ifft(F)));
G2=abs(fft(H2,210));figure
subplot(221),plot(W,G1(1:105)),
axis tight,title('SAMPLING')
subplot(222),stem(H2),
axis tight,title('SAMPLING')
H3=firpm(20,[0 0.4 0.6 1.0],[1 1 0 0]);
G3=abs(fft(H3,210));figure
subplot(221),plot(W,G3(1:105)),
axis tight,title('EQUIRIPPLE')
subplot(222),stem(H3),
axis tight,title('EQUIRIPPLE')
[B A]=butter(10,0.5);figure
G4=abs(fft(B,210))./abs(fft(A,210));
subplot(221),plot(W,G4(1:105)),
axis tight,title('BUTTERWORTH')
subplot(222),zplane(B,A)
title('BUTTERWORTH')
```

gives the results shown below. Impulse response is the left half and frequency response is the right half. They are all quite similar. The equiripple design has a sharper transition, but also has ripples in the pass-band and stopband.

Matlab also has a nice GUI for filter design called `fdatool`. It is pretty much self-explanatory.

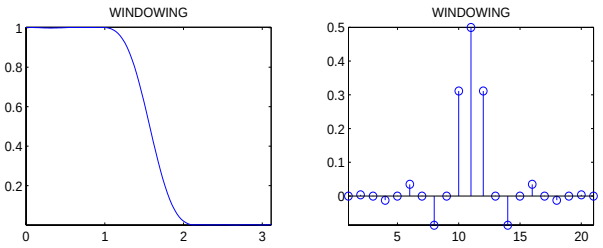


Figure 9.1: Hamming window design.

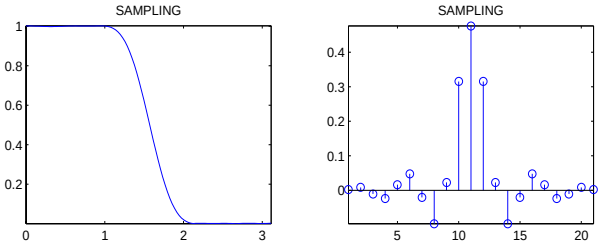


Figure 9.2: Frequency sampling design.

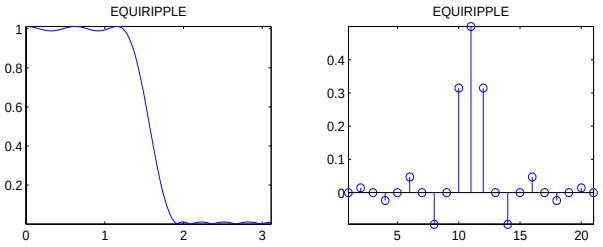


Figure 9.3: Equiripple design.

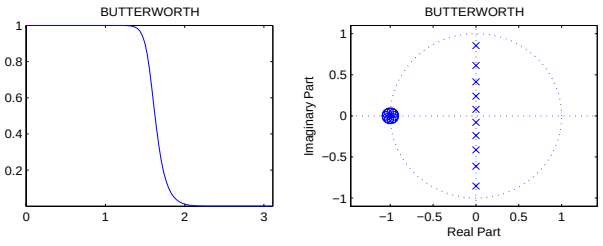


Figure 9.4: Butterworth filter design.

Chapter 10

Multirate Signal Processing

The basic idea behind multirate signal processing is changing the sampling rate after the fact. If a signal was originally sampled at 1000 samples per sec. and we would like to change that to 8000 samples per sec., we can do this by *upsampling*, followed by (discrete-time) *interpolation*. If we want to change it by a rational but non-integer factor, we use a combination of *upsampling and interpolation* and *downsampling*.

Three reasons for changing the sampling rate are:

- To replace the final discrete-time-to-continuous-time interpolating filter with a simpler one, or with sample-and-hold (analog) interpolation;
- To create the sounds of a musical instrument playing all musical notes from the sound of it playing a single musical note;
- To design very selective (in frequency) discrete-time filters that have very sharp transitions between their pass-band and their stop-band. We can do this using a series of less-selective filters and multirate signal processing.

There are four parts to multirate signal processing:

- Downsampling (decimating)
- Upsampling (zero-stuffing)
- Interpolation (low-pass filtering)
- Multirate processing (combining these)

We analyze each of these in the next four sections.

10.1 Downsampling

10.1.1 Downsampling: Time Domain

Downsampling, also called decimation, reduces the sampling rate by an integer factor. This is very easy to do. To change the sampling rate of a signal from 1200 samples per sec. to 600 samples per sec., we simply omit every other sample. To change the rate from 1200 to 400 samples per sec., we simply omit two out of three samples, keeping only every third.

Example: Downsampling.
Downsample by a factor of 2 the signal
$x[n]=\{\ldots \underline{3}, 1, 4, 1, 5, 9, 2, 6, 5 \ldots\}$

Solution:
Omitting every other sample leaves
$y[n]=\{\ldots \underline{3}, 4, 5, 2, 5 \ldots\}$

This is what we would have gotten had we sampled the original signal at half of the sampling rate. So we have indeed halved the sampling rate.

The notation for downsampling by two is

$x[n] \rightarrow \boxed{\downarrow 2} \rightarrow y[n]=x[2n]$ for all integers n:

$y[0]=x[0]$, $y[1]=x[2]$, $y[2]=x[4]$, $y[3]=x[6]$,etc.
Time has been compressed by a factor of two.

Downsampling Matlab code: `Y=X[1:2:end];`

The generalization to downsampling by a factor of integer L is evident: Replace 2 with L above.

10.1.2 Downsampling: Spectrum

The effect of downsampling on a sinusoid is

$$\cos(\omega_o n) \rightarrow \boxed{\downarrow \mathrm{L}} \rightarrow \cos(\omega_o(Ln)) = \cos((\omega_o L)n). \tag{10.1}$$

Downsampling $\rightarrow \boxed{\downarrow \mathrm{L}} \rightarrow$ increases frequency by L

To see what this does to spectra, consider L=2. Omitting every other sample is performed using

$$\frac{1}{2}\left[1+(-1)^n\right] = \begin{cases} 1 & \text{for } n \text{ even} \\ 0 & \text{for } n \text{ odd} \end{cases} \tag{10.2}$$

Using the modulation property of the DTFT and changing variables from n to n'=2n, the DTFT of $y[n]$=$x[2n]$ can be computed as (recall –1=$e^{j\pi}$)

$$\begin{aligned} Y(e^{j\omega}) &= \sum x[2n]e^{-j\omega(2n)/2} \\ &= \sum x[n'](1/2)[1+(-1)^{n'}]e^{-j\omega n'/2} \\ &= [X(e^{j\omega/2}) + X(e^{j(\omega+2\pi)/2})]/2 \end{aligned} \tag{10.3}$$

- $y[n]$=$x[2n]$ compresses time by a factor of 2.
- $Y(e^{j\omega})$=$X(e^{j\omega/2})$ expands frequency by 2. In particular, frequency of sinusoids doubles. This doubles the period of $X(e^{j\omega/2})$ from 2π to 4π.
- But since $Y(e^{j\omega})$ *must* be periodic with period 2π, we must insert *additional* copies of $X(e^{j\omega/2})$ to make $Y(e^{j\omega})$ periodic with period 2π.

Now we treat the general case of integer L. Replacing (10.2) with the more general relation

$$\frac{1}{L}\sum_{k=0}^{L-1} e^{j2\pi nk/L} = \begin{cases} 1 & \text{for } n \text{ a multiple of } L \\ 0 & \text{otherwise} \end{cases} \tag{10.4}$$

and using a similar derivation gives

$$Y(e^{j\omega}) = \frac{1}{L}\sum_{k=0}^{L-1} X(e^{j(\omega+2\pi k)/L}). \tag{10.5}$$

as the effect on spectrum of downsampling by L.

Example: Downsampling: spectrum effect.

A signal has spectrum shown in the next plot. For convenience, π is approximated by 3. It is downsampled by 4. Sketch the output spectrum.

Solution:

The spectrum is stretched by a factor of 4. Three copies are inserted to keep the period 2π. The height is reduced by a factor of 4 (not shown).

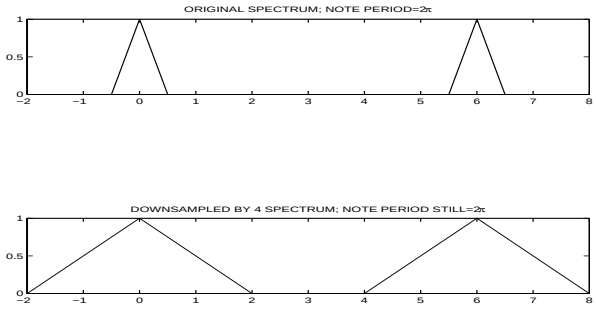


Figure 10.1: Effect of downsampling on spectrum.

Downsampling $\rightarrow \boxed{\downarrow \mathrm{L}} \rightarrow$ stretches spectra by L. Extra copies are inserted to maintain period 2π.

10.2 Upsampling

10.2.1 Upsampling: Time Domain

Upsampling, also called zero stuffing, followed by interpolation, increases the sampling rate by an integer factor. We first analyze upsampling, to show the need and effect for interpolation. "Upsampling" is also used to denote the combination of zero stuffing and interpolation. This describes the effect, but destroys the notational symmetry with downsampling. So we use upsampling to mean only zero-stuffing.

To upsample by L, simply insert L–1 zeros between each sample of a signal.

Example: Upsampling.
Upsample by a factor of 2 the signal
$x[n]$=$\{\ldots \underline{3}, 1, 4, 1, 5, 9 \ldots\}$

Solution:
Inserting a zero between samples gives
$y[n]$=$\{\ldots \underline{3}, 0, 1, 0, 4, 0, 1, 0, 5, 0, 9 \ldots\}$

The notation for upsampling by two is

$x[n] \rightarrow \boxed{\uparrow 2} \rightarrow y[n] = \begin{cases} x[n/2] & \text{for even } n \\ 0 & \text{for odd } n \end{cases}$

$y[0]=x[0]$, $y[2]=x[1]$, $y[4]=x[2]$, $y[6]=x[3]$,etc.

Matlab: `Z=[X;zeros(1,length(X))];Y=Z(:)';`

10.2.2 Upsampling: Spectrum

The effect of upsampling on spectra can be analyzed using (10.2). Recalling that upsampling is

- $x[n] \rightarrow \boxed{\uparrow 2} \rightarrow y[n] = \begin{cases} x[n/2] & \text{for even } n \\ 0 & \text{for odd } n \end{cases}$

and changing variables from n to $n'=n/2$ gives

$$\begin{aligned} Y(e^{j\omega}) &= \sum_{\text{even } n} x[n/2]e^{-j\omega 2(n/2)} \\ &= \sum x[n']e^{-j\omega 2n'} \\ &= X(e^{j2\omega}) \end{aligned} \tag{10.6}$$

Upsampling expands time by inserting zeros.
Upsampling by L compresses the spectrum by L.
The spectrum is now periodic with period $\frac{2\pi}{L}$.
It still repeats every 2π, but also more often.

Example: Upsampling effect on spectrum.
A signal has spectrum shown in the next plot. For convenience, π is approximated by 3. It is upsampled by 4. Sketch the output spectrum.

Solution:

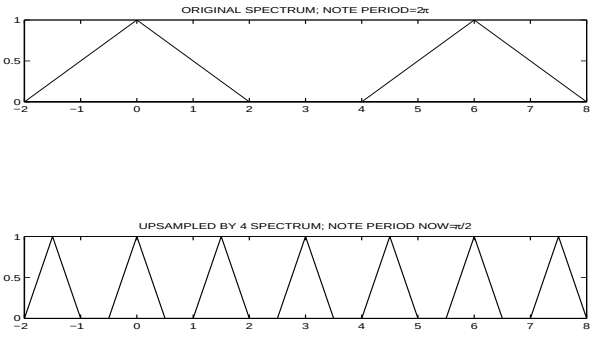


Figure 10.2: Effect of upsampling on spectrum.

Upsampling $\rightarrow \boxed{\uparrow \text{L}} \rightarrow$ compresses spectrum by L

10.3 Interpolation

10.3.1 Interpolating a Signal

Interpolating a signal $x(t)$ means assigning values to $x(t)$ at times t for which the signal is not already specified. If $x(t)$ is specified for $t=nT$, so that $\{x(nT)\}$ is known, interpolating $x(t)$ means assigning values to $\{x(t), t \neq nT\}$, based on the given values $\{x(nT)\}$. For example, linear interpolation means $x(t)$ varies linearly between $x(nT)$ and $x((n+1)T)$. Interpolation "connects the dots" between given $\{x(nT)\}$.

Interpolation can also occur entirely in discrete time. Let $x[n]$ be a signal that has just been upsampled by L, so that $x[n]$ has the form

$$x[n] = \{\ldots x[0], \underbrace{0\ldots 0}_{L-1}, x[L], \underbrace{0\ldots 0}_{L-1}, x[2L], \ldots\} \tag{10.7}$$

We interpolate $x[n]$ by changing its zero values to the nonzero values that make $x[n]$ have maximum frequency $\frac{\pi}{L}$. The goal of this subsection is to show we can do this by filtering $x[n]$ with the lowpass filter

$$H(e^{j\omega}) = \begin{cases} L & \text{for } 0 \leq |\omega| < \frac{\pi}{L} \\ 0 & \text{for } \frac{\pi}{L} < |\omega| < \pi \end{cases} \tag{10.8}$$

which has the impulse response

$$h[n] = L\frac{\sin((\pi/L)n)}{\pi n}. \tag{10.9}$$

Let the interpolated $x[n]$ be $y[n]$. Then

$$y[n] = \sum x[i]h[n-i] = \sum x[i]L\frac{\sin((\pi/L)(n-i))}{\pi(n-i)}. \tag{10.10}$$

Since $x[i]=0$ unless i is a multiple of L, *let* $i=i'L$. Since the goal is to show $y[n]=x[n]$ for n a multiple of L, *set* $n=n'L$. Then

$$\begin{aligned} y[n'L] &= \sum x[i'L]L\frac{\sin((\pi/L)(n'L-i'L))}{\pi(n'L-i'L)} \\ &= \sum x[i'L]\frac{\sin(\pi(n'-i'))}{\pi(n'-i')} \\ &= \sum x[i'L]\delta[n'-i'] = x[n'L] \end{aligned} \tag{10.11}$$

so the given values $x[n'L]$ of $x[n]$ are preserved after filtering with $h[n]$, which thus interpolates $\{x[n'L]\}$.

10.3.2 Upsampling and Interpolation

Upsampling by L alone does no just reduce frequencies of sinusoids by L–it also introduces additional sinusoids. These can be removed by a lowpass filter. The combined operation of *upsample and interpolate* reduces frequencies without adding new ones, since additional frequencies are filtered by interpolation.

- $x[n] \rightarrow \boxed{\uparrow L} \rightarrow \boxed{h[n]} \rightarrow y[n]$.
- $h[n]$ is lowpass filter with cutoff $\omega_o = \frac{\pi}{L}$. Design using a technique from the previous chapter.
- `N=length(X);Z=[X;zeros(L-1,N)];Y=Z(:)';F=fft(Y);F(N/2:N*L+2-N/2)=0;Y=ifft(F);` since `Y` has length `N*L` after the upsampling.

Example: Upsampling and interpolation.
A signal has spectrum shown in the next plot. For convenience, π is approximated by 3 again. The signal is upsampled by 4 and interpolated. Sketch the output spectrum after interpolation.

Solution:
The spectrum is compressed by a factor of 4. The period was $\frac{2\pi}{4} = \frac{\pi}{2}$. It still repeated every 2π. But after filtering, the period is now back to 2π. 3 out of 4 copies are now eliminated by filtering. The signal has all its frequencies reduced by 4.

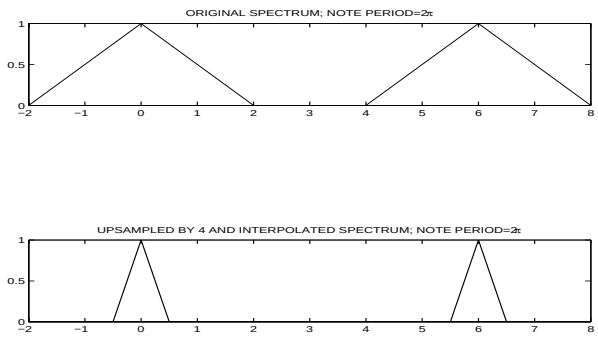


Figure 10.3: Upsampling and interpolation.

10.4 Multirate

The sampling rate can be multiplied by a rational number $\frac{M}{N}$ using the following procedure:

- Upsampling by M; interpolating; then
- Downsampling by N, in that order:
- $x[n] \rightarrow \boxed{\uparrow M} \rightarrow \boxed{h[n]} \rightarrow \boxed{\downarrow N} \rightarrow y[n]$
- `L=length(X);Z=[X;zeros(M-1,L)];Y=Z(:)';F=fft(Y);F(L/2:L*M+2-L/2)=0;Y=ifft(F);Y=Y(1:N:end);` The middle length is `L*M`.
- This multiplies sinusoidal frequencies by $\frac{N}{M}$.

It is important to perform the operations in this specific order, since downsampling first may lead to aliasing at the middle step. Upsampling cannot lead to aliasing, but upsampling after downsampling may be too late to avoid aliasing. Upsampling first cannot lead to aliasing. Of course, the final sampling rate must be fast enough to avoid aliasing.

Example: Down- and up-sampling.
What are the outputs of these systems?

1. $x[n] \rightarrow \boxed{\uparrow 4} \rightarrow \boxed{\downarrow 4} \rightarrow ?$
2. $x[n] \rightarrow \boxed{\downarrow 4} \rightarrow \boxed{\uparrow 4} \rightarrow ?$

Solution:

1. $x[n] \rightarrow \boxed{\uparrow 4} \rightarrow \boxed{\downarrow 4} \rightarrow x[n]$. Upsampling inserts zeros and then downsampling removes them.
2. $x[n] \rightarrow \boxed{\downarrow 2} \rightarrow \boxed{\uparrow 2} \rightarrow x[n]$ if n even; 0 if n odd. Replaces half of the $x[n]$ values with zeros.

The following examples illustrate this. We label spectral lines with their continuous-time frequencies f in Hertz, rather than with discrete-time frequency $\omega = 2\pi\frac{f}{S}$, where S is the sampling rate, for clarity.

Example: Multirate filtering.
Each of the following uses a sampling rate of 2400 samples per sec. For each configuration, the input

is a single sinusoid at the frequency shown. Determine the frequencies of the output sinusoids, in Hertz. Analog sampling and reconstruction are omitted.

1. 600 Hertz→$\boxed{\uparrow 3}$→?
2. 500 Hertz→$\boxed{\downarrow 3}$→$\boxed{\uparrow 2}$→?
3. 500 Hertz→$\boxed{\uparrow 2}$→$\boxed{h[n]}$→$\boxed{\downarrow 3}$→?
   Note the order changed in the latter two.

Solution:
Discrete-time spectra periodic with period 2π.
Continuous-time spectra periodic at 2400 Hertz.
Component at $f_o \rightarrow \pm\{f_o, 2400 \pm f_o, 4800 \pm f_o\}$.

1. 600 Hertz→$\boxed{\uparrow 3}$→$\boxed{\{200, 600, 1000\}}$
   Input after sampling: $\pm\{600, 2400 \pm 600, \ldots\}$
   Upsampling divides by 3: $\pm\{200, 600, 1000, \ldots\}$
   Others are above the Nyquist rate of 1200 Hertz.
   Note a lowpass filter would remove two of these, leaving only $\frac{1}{3}(600)$=200 Hertz.
2. 500→$\boxed{\downarrow 3}$→$\boxed{\uparrow 2}$→$\boxed{\{450, 750\}}$
   Input after sampling: $\pm\{500, 2400 \pm 500, \ldots\}$
   Downsampling mults by 3: $\pm\{1500, 5700, \ldots\}$
   Aliases down to: $\pm\{1500$–$2400, 5700$–$4800 \ldots\}$
   $=\pm\{900, 2400 \pm 900\}$. Rest above Nyquist rate.
   Upsampling divides by 2: $\pm\{450, 750, \ldots\}$
   We wanted $\frac{3}{2}(500)$=750! How can we do this?
3. 500 Hertz→$\boxed{\uparrow 2}$→$\boxed{h[n]}$→$\boxed{\downarrow 3}$→$\boxed{750}$
   Input after sampling: $\pm\{500, 2400 \pm 500, \ldots\}$
   Upsampling divides by 2: $\pm\{250, 950, \ldots\}$
   Lowpass filtering removes 950, leaving $\pm\{250\}$.
   Downsampling multiplies by 3: $\pm\{750\}$ only.

10.5 APPLICATION: Discrete to Continuous Time

10.5.1 Motivation for Oversampling

Ultimately, discrete-time signals must be converted to continuous-time signals, which can be physically displayed or heard. This requires: (1) multiplication by a pulse train whose period is the same as the sampling interval $\frac{1}{S}$ (where S is the sampling rate); followed by (2) an *analog* lowpass filter made of capacitors, such as a Butterworth filter. However, the gain $|H(j2\pi f)|$ of this filter drops off only as $1/f^L$ where L capacitors are used (this is $6L$ dB/octave=$20L$ dB/decade). Define the following:

- S=sampling rate in samples/sec.;
- L=#of capacitors in the filter;
- B=maximum frequency of signal;
- $2B$=Nyquist (minimum) sampling rate.

Sampling creates copies of the original signal spectrum repeating every S Hertz. The largest copy occupies S–$B < f < S$+B. The Butterworth filter must eliminate this copy. If we define "eliminate" as reduction by a factor of 1000, then we require

$$\left[\frac{S-B}{B}\right]^L = 1000; L = 3/\log_{10}\left[\frac{S-B}{B}\right] \quad (10.12)$$

neglecting the gain reduction at corner frequencies. For example, a speech signal with a maximum frequency of 4000 Hertz is sampled at the CD sampling rate of 44100 samples per sec. This requires $3/[\log_{10}\frac{44100-4000}{4000}]$=3 capacitors in the filter.

We could reduce L by increasing the sampling rate S, but this requires much more storage. However, there is another way: we can increase the sampling rate by upsampling and interpolating! We are, in effect, replacing a sharp analog filter, with many capacitors, with a sharp discrete-time filter, which can be implemented entirely in software and designed using the methods in the last chapter. This is called *oversampling*; "16x oversampling" means the implemented sampling rate is 16 times the actual sampling rate. Making the implemented sampling rate so high means only a crude analog lowpass filter is needed; the *mechanical* lowpass filter inherent in the frequency response of an earbud may be enough.

10.5.2 Implementing Oversampling

Let $x[n]$ be samples of a signal $x(t)$ with maximum frequency B Hertz sampled at S=$4B$ samples per sec.

This is double the (minimum sampling) Nyquist rate of $S{=}2B$. A sharp analog filter, with about seven capacitors, would be necessary to recover $x(t)$ from its samples $x[n]$. Specifically, the gain must satisfy

$$|H(j2\pi f)| = \begin{cases} 1 & \text{for } |f| < B \\ < 0.001 & \text{for } |f| > 3B \end{cases} \tag{10.13}$$

Instead, we can implement the oversampling system:

$$x[n] \rightarrow \boxed{\uparrow 10} \rightarrow \boxed{h[n]} \rightarrow \boxed{\text{ANALOG}} \rightarrow x(t) \tag{10.14}$$

where discrete-time filter $h[n]$ satisfies ($S{=}4B$)

$$|H(e^{j\omega})| = \begin{cases} 1 & \text{for } |\omega| < 2\pi\frac{B}{10S}{=}\frac{\pi}{20} \\ < 0.001 & \text{for } |\omega| > 2\pi\frac{3B}{10S}{=}\frac{3\pi}{20} \end{cases} \tag{10.15}$$

and the analog LPF satisfies (note $10S$–$B{=}39B$)

$$|H(j2\pi f)| = \begin{cases} 1 & \text{for } |f| < B \\ < 0.001 & \text{for } |f| > 39B \end{cases} \tag{10.16}$$

which requires only two capacitors.

Also note that since the upsampled $x[n]$ is 90% zeros, the computation of filtering with $h[n]$ is reduced by 90% to 10% of its usual amount.

10.6 APPLICATION: Audio Signal Processing

Suppose we are given a snippet of a trumpet playing a single note. The goal is to generate snippets of the trumpet playing all of the other musical notes from the single note. We can do this using multirate filtering and the *Circle of Fifths* from music theory.

10.6.1 Music Notation & Frequencies

Most Western music is based on the 12-tone or chromatic musical scale. This scale consists of 7 whole notes, designated by letters A,B,C,D,E,F,G and 5 accidental notes, designated by A#,C#,D#,F#,G# and pronounced "A-sharp," etc. The frequencies of these 12 notes are in geometric progression, with a common ratio of $2^{\frac{1}{12}}$, so they span a factor of two. This is called an octave. The reason for using frequencies in geometric progression is to preserve the ratio between frequencies of notes after a key change.

The frequency of A# is between the frequencies of notes A and B. A# is also designated Bb, pronounced "B-flat," and similarly for the other accidental notes. On a piano keyboard, whole notes are played on the white keys, and accidental notes are played on the black keys. This can all be summarized as follows:

Note	A	A#	B	C	C#	D
$\frac{\text{FREQ}}{440\,\text{Hz}}$	$2^{\frac{0}{12}}$	$2^{\frac{1}{12}}$	$2^{\frac{2}{12}}$	$2^{\frac{3}{12}}$	$2^{\frac{4}{12}}$	$2^{\frac{5}{12}}$
Hertz	440	466	494	523	554	587

Note	D#	E	F	F#	G	G#
$\frac{\text{FREQ}}{440\,\text{Hz}}$	$2^{\frac{6}{12}}$	$2^{\frac{7}{12}}$	$2^{\frac{8}{12}}$	$2^{\frac{9}{12}}$	$2^{\frac{10}{12}}$	$2^{\frac{11}{12}}$
Hertz	622	659	698	740	784	830

10.6.2 Musical Circle of Fifths

Note A is 440 Hertz and E is 659 Hertz, almost exactly a 3:2 ratio. The 3:2 ratio holds between different pairs of notes called "fifths." Indeed, Western music was first based on using these ratios, rather than the equally-spaced logarithms of frequencies used today. The change occurred when J.S. Bach composed "The Well-Tempered Clavier" using the 12-tone scale. It sounded better than using the frequencies based on 3:2 ratios ("The Ill-Tempered Clavier"?).

The relevance of this to the present problem is apparent from the following table:

Freq. 440 Hz	$\frac{(\frac{3}{2})^0}{2^0}$	$\frac{(\frac{3}{2})^1}{2^0}$	$\frac{(\frac{3}{2})^2}{2^1}$	$\frac{(\frac{3}{2})^3}{2^1}$	$\frac{(\frac{3}{2})^4}{2^2}$	$\frac{(\frac{3}{2})^5}{2^2}$
Hertz	440	660	495	742	557	835
Note	A	E	B	F#	C#	G#

$\frac{(\frac{3}{2})^6}{2^3}$	$\frac{(\frac{3}{2})^7}{2^4}$	$\frac{(\frac{3}{2})^8}{2^4}$	$\frac{(\frac{3}{2})^9}{2^5}$	$\frac{(\frac{3}{2})^{10}}{2^5}$	$\frac{(\frac{3}{2})^{11}}{2^6}$	$\frac{(\frac{3}{2})^{12}}{2^7}$
626	470	705	529	793	595	446
D#	A#	F	C	G	D	A

By repeatedly upsampling by 2 and downsampling by 3, we can start with the snippet of a trumpet playing note A and multiply that frequency (440 Hertz) by powers of $\frac{3}{2}$. This sweeps out all of the 12 notes, in the order shown in the above table. Occasionally an extra upsampling by 2 is necessary to keep the

frequencies within this octave. We can then obtain the 12 notes in all other octaves by upsampling and downsampling by 2 to synthesize the same note in a different octave.

This is called the Circle of Fifths because arranging the 12 semitones in a circle, like hours on a clock face, and taking repeated jumps of 7 "hours" each, we can attain all 12 "hours." A Fifth is the interval between 5 whole tones (such as A to E above) or 7 notes. This will sweep out all of the notes in the octave because: (1) 7 and 12 are relatively prime; and (2) $\frac{3}{2} \approx 2^{7/12}$.

10.6.3 Application of Multirate

Ironically, this would be easier to implement in continuous time! A variable speed tape recorder or record turntable could implement multirate processing by speeding up or slowing down the tape or turntable, using a variable-speed motor. "Alvin and the Chipmunks" were recorded this way in the 1950's by speeding up the playback by a factor of two on a variable-speed tape recorder. It was then necessary to (literally!) cut and paste snippets of tape!

We can do this more easily by repeatedly using

- $x[n] \rightarrow \boxed{\uparrow 2} \rightarrow \boxed{h[n]} \rightarrow \boxed{\downarrow 3} \rightarrow y[n]$
- $H(e^{j\omega})$=3 $\begin{cases} 1 & \text{for } 0 < |\omega| < \frac{\pi}{2} \\ 0 & \text{for } \frac{\pi}{2} < |\omega| < \pi \end{cases}$
- If the fundamental frequency>784 Hertz then upsample by 2 to reduce the frequency by half.

10.7 Filter Design

A selective filter has a sharp transition between its pass-band and its stop-band. We have seen in continuous time that a selective Butterworth filter requires many capacitors. In discrete time, there are no such physical requirements, but the order (length) of the filter must be large. We have seen this for notch filters, for which a more selective frequency response requires poles nearer the unit circle, and thus longer impulse and transient responses.

We can use multirate to implement a very selective filter as a cascade of less selective filters, with downsampling and upsampling. The idea is as follows:

- Use downsampling to stretch the frequency axis;
- Design the filter on the stretched axis;
- Use upsampling to compress the frequency axis.

The transition band is wider on the stretched axis.

We illustrate this idea with an example.

Example: Filter design using multirate.
Implement the filter with specs

$$H_D(e^{j\omega}) = \begin{cases} 1 & \text{for } |\omega| < 0.0095\pi \\ 0 & \text{for } |\omega| > 0.01\pi \end{cases} \tag{10.17}$$

using the three filters with specs

$$H_1 = \begin{cases} 1 & \text{for } |\omega| < 0.0095\pi \\ 0 & \text{for } |\omega| > 0.1\pi \end{cases} \tag{10.18}$$

$$H_2 = \begin{cases} 1 & \text{for } |\omega| < 0.095\pi \\ 0 & \text{for } |\omega| > 0.1\pi \end{cases} \tag{10.19}$$

$$H_3 = \begin{cases} 1 & \text{for } |\omega| < 0.01\pi \\ 0 & \text{for } |\omega| > 0.1\pi \end{cases} \tag{10.20}$$

Note the transition widths:

- $H_D(e^{j\omega})$: 0.0005π Very sharp!
  This is the desired frequency response.
- $H_1(e^{j\omega})$: 0.0905π Very dull!
  This is an anti-alias filter before downsampling.
- $H_2(e^{j\omega})$: 0.0050π Rather sharp.
  $H_2(e^{j\omega})$ is $H_D(e^{j\omega})$ stretched by 10.
- $H_3(e^{j\omega})$: 0.0900π Very dull!
  This is the interpolating filter after upsampling.

We use the following system:

$$x[n] \rightarrow \boxed{H_1} \rightarrow \boxed{\downarrow 10} \rightarrow \boxed{H_2} \rightarrow \boxed{\uparrow 10} \rightarrow \boxed{H_3} \rightarrow y[n] \tag{10.21}$$

First, we design the rather sharp filter $H_2(e^{j\omega})$. Then upsampling compresses the frequency axis, so both the pass-band-to-transition-band border 0.095π and the stop-band-to-transition-band border 0.1π are divided by 10, which are the specs for $H_2(e^{j\omega})$. We implemented $H_2(e^{j\omega})$ with less-sharp filters!

Chapter 11

Correlation

Correlation is a general term that describes three different functions of two signals $x[n]$ and $y[n]$. These functions, their applications, and their definitions:

- *Auto*correlation, for the period of a signal. Autocorrelation $r_x[n] = x[n] * x[-n]$;
- *Cross*-correlation, for the delay of a signal. Cross-correlation $r_{xy}[n] = x[n] * y[-n]$;
- *Correlation*, for presence or absence of signals. Correlation $r_{xy}[0] = \sum_{i=0}^{N-1} x[i]y[i]$, where N is the durations of $x[n]$ and $y[n]$.

The cross-correlation of two signals is another signal, while the corrrelation of two signals is a number. Correlation can also be used to determine which one of several possible signals is present (see below).

We discuss each of these three functions in more detail. We then apply them to real-world signals.

11.1 Autocorrelation

The *autocorrelation* $r_x[n]$ of $x[n]$ is

$$r_x[n] = x[n] * x[-n] = \sum_{i=-\infty}^{\infty} x[i]x[i+n] \qquad (11.1)$$

The *lag* n has units of time but it is *not* time. Lag is the delay of one signal relative to itself.

Writing out the definition of $r_x[n]$ gives

- $r_x[0]$=$\ldots x[-1]^2$+$x[0]^2$+$x[1]^2$+$\ldots$
- $r_x[1]$=$\ldots x[-1]x[0]$+$x[0]x[1]$+$x[1]x[2]$+$\ldots$
- $r_x[2]$=$\ldots x[-1]x[1]$+$x[0]x[2]$+$x[1]x[3]$+$\ldots$

Autocorrelation has these properties:

- $r_x[n]$ is an even function: $r_x[n]$=$r_x[-n]$;
- $r_x[0]$=$\sum_{i=-\infty}^{\infty} x[i]^2 \geq 0$=energy of $x[n]$;
- $r_x[0] \geq r_x[n]$ for any *lag* n. Usually $r_x[0] >> r_x[n]$ unless $x[n]$ is periodic.
- The DTFT $R_x(e^{j\omega})$ of $r_x[n]$ is related to $X(e^{j\omega})$ by $R_x(e^{j\omega})$=$X(e^{j\omega})X(e^{-j\omega})$=$|X(e^{j\omega})|^2$ using the time-reversal and conjugate-symmetry properties of the DTFT. Also note that $|X(e^{j\omega})|^2$ real and even implies $r_x[n]$ is even.

If $x[n]$ has finite duration N, then

- $r_x[n]$=$x[n] * x[-n]$ has duration $2N$–1.
- $r_x[n]$=0 except for –(N–1) $\leq n \leq$ N–1.
- Delaying $x[n]$ by D does not affect $r_x[n]$: $R_x(e^{j\omega})$=$|X(e^{j\omega})e^{-j\omega D}|^2$=$|X(e^{j\omega})|^2$.
- $r_x[n]$ can be computed rapidly from $x[n]$ using `N=length(X);M=nextpow2(2*N-1);` `Rx=real(ifft(abs(fft(X,M)).^2));` `Rx=fftshift(Rx)` puts $r_x(0)$ in the middle.

Example: Computing autocorrelation.
$x[n] = \{3, 1, 4\}$. Compute autocorrelation $r_x[n]$. Note the location of n=0 in $x[n]$ is irrelevant!

Solution:

- $r_x[0]$=3^2+1^2+4^2=26.
- $r_x[1]$=(3)(1)+(1)(4)=7.
- $r_x[2]$=(3)(4)=12.
- $r_x[n]$=$\{12, 7, \underline{26}, 7, 12\}$.

$r_x[n]$ is even and $r_x[0] \geq r_x[n]$, as expected.

11.2 Cross-correlation

The *cross-correlation* $r_{xy}[n]$ of $x[n]$ and $y[n]$ is

$$r_{xy}[n] = x[n] * y[-n] = \sum_{i=-\infty}^{\infty} x[i]y[i-n].$$

$$r_{yx}[n] = y[n] * x[-n] = \sum_{i=-\infty}^{\infty} x[i]y[i+n]. \quad (11.2)$$

The *lag* n has units of time but it is *not* time.
Lag is the delay of one signal relative to the other.

Writing out the definition of $r_{xy}[n]$ gives

- $r_{xy}[0]$=$\ldots x[-1]y[-1]$+$x[0]y[0]$+$x[1]y[1]$+$\ldots$
- $r_{xy}[1]$=$\ldots x[0]y[-1]$+$x[1]y[0]$+$x[2]y[1]$+$\ldots$
- $r_{xy}[2]$=$\ldots x[0]y[-2]$+$x[1]y[-1]$+$x[2]y[0]$+$\ldots$

Cross-correlation has these properties:

- $r_{yx}[n] = r_{xy}[-n]$ is *not* an even function.
- DTFT $R_{xy}(e^{j\omega})$ of $r_{xy}[n]$ is related to $X(e^{j\omega})$ and $Y(e^{j\omega})$ by $R_{xy}(e^{j\omega})$=$X(e^{j\omega})Y(e^{j\omega})^*$ using time-reversal and conjugate-symmetry.
- $r_{xy}[n]$ can be computed from $x[n]$ and $y[n]$ using `L=length([X Y]);M=nextpow2(L-1);` `Rxy=real(ifft(fft(X,M).*conj(fft(Y,M))));` `Rxy=fftshift(Rxy)` puts $r_{xy}(0)$ in the middle.

Example: Computing cross-correlation.
$x[n]$=$\{3, 1, 4\}$ and $y[n]$=$\{2, 7, 1\}$. Compute $r_{xy}[n]$.

Solution:

- $r_{xy}[-2]$=(3)(1)=3.
- $r_{xy}[-1]$=(3)(7)+(1)(1)=22.
- $r_{xy}[0]$=(3)(2)+(1)(7)+(4)(1)=17.
- $r_{xy}[1]$=(1)(2)+(4)(7)=30.
- $r_{xy}[2]$=(4)(2)=8.
- $r_{xy}[n]$=$\{3, 22, \underline{17}, 30, 8\}$.

We can also use $\{3, 1, 4\} * \{1, 7, 2\}$.

11.3 Correlation

The *correlation* between signals $x[n]$ and $y[n]$ of durations N each (zero-pad if needed) is the *number*

$$r_{xy}[0] = \sum_{i=0}^{N-1} x[i]y[i]. \quad (11.3)$$

This is the inner product of the vectors of the elements of $\{x[n]\}$ and $\{y[n]\}$. In Matlab: `X'*Y`. Recalling that the inner product of vectors $\vec{x}$ and $\vec{y}$ is

$$\vec{x}^T\vec{y} = ||\vec{x}|| \cdot ||\vec{y}|| \cos\theta \quad (11.4)$$

where θ is the angle between $\vec{x}$ and $\vec{y}$ leads to the concept of signal similarity, which is discussed next.

11.3.1 Signal Similarity

The significance of correlation is that it is a measure of *signal similarity* (how much two signals are alike).

The *Cauchy-Schwarz inequality* states that

$$|r_{xy}[0]| \leq \sqrt{r_x[0]r_y[0]}. \quad (11.5)$$

The Cauchy-Schwarz inequality can be derived quickly by noting that

$$\begin{aligned} 0 &\leq \sum_{n=0}^{N-1} \left[\frac{x[n]}{\sqrt{r_x[0]}} \pm \frac{y[n]}{\sqrt{r_y[0]}} \right]^2 \\ &= \sum_{n=0}^{N-1} \frac{x[n]^2}{r_x[0]} + \sum_{n=0}^{N-1} \frac{y[n]^2}{r_y[0]} \pm 2 \sum_{n=0}^{N-1} \frac{x[n]y[n]}{\sqrt{r_x[0]r_y[0]}} \\ &= 1 + 1 \pm 2\frac{r_{xy}[0]}{\sqrt{r_x[0]r_y[0]}} \quad (11.6) \end{aligned}$$

Multiplying by $\sqrt{r_x[0]r_y[0]}$ and choosing the sign so $\pm r_{xy}[0] \leq 0$ yields the Cauchy-Schwarz inequality.

We define the "angle" θ between $x[n]$ and $y[n]$ by

$$\cos\theta = r_{xy}[0]/\sqrt{r_x[0]r_y[0]} = r_{xy}[0]/\sqrt{E_x E_y} \quad (11.7)$$

since $r_x[0]$ is the energy E_x of $x[n]$.

$\cos\theta$ is called the *correlation coefficient* of $x[n]$ and $y[n]$; it is the energy-normalized version of correlation. In statistics, the correlation coefficient is defined by first subtracting off the means of $x[n]$ and $y[n]$ and then applying (11.7) to the results, but that is not necessary here. The closer θ is to zero, the more similar are $x[n]$ and $y[n]$, excluding a scale factor.

Example: Signal Similarity
Let the signals $x[n]$ and $y[n]$ be

$$\begin{aligned} x[n] &= A_1\cos(2\pi(M/N)n + \theta_1) \\ y[n] &= A_2\cos(2\pi(M/N)n + \theta_2). \end{aligned} \quad (11.8)$$

Some straightforward algebra shows that

$$\cos\theta = r_{xy}[0]/\sqrt{r_x[0]r_y[0]} = \cos(\theta_1 - \theta_2).$$

θ is literally the phase angle between $x[n]$ and $y[n]$!

So $A_1\cos(2\pi(M/N)n)$ and $A_2\sin(2\pi(M/N)n)$ are both *orthogonal* since θ=90^o. Note that using frequency $2\pi(M/N)$ ensures an integer number M of periods in the interval $0 \leq n \leq N$–1.

11.3.2 White Gaussian Noise

Zero-mean white Gaussian noise is a commonly-used model. Its pertinent properties here are as follows.

Let $w[n]$ be a zero-mean white Gaussian noise random process with strength σ^2. Then we have:

- $w[i]$ is a zero-mean Gaussian random variable: $\Pr[-\sigma \leq w[n] \leq \sigma] \approx \frac{2}{3}$ since $w[n]$ is Gaussian.
- $w[i]$ and $w[j]$ are independent for all $i \neq j$: Knowing the value of $w[i]$ tells nothing about the value of $w[j]$, even for small values of i–j.
- We make assumptions about $\{w[n]\}$ and $\{x[n]\}$: $r_w[n] \approx \sigma^2\delta[n]$ and $r_{wx}[n] \approx 0$.

11.4 APPLICATION: Period

Let $x[n]$ be a signal with unknown period N. The goal is to estimate N from the noisy observations

$$y[n] = x[n] + w[n] \quad (11.9)$$

The autocorrelation $r_y[n]$ of $y[n]$ is

$$\begin{aligned} r_y[n] &= (x[n] + w[n]) * (x[-n] + w[-n]) \\ &= x[n] * x[-n] + w[n] * w[-n] \\ &+ \underbrace{x[n] * w[-n]}_{r_{xw}[n]\approx 0} + \underbrace{x[-n] * w[n]}_{r_{xw}[-n]\approx 0} \\ &\approx r_x[n] + r_w[n] \approx r_x[n] + \sigma^2\delta[n] \end{aligned} \quad (11.10)$$

But since $x[n]$=$x[n+N]$, we have

$$\begin{aligned} r_x[N] &= \sum x[i]x[i+N] = \sum x[i]x[i] = r_x[0] \\ r_x[2N] &= \sum x[i]x[i+2N] = \sum x[i]x[i] = r_x[0] \\ &\vdots \quad \vdots \quad \vdots \\ r_x[n] &\text{ is periodic with period } N. \end{aligned} \quad (11.11)$$

So, to a good approximation,

$$r_y[n] \approx \begin{cases} \sigma^2 + \sum x[i]^2 & \text{for } n = 0 \\ \sum x[i]^2 & \text{for } n = N \\ \sum x[i]^2 & \text{for } n = 2N \\ \vdots & \vdots \\ 0 & \text{otherwise} \end{cases} \quad (11.12)$$

So the period N is found by looking for large peaks in $r_y[n]$, which will be at n=$0, N, 2N, \ldots$. The peak at n=0 is much larger, since σ^2 is added to it.

The following example demonstrates how to use autocorrelation to determine the period of a signal from noisy observations of it.

Example: Period Estimation
Determine the note played by a trumpet from noisy observations of the trumpet signal.

Solution:
The following three plots show:
1. Noiseless trumpet signal;
2. Trumpet signal with noise added;

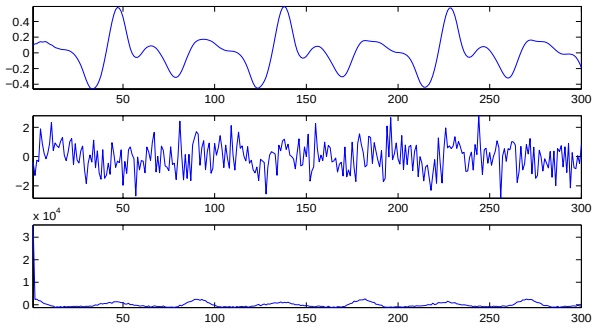


Figure 11.1: Estimating Period of Trumpet.

3. Autocorrelation of the noisy trumpet.

Try estimating the period by examining the second plot of the noisy trumpet signal! But the autocorrelation is clearly periodic with period 90, except the peak at n=0 is larger since σ^2 is added to it.

The sampling rate is the usual CD sampling rate of 44100 samples per sec., so the period is $\frac{90}{44100}=\frac{1}{490}$ sec. and the fundamental frequency is 490 Hertz. The trumpet is playing note B (494 Hertz).

The Matlab code used for this example:

```
load trumpet;%length(X)=32768
Y=X+randn(1,32768);%add noise.
R=real(ifft(abs(fft(Y,65536)).^2));
subplot(411),plot(X(1:300)),axis tight
subplot(412),plot(Y(1:300)),axis tight
subplot(413),plot(R(1:300)),axis tight
```

11.5 APPLICATION: Delay

Let $x[n]$ be a known signal with unknown delay N. Without loss of generality, the energy of $x[n]$ is one. The goal is to estimate N from the noisy observations

$$y[n] = x[n-N] + w[n] \qquad (11.13)$$

$w[n]$ is again white Gaussian noise. $r_{yx}[n]$ is

$$\begin{aligned} r_{yx}[n] &= (x[n-N]+w[n]) * x[-n] \\ &= \underbrace{x[n-N]*x[-n]}_{r_x[n-N]} + \underbrace{w[n]*x[-n]}_{r_{xw}[-n]\approx 0} \\ &\approx r_x[n-N] \approx \delta[n-N] \qquad (11.14) \end{aligned}$$

The final approximation $r_x[n] \approx \delta[n]$ is *very* rough. But the cross-correlation of the observations with the known signal is expected to have a peak at n=N.

Example: Time delay estimation.
A known short random pulse $x[n]$ is delayed by an unknown N. Estimate N from noisy observations.

Solution:
The following three plots show:
1. Noiseless already-delayed pulse.
2. Delayed pulse with noise added;
3. Cross-correlation of noisy delayed pulse with known non-delayed pulse.

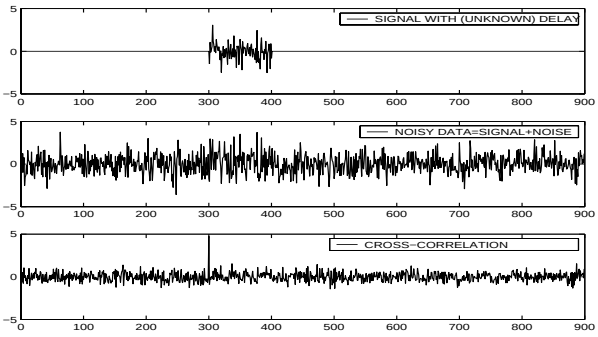


Figure 11.2: Time delay estimation.

Try estimating the delay by examining the second plot of the noisy delayed pulse! But the third plot shows clearly that the delay is N=300. This is an extreme example; usually the peak is not that sharp. But $r_x[n]$ does tend to have a decent peak at n=0.

11.6 APPLICATION: Classification

Suppose we have noisy observations of exactly one (but *which* one is unknown) of L possible known signals of durations N each: $\{x_1[n] \dots x_L[n]\}$:

$$y[n] = w[n] + \begin{cases} x_1[n] & \text{OR} \\ \vdots & \\ x_L[n] & \end{cases} \qquad (11.15)$$

where $w[n]$ is zero-mean white Gaussian noise. The goal is to determine which one of $\{x_1[n] \dots x_L[n]\}$ is present, from the noisy observations $y[n]$.

The solution is to choose the signal that minimizes

$$\sum_{n=0}^{N-1} \left[\frac{y[n]}{\sqrt{r_y[0]}} - \frac{x_i[n]}{\sqrt{r_{x_i}[0]}} \right]^2 \tag{11.16}$$

which is the "squared distance" between the energy-normalized data $y[n]$ and each of the energy-normalized signals $x_i[n]$. Using the same algebra (11.6) used to derive the Cauchy-Schwarz inequality, this is minimized by computing the correlation coefficient of $y[n]$ with each of $\{x_1[n] \dots x_L[n]\}$ and choosing the largest (closest to one):

$$\underset{i}{\text{argmax}} \left[\frac{r_{yx_i}[0]}{\sqrt{r_y[0] r_{x_i}[0]}} = \frac{\sum_{n=0}^{N-1} y[n] x_i[n]}{\sqrt{\sum_{n=0}^{N-1} y[n]^2 \sum_{n=0}^{N-1} x_i[n]^2}} \right] \tag{11.17}$$

Example: Time delay estimation, revisited

Formulate the time delay problem as a signal detection problem, and solve it using correlation.

Solution:

Formulate the time delay problem as

$$y[n] = w[n] + \begin{cases} x[n] & \text{OR} \\ x[n-1] & \text{OR} \\ x[n+1] & \text{OR} \\ \vdots \end{cases} \tag{11.18}$$

so that $x_i[n]$=$x[n-i]$. No energy normalization is necessary here, since all of the $x_i[n]$ have the same energy. The solution is then

$$\underset{i}{\text{argmax}} \left[\sum y[n] x_i[n] = \sum y[n] x[n-i] = r_{yx}[i] \right] \tag{11.19}$$

which agrees with the solution used before.

By combining time delay and signal detection, we can also incorporate unknown time delays N_i in each of the $x_i[n]$ in the general signal detection problem.

Example: Classification

We have noisy observations of one of these two signals, It is not evident which signal is present.

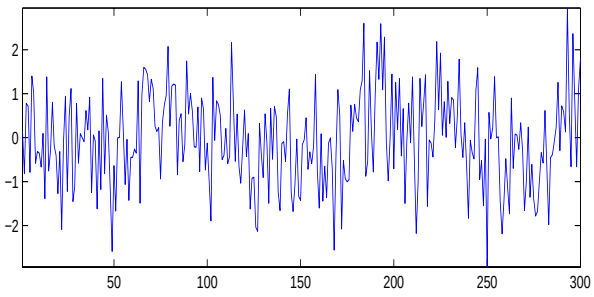


Figure 11.3: Noisy Observations of One Signal.

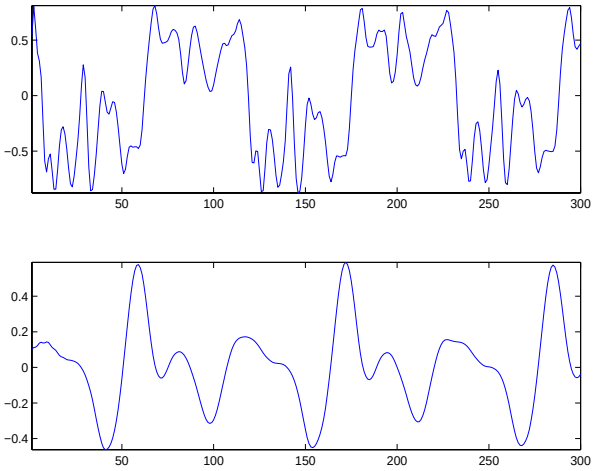


Figure 11.4: The Two Possible Signals.

The correlation coefficients of the noisy observations with each of the two signals are, respectively, 0.441 and -0.055. So the *first* signal is the one that is present. The signal to noise ratio was -5.95.

The Matlab code used for this example:

```
clear;load corre;N=randn(1,1000);
subplot(211),plot(X1(1:300)),axis tight
subplot(212),plot(X2(1:300)),axis tight
Y=X1+N;figure%Don't know it's X1.
subplot(211),plot(Y(1:300)),axis tight
RYX1=Y*X1'/sqrt(X1*X1')/sqrt(Y*Y');
RYX2=Y*X2'/sqrt(X2*X2')/sqrt(Y*Y');
[RYX1,RYX2,10*log10((X1*X1')/(N*N'))]
```

11.7 APPLICATIONS: Biomedical

We now present three biomedical applications:

- Computation of heartbeat rate from an EKG using autocorrelation;
- Classification of an EKG into one of three possible types (one normal, two abnormal);
- Computation of time delay in ultrasound.

A sampling rate of 100 samples/second is used. Each EKG signal has a duration of 10 seconds (1000 samples). The program used to generate these results is on the CD.

11.7.1 Electrocardiograms (EKG's)

An *electrocardiogram* (EKG) is a measurement of the electrical potential of the heart as a function of time. An EKG is measured using a collection of wire leads taped to the surface of the chest. Notch filters (see Section 8-2) are often used to remove 60 Hz interference from the wire leads. It is (hopefully!) periodic with a period of about one second (60 beats per minute); exercise raises this rate, but a fast heartbeat (i.e., smaller period) for an extended time is cause for concern.

An EKG waveform is useful in diagnosing heart malfunctions. Electrocardiograms are also called ECG's, but usually ECG refers to electroencephalogram (brain electrical potential).

Three types of EKG waveforms are shown in the next three figures. Although these waveforms are synthetic, they do bear some resemblance to actual EKG waveforms. The vertical scales are in millivolts (mv).

The first waveform is a normal EKG, with a period of one second (60 beats per minute).

The second waveform is an *atrial flutter*. Note there are three additional bumps in the waveform, so the heart is beating faster, at 240 beats per minute. The period of this waveform is one second, but its spectrum is dominated by its 4^{th} harmonic of 4 Hz (with period 0.25 second).

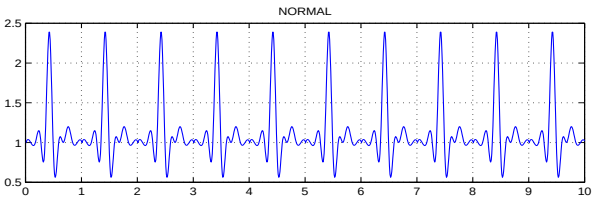


Figure 11.5: Normal EKG

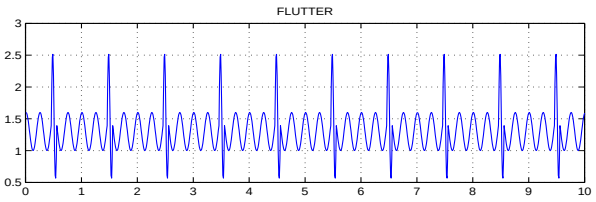


Figure 11.6: Atrial Flutter EKG

The third waveform is an *atrial fibrillation*, often called "atrial fib." The waveform consists entirely of bumps, at 300 beats per minute. The fundamental of its spectrum is at 5 Hz. This is a serious heart malfunction; a pacemaker is required to correct this to normal cardiac action.

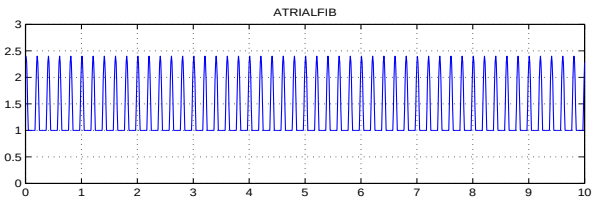


Figure 11.7: Atrial Fib EKG

11.7.2 Heart Rate by Autocorrelation

The obvious way to compute the period of a periodic signal in noise is to compute its autocorrelation.

Zero-mean white Gaussian noise with standard deviation 0.5 (see Section 6-9.1) was added to the ten-second-long segment of each waveform. Autocorrelations of the noisy signals were then computed. The noisy waveforms, and their autocorrelations, are shown below.

Normal EKG

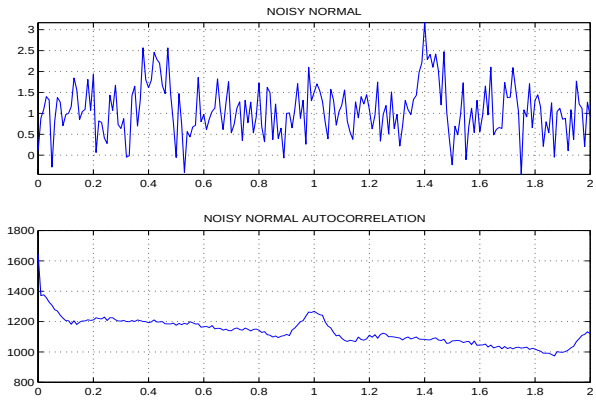


Figure 11.8: Noisy Normal EKG and its Autocorrelation

There is clearly a peak at one second, so this is the period (60 beats per minute). There is a second peak at double the period, as expected. Despite the noise, the peak is quite apparent.

Atrial Flutter EKG

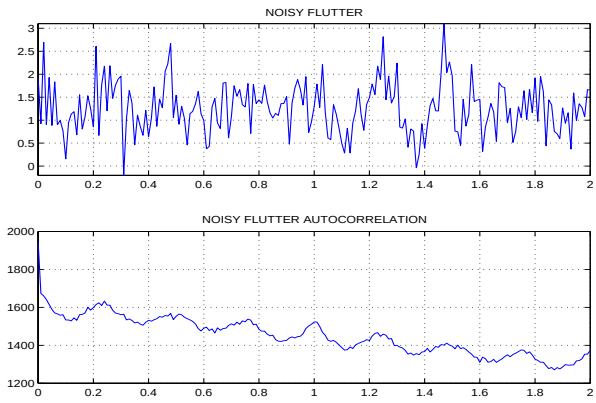


Figure 11.9: Noisy Atrial Flutter EKG and its Autocorrelation

There are peaks at $\{0.25, 0.50, 0.75 \ldots\}$ seconds, confirming that the period is 0.25 second (240 beats per minute). However, the larger peak at 1 second confirms that there is a second periodicity present. Althought the period of the signal is technically one second (it repeats every one second), the signal can also be regarded as the sum of two periodic signals with periods 0.25 and one second. This is not at all evident from the noisy waveform directly.

Atrial Fib EKG

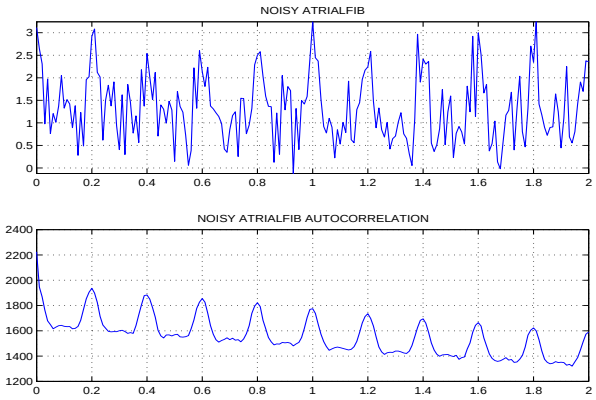


Figure 11.10: Noisy Atrial Fib EKG and its Autocorrelation

There are peaks at $\{0.2, 0.4, 0.6 \ldots\}$ seconds, confirming that the period is 0.2 second (300 beats per minute). This is not at all evident from the noisy waveform directly. This patient requires immediate attention!

11.7.3 Heart Rate from Spectrum

The period (1 second) of the atrial flutter EKG is not a good indicator of the actual heart rate (240 beats per minute). This suggests computing the spectrum of an EKG signal does a better job of indicating its features. This can also handle larger amounts of noise.

Zero-mean white Gaussian noise with standard deviation 1.0 (see Section 6-9.1) was added to the ten-second-long segment of each waveform. This is double the noise level used for autocorrelation above. Spectra of the noisy signals were then computed; these are shown below.

Normal EKG

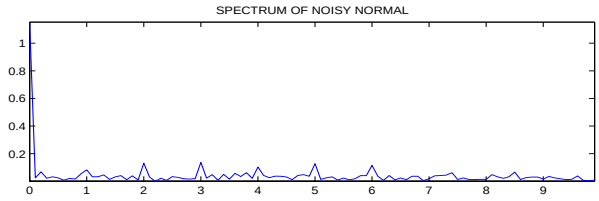


Figure 11.11: Spectrum of Noisy Normal EKG

There are clearly peak at 1,2,3... Hz (period one second). The harmonics appear to be significant only up to 10 Hz.

Atrial Flutter EKG

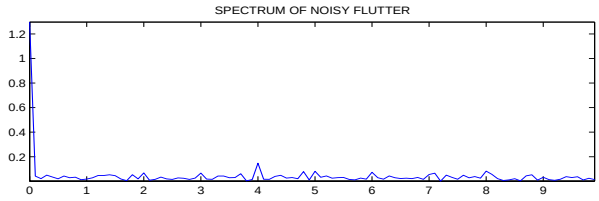


Figure 11.12: Spectrum of Noisy Atrial Flutter EKG

The fundamental at 1 Hz is barely discernable; the harmonic at 4 Hz (period 0.25 seconds) dominates the spectrum. This indicates a heart rate of 240 beats per minute, not the normal 60 beats per minute. The spectrum does a much better job of indicating the elevated heart rate.

Atrial Fib EKG

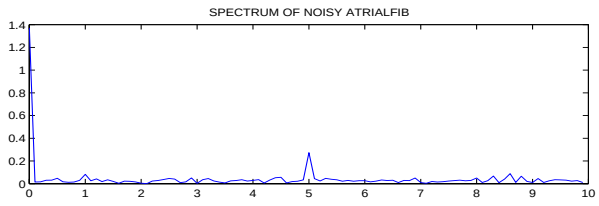


Figure 11.13: Spectrum of Noisy Atrial Fib EKG

The spectrum has a peak at 5 Hz (period 0.20 seconds) and this dominates the spectrum. This indicates an elevated heart rate of 300 beats per minute.

11.7.4 Classification

Zero-mean white Gaussian noise with standard deviation 0.5 (see Section 6-9.1) was added to the ten-second-long segment of one of the three EKG waveforms. The noisy waveform is shown. Which one of the three EKG waveforms is this?

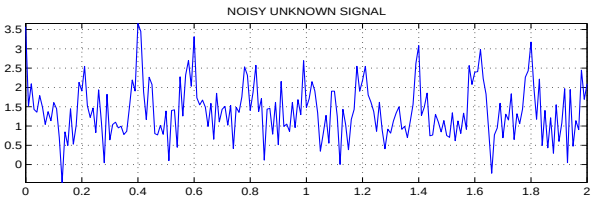


Figure 11.14: Noisy Unknown Waveform

The corrrelation coefficients of the noisy waveform with each of the three known waveforms were computed. The results were: $\{0.8763, 0.8586, 0.9451\}$, respectively. So the unknown waveform is found (correctly) to be atrial fib. This is important in cardiac diagnosis.

11.7.5 Ultrasound: Time Delay

In ultrasound imaging, a pulse which is a short (about one μsec) segment of a decaying (about 5 MHz) sinusoid is sent from a transducer into a body part. The pulse propagates into the body and reflects off of a bone, organ, or inclusion. A few μsec later, the reflected pulse returns to the transducer. The time delay between the transmitted and received pulses is the two-way traveltime to the inclusion. Knowing the speed of ultrasound propagation in the body (about 1.54 mm/μsec), the depth of the inclusion can be computed from this time delay. A delay of 13 μsec indicates a depth of $\frac{1}{2}(1.54\frac{\text{mm}}{\mu\text{sec}})(13\mu\text{sec})$=10 mm=1 cm. However, because of scattering of the pulse off of soft body tissues, the transducer measures not only the delayed pulse, but noise.

An ultrasound pulse was modelled as a 1μsec segment of a 5 MHz sinusoid that decayed from 2 V to 1 V in the 1μsec. The pulse was delayed by an unknown time, and zero-mean white Gaussian noise with standard deviation two (see Section 6-9.1) was added to the delayed pulse. The sampling rate is 10^8 samples/second.

The noisy received signal, and its cross-correlation with the known pulse, are shown below. The amount by which the pulse is delayed is not at all evident from the noisy received signal, but is clearly seen to be 30μsec on the cross-correlation plot. This corresponds to an inclusion depth of 2.31 cm.

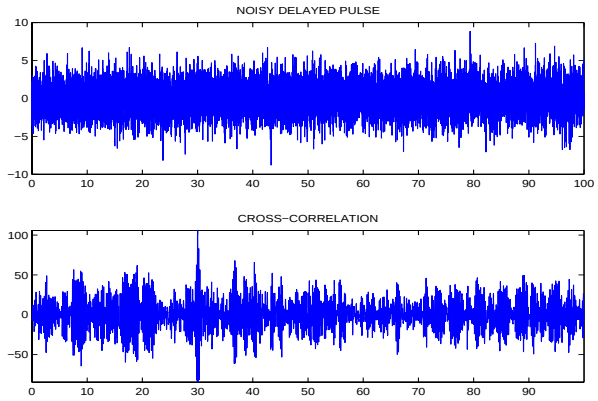


Figure 11.15: Noisy Delayed Ultrasound Pulse and Cross-correlation

Chapter 12

Image Processing

A significant part of image processing is the extension of the material presented in these notes from 1-D to 2-D. Some basic concepts in image processing, specifically the basics of noise filtering and deconvolution, can be presented here. An image processing course provides more complete treatment.

12.1 Image Processing Basics

12.1.1 Extensions from 1-D to 2-D

The following concepts generalize in a straightforward manner from 1-D to 2-D:

- LTI; impulses; convolution; impulse response (now called *point-spread* function);
- Frequency (now *wavenumber*) response;
- Fourier transforms; sampling theorem;
- DTFT (now DSFT); DTFS; DFT; FFT.

The following concepts do *not* generalize in a useful manner from 1-D to 2-D:

- Laplace transforms; difference equations;
- Transfer functions; poles and zeros;
- Partial fraction expansions; z-transforms.

12.1.2 2-D Signals and Sampling

An *image* is a 2-D discrete-*space* signal $x[m,n]$, where m and n are integers. Images are often obtained by sampling a 2-D continuous-*space* signal $z(x,y)$ at points $x{=}mT$ and $y{=}nT$ for some small discretization *length* T. Images are usually very oversampled. Discrete images may also be generated directly. Each value of $x[m,n]$ is called a *pixel* of the image.

The 2-D continuous-space Fourier transform consists of two successive 1-D continuous Fourier transforms. One Fourier transform takes spatial variable x to *wavenumber* (spatial frequency) k_x, and the other takes spatial variable y to wavenumber k_y, giving

$$Z(k_x,k_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} z(x,y)e^{-(jk_x x+k_y y)}dxdy. \tag{12.1}$$

$z(x,y)$ is bandlimited to B radians per length if $|Z(k_x,k_y)|{=}0$ unless both $|k_x| < B$ and $|k_y| < B$. Applying the sampling theorem to x and then y,

- If 2-D continuous-space signal $z(x,y)$ is bandlimited to *wavenumber* $\frac{2\pi}{2T}$ radians per length, then $z(x,y)$ can be uniquely reconstructed from its samples $z[m,n]{=}z(mT,nT)$ for integers m,n.

Image processing is usually performed on discrete-space signals $x[m,n]$. For the rest of this chapter, we will consider only discrete-space image processing.

12.1.3 LSI Systems and Convolution

A 2-D system accepts as input an image $x[m,n]$ and outputs another image $y[m,n]$.

A 2-D system is LSI (Linear *Shift* Invariant) if:

- $x[m,n] \rightarrow \boxed{\text{LSI}} \rightarrow y[m,n]$ implies:

- $x[m–M, n–N] \rightarrow \boxed{\text{LSI}} \rightarrow y[m–M, n–N]$ for any integers M and N;
- $cx[m,n] \rightarrow \boxed{\text{LSI}} \rightarrow cy[m,n]$ for any constant c;
- The response to the sum of inputs is the sum of the responses to the inputs.

This is a direct generalization of 1-D LTI to 2-D LSI.

A 2-D impulse $\delta[m,n]$ is defined as

$$\delta[m,n] = \begin{cases} 1 & m = n = 0 \\ 0 & \text{otherwise} \end{cases} \quad (12.2)$$

The *Point-Spread* Function (PSF) $h[m,n]$ of a 2-D LSI system is its 2-D impulse response:

- $\delta[m,n] \rightarrow \boxed{\text{LSI}} \rightarrow h[m,n]$.

In medical imaging systems, the PSF $h[m,n]$ is sometimes measured physically by imaging a small bead, which acts as a 2-D impulse $\delta[m,n]$. $h[m,n]$ is the spread of the bead, its "point-spread function."

The response of a 2-D LSI system to any image $x[m,n]$ is the 2-D convolution with its PSF $h[m,n]$:

$$\begin{aligned} y[m,n] &= h[m,n] * *x[m,n] \quad (12.3) \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h[i,j]x[m-i,n-j]. \end{aligned}$$

The derivation of this result is analogous to the derivation of 1-D discrete-time convolutions.

Example: Compute the 2-D Convolution:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = ? \quad (12.4)$$

Solution: We have

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 16 & 12 \\ 22 & 60 & 40 \\ 21 & 52 & 32 \end{bmatrix} =$$

$$\begin{bmatrix} 1 \cdot 5 & 1 \cdot 6 + 2 \cdot 5 & 2 \cdot 6 \\ 1 \cdot 7 + 3 \cdot 5 & 4 \cdot 5 + 3 \cdot 6 + 2 \cdot 7 + 1 \cdot 8 & 2 \cdot 8 + 4 \cdot 6 \\ 3 \cdot 7 & 3 \cdot 8 + 4 \cdot 7 & 4 \cdot 8 \end{bmatrix}$$

Another way to compute this is as follows:

$$1 \begin{bmatrix} 5 & 6 & 0 \\ 7 & 8 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2 \begin{bmatrix} 0 & 5 & 6 \\ 0 & 7 & 8 \\ 0 & 0 & 0 \end{bmatrix} + 3 \begin{bmatrix} 0 & 0 & 0 \\ 5 & 6 & 0 \\ 7 & 8 & 0 \end{bmatrix} + 4 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 6 \\ 0 & 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 16 & 12 \\ 22 & 60 & 40 \\ 21 & 52 & 32 \end{bmatrix} \quad (12.5)$$

This illustrates the term "point-spread function."

Matlab code for this example:

```
conv2([1 2;3 4],[5 6;7 8])
```

12.2 Fourier Transforms

As in 1-D, the two important Fourier transforms of images are the DSFT and the 2-D DFT.

12.2.1 DSFT

The Discrete-Space-Fourier-Transform (DSFT) is the 2-D DTFT computed by successive 1-D DTFTs:

$$X(e^{j\omega_1}, e^{j\omega_2}) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[m,n] e^{-j(\omega_1 m + \omega_2 n)}. \quad (12.6)$$

The DSFT is periodic in ω_1 and ω_2 with periods 2π.

The following two applications of the (1-D) DTFT generalize directly to the (2-D) DSFT:

- The DSFT of an image $x[m,n]$ is its 2-D spectrum $X(e^{j\omega_1}, e^{j\omega_2})$;
- The DSFT of the point-spread function $h[m,n]$ is the wavenumber response $H(e^{j\omega_1}, e^{j\omega_2})$:

$$\begin{aligned} \cos(\omega_1 m + \omega_2 n) &\rightarrow \boxed{H(e^{j\omega_1}, e^{j\omega_2})} \rightarrow \\ &|H(e^{j\omega_1}, e^{j\omega_2})| \cos(\omega_1 m + \omega_2 n + \theta). \\ H(e^{j\omega_1}, e^{j\omega_2}) &= |H(e^{j\omega_1}, e^{j\omega_2})| e^{j\theta}. \quad (12.7) \end{aligned}$$

The spectrum of an image is usually much less important than the spectrum of a signal. One situation where the spectrum does indicate a property of the image is depicted in the following example.

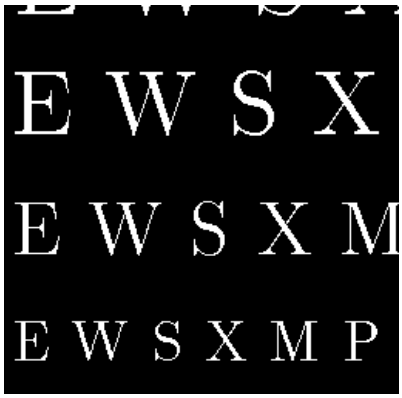


Figure 12.1: Eyechart Image.

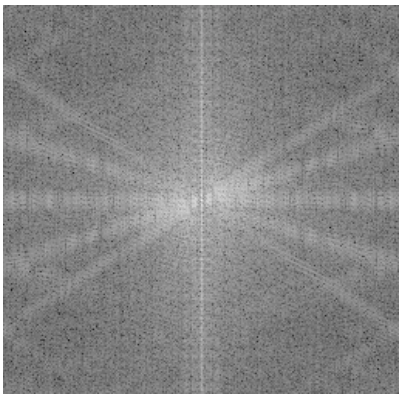


Figure 12.2: Log of Spectrum of Eyechart.

Example: Compute the 2-D Spectrum of:

Solution: Since the image is non-negative, an image of its spectrum appears as a single dot at (ω_1, ω_2)=(0,0), since its dc value

$$X(e^{j0}, e^{j0}) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] \tag{12.8}$$

is much larger than other values of $|X(e^{j\omega_1}, e^{j\omega_2})|$. So we plot not $|X(e^{j\omega_1}, e^{j\omega_2})|$, but its *logarithm*. Taking logarithms of a set of numbers reduces their range of values. Here, features of the spectrum appear in $\log|X(e^{j\omega_1}, e^{j\omega_2})|$ that do not appear in $|X(e^{j\omega_1}, e^{j\omega_2})|$. We obtain

Note the following about this 2-D spectrum:

- The dc value $X(e^{j0}, e^{j0})$ appears in the center, due to `fftshift` in the program below;
- The ranges of wavenumbers ω_1 and ω_2 are both $-\pi < \omega_1, \omega_2 < \pi$.
- The DSFT is periodic in ω_1 and ω_2 with periods 2π, as the DTFT is periodic with period 2π. Only the single period centered at dc is plotted.
- The lines in the spectrum appear because the letters in the original image consist mostly of line segments oriented in four directions.

Matlab code for this example:

```
clear;load echart.mat;
X=255-echart;X=X(2:257,:);
imagesc(X),colormap(gray),axis off
FX=fftshift(log(abs(fft2(X))));
imagesc(FX),colormap(gray),axis off
```

12.2.2 2-D DFT

In practice, the DSFT of an $(M \times N)$ image or PSF is computed numerically by sampling in wavenumber. This results in the 2-D Discrete Fourier Transform (DFT), just as in 1-D.

Sampling the DSFT of an $(M \times N)$ image at

- $\omega_1 = 2\pi \frac{k_1}{M}, k_1 = 0, 1 \ldots (M-1)$
- $\omega_2 = 2\pi \frac{k_2}{N}, k_2 = 0, 1 \ldots (N-1)$

gives the $(M \times N)$ 2-D Discrete Fourier Transform

$$X_{m,n} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] e^{-j2\pi(\frac{k_1 m}{M} + \frac{k_2 n}{N})}. \tag{12.9}$$

The 2-D DFT of `X` can be computed rapidly using a 2-D version of the FFT. In Matlab, it can be computed using `fft2(X)=fft((fft(X)).')`, which applies the 1-D FFT to each row, and then to each column.

The 2-D DFT can also be used to compute the 2-D *cyclic* convolution of two images, as the following example shows.

Example: Compute the 2-D Convolution:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \textcircled{c}\textcircled{c} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = ? \tag{12.10}$$

Solution: The (2×2) 2-D DFTs are:

$$\begin{bmatrix} 1+2+3+4 & 1+3-2-4 \\ 1+2-3-4 & 1+4-2-3 \end{bmatrix} = \begin{bmatrix} 10 & -2 \\ -4 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 5+6+7+8 & 5+7-6-8 \\ 5+6-7-8 & 5+8-6-7 \end{bmatrix} = [\,26 \quad -2 \quad -4 \quad 0\,]$$

$$\begin{bmatrix} 26 \cdot 10 & -2 \cdot -2 \\ -4 \cdot -4 & 0 \cdot 0 \end{bmatrix} = \begin{bmatrix} 260 & 4 \\ 16 & 0 \end{bmatrix}$$

$$\frac{1}{2^2} \begin{bmatrix} 260+4+16 & 260+16-4 \\ 260+4-16 & 260-4-16 \end{bmatrix} = \begin{bmatrix} 70 & 68 \\ 62 & 60 \end{bmatrix}$$

Matlab code for this example:

```
FX=fft2([1 2;3 4]);FY=fft2([5 6;7 8]);
Y=real(ifft2(FX.*FY));
```

This 2-D cyclic convolution can also be computed by aliasing the 2-D linear convolution computed above:

$$\begin{bmatrix} 5 & 16 & 12 \\ 22 & 60 & 40 \\ 21 & 52 & 32 \end{bmatrix} \rightarrow \begin{bmatrix} 5{+}12 & 16 \\ 22{+}40 & 60 \\ 21{+}32 & 52 \end{bmatrix} \rightarrow \begin{bmatrix} 17{+}53 & 16{+}52 \\ 62 & 60 \end{bmatrix} \tag{12.11}$$

which gives the same answer.

12.3 Wavenumber Response

The concepts and effects of filtering and spectrum generalize directly from 1-D to 2-D, as the following example demonstrates.

Example: 2-D Wavenumber Response.
Compute the point-spread function and the 2-D wavenumber response of the 2-D LSI system

$$\begin{aligned} y[m,n] &= \frac{1}{4}x[m,n] + \frac{1}{4}x[m-1,n-1] \\ &+ \frac{1}{4}x[m-1,n] + \frac{1}{4}x[m,n-1] \end{aligned} \tag{12.12}$$

This system averages the pixels in a 2×2 block of the image, so it is a 2-D version of the two-point averager.

Solution:
We can read off the PSF $h[m,n]$, as in 1-D:

$$\begin{aligned} h[m,n] &= \frac{1}{4}\delta[m,n] + \frac{1}{4}\delta[m-1,n-1] \\ &+ \frac{1}{4}\delta[m-1,n] + \frac{1}{4}\delta[m,n-1] \\ &= \frac{1}{4}\begin{bmatrix} \underline{1} & 1 \\ 1 & 1 \end{bmatrix}. \end{aligned} \tag{12.13}$$

The 2-D point-spread function $h[m,n]$ can be regarded as a 2×2 image of pixel values $\frac{1}{4}$ each.

The 2-D wavenumber response $H(e^{j\omega_1}, e^{j\omega_2})$ is the DSFT of the PSF $h[m,n]$. Using phase splitting,

$$\begin{aligned} H(e^{j\omega_1}, e^{j\omega_2}) &= \sum_{m=0}^{1}\sum_{n=0}^{1} \frac{1}{4} e^{-j(\omega_1 m + \omega_2 n)} \\ &= \frac{1}{4}[1 + e^{-j(\omega_1+\omega_2)} + e^{-j\omega_1} + e^{-j\omega_2}] \\ &= \cos(\omega_1/2)\cos(\omega_2/2)e^{-j(\omega_1+\omega_2)/2}. \end{aligned} \tag{12.14}$$

The 2-D gain $|H(e^{j\omega_1}, e^{j\omega_2})|$ is plotted in the 3-D plot below. The origin $(\omega_1, \omega_2){=}(0,0)$ is in the middle of the plot (the gain is one there), and the highest wavenumbers $\omega_1{=}\pm\pi$ and $\omega_2{=}\pm\pi$ at the edges.

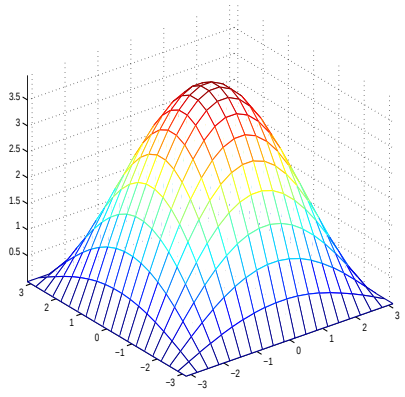


Figure 12.3: 2-D gain for low-pass filter.

The system is a crude spatial low-pass filter. For low wavenumbers $(\omega_1, \omega_2) \approx 0$, the gain$\approx$1. For large

wavenumbers $\omega_1 \approx \pi$ or $\omega_2 \approx \pi$, the gain is close to zero. So low-wavenumber components of the input image are passed, and high-wavenumber components of the input image rejected. Also,

$$H(e^{j\omega_1}, e^{j\omega_2}) = H_1(e^{j\omega_1})H_1(e^{j\omega_2}) \qquad (12.15)$$

where $H_1(e^{j\omega})$ is the frequency response of the two-point averager from a previous chapter.

The Matlab code for this example:

```
W=linspace(-pi,pi,20);
[X Y]=meshgrid(W,W);
Z=4*cos(X/2).*cos(Y/2);
mesh(X,Y,Z),axis tight
```

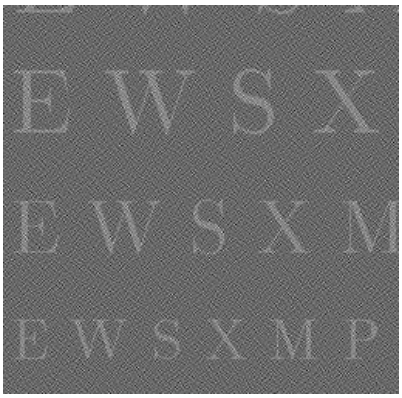


Figure 12.4: Noisy image.

12.4 2-D Noise Filtering

12.4.1 Implementation of Filtering

Image processing is usually performed using batch processing, often using the 2-D DFT, computed using the 2-D FFT. The FFT is even more important in 2-D than in 1-D, since 2-D images often contain more samples (pixels) than 1-D signals.

The simplest way to perform noise filtering of images is to set high wavenumber components of the noisy image to zero. This implements a 2-D brick-wall low-pass filter. However, there is a tradeoff: a small cutoff wavenumber eliminates more noise, but also eliminates some wavenumber components of the image. This has the effect of distorting edges in the image. A high cutoff wavenumber has less effect on the image, but also eliminates less noise.

The following example illustrates this tradeoff.

Example: DFT to noise filter images.

An image of letters has high-wavenumber noise added to it. Use the 2-D DFT to reduce the noise.

Solution:

The noisy image is shown in the next figure.

Two 2-D brick-wall low-pass filters with two different cutoffs are implemented using the 2-D DFT. The filtered images are shown in the next two figures for low and high cutoffs, respectively.

The Matlab code for this example:

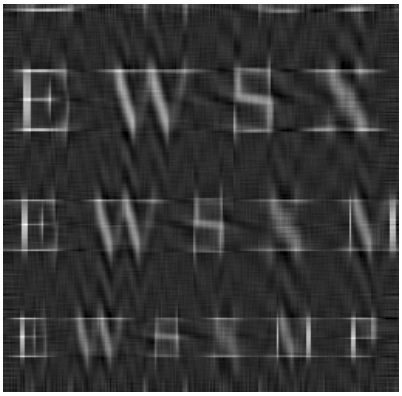


Figure 12.5: Image filtered with cutoff L=10.

```
clear;load echart.mat;
X=255-echart;X=X(2:257,:);
N=randn(257,257);
NN1=N(1:256,1:256)-N(1:256,2:257);
NN2=N(2:257,1:256)-N(2:257,2:257);
Y=X+100*(NN1-NN2);FY=fft2(Y);
FY(20:238,20:238)=0;Z1=real(ifft2(FY));
FY(10:248,10:248)=0;Z2=real(ifft2(FY));
imagesc(Y),colormap(gray),axis off
imagesc(Z2),colormap(gray),axis off
imagesc(Z1),colormap(gray),axis off
```

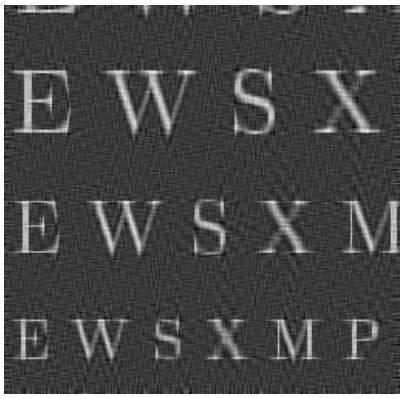


Figure 12.6: Image filtered with cutoff L=20.

12.4.2 Discussion of Results

Both filters greatly reduced the noise in the image.

The image filtered with a lower cutoff has less noise, but the letters are distorted.

The image filtered with a higher cutoff has more noise, but the letters are not distorted.

This is a simple example of *bias-variance tradeoff*. The noise can be reduced (low variance) at the price of distorting the image (high bias). Or the image can be undistorted (low bias) at the price of high noise level (high variance). This tradeoff occurs throughout signal and image processing.

The noise was generated by passing white Gaussian noise through the 2-D system

$$\begin{aligned} y[m,n] &= x[m,n] + x[m-1,n-1] \\ &- x[m-1,n] - x[m,n-1]. \end{aligned} \quad (12.16)$$

The reader should be able to show the 2-D gain is

$$|H(e^{j\omega_1}, e^{j\omega_2})| = 4|\sin(\omega_1/2)\sin(\omega_2/2)| \quad (12.17)$$

which is a crude high-pass filter which emphasizes the high-wavenumber components of the noise.

The program does not set all high wavenumber components to zero. This is because the image is known to consist mostly of vertical and horizontal components, so that components along both the ω_1 or ω_2 axes are significant. Keeping these components greatly improves the filtered image. Note that the blurring shows up mostly along the "X's," which have components at 45^o, not horizontal or vertical ones.

12.5 Image Deconvolution

12.5.1 Problem Formulation

The image deconvolution problem is to reconstruct an image $x[m,n]$ from its convolution $y[m,n]$ with a known point-spread-function (PSF) $h[m,n]$. Many problems in medical and optical imaging can be formulated as image deconvolution problems. The PSF represents the effects of any of the following:

- The imaging system;
- Blurring caused by defocusing;
- An antenna aperture function;
- Distortion caused by the atmosphere.

Taking the 2-D DFT of the imaging equation

$$y[m,n] = h[m,n] * *x[m,n] \quad (12.18)$$

gives, in analogy to 1-D deconvolution,

$$X_{k_1,k_2} = Y_{k_1,k_2}/H_{k_1,k_2}. \quad (12.19)$$

Example: Noiseless 2-D deconvolution.

The letters image is convolved with a 2-D Gaussian PSF (a common test PSF), blurring it. Reconstruct the image from its blurred version.

Solution:

The image sizes are as follows:

- Letters image: (256×256).
- Gaussian PSF: (21×21).
- Blurred image: (276×276) (256+21–1=276).

We zero-pad all the images to (280×280), since 280=(2)(2)(2)(5)(7) has many factors, so the DFT will be fast. The blurred image is in the next figure.

The deconvolved image is shown in the next figure. The reconstruction is accurate, but it is very sensitive to noise, as the next example will show.

The Matlab code for this example:

```
clear;load echart.mat;
X=255-echart;X=X(2:257,:);
H=exp(-[-10:10].*[-10:10]/20);
FH=fft2(H'*H,280,280);
```

Figure 12.7: Image blurred with Gaussian PSF.

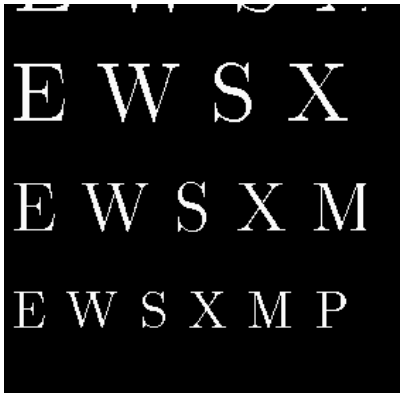


Figure 12.8: Image deconvolved from noiseless data.

```
FX=fft2(X,280,280);FY=FH.*FX;
Y=real(ifft2(FY));
imagesc(Y),colormap(gray),axis off
Z=real(ifft2(FY./FH));
imagesc(Z),colormap(gray),axis off
```

12.5.2 Tikhonov Regularization: I

The problem with using the DFT to perform deconvolution, as performed above, is that in many practical applications $H_{k_1,k_2} \approx 0$ for large wavenumbers k_1 and k_2. This means that the factor $\frac{1}{H_{k_1,k_2}}$ multiplying Y_{k_1,k_2} will be very large and any noise present in Y_{k_1,k_2} will be amplified, for large wavenumbers.

To get around this difficulty, the deconvolution problem must be *regularized*. This means that we do not solve the deconvolution problem itself, but a variation of it. Usually this requires minimizing a *cost functional* that trades off accuracy of the reconstruction (bias) with the noise in it (variance). Bias-variance tradeoff was encountered previously in the noise filtering problem. Now it appears again in the context of deconvolution.

A common type of regularization is *Tikhonov regularization*. In Tikhonov regularization, instead of simply solving the problem $y[m,n]{=}h[m,n]{*}{*}x[m,n]$ for $x[m,n]$, we minimize over $x[m,n]$ the cost

$$\sum\sum[(y[m,n]-h[m,n]*{*}x[m,n])^2+\lambda^2(x[m,n])^2] \tag{12.20}$$

for some parameter λ. λ specifies the tradeoff between accuracy of the reconstruction (the first term, i.e., the bias) and noise in the reconstructed image (the second term, i.e., the variance):

- Small λ leads to accurate but noisy reconstructed images $x[m,n]$
- Large λ leads to less accurate but less noisy reconstructed images.

Note that this is the same tradeoff that we saw in the image filtering problem, except that the tradeoff was specified by the cutoff wavenumber in the brick-wall low-pass filter.

12.5.3 Tikhonov Regularization: II

Using Parseval's theorem for the DFT, and assuming that $x[m,n]$ and $h[m,n]$ have been zero-padded, the Tikhonov cost functional (12.20) is equal to

$$\frac{1}{MN}\sum\sum[|Y_{k_1,k_2}-H_{k_1,k_2}X_{k_1,k_2}|^2+\lambda^2|X_{k_1,k_2}|^2]. \tag{12.21}$$

This can be minimized separately for each (k_1,k_2). The X_{k_1,k_2} minimizing this for each (k_1,k_2) is

$$X_{k_1,k_2}=Y_{k_1,k_2}\frac{H^*_{k_1,k_2}}{|H_{k_1,k_2}|^2+\lambda^2}. \tag{12.22}$$

This is sometimes called a *Wiener* filter. A Wiener filter is used often in image reconstruction. Its effect can be summarized as follows.

Unless $H_{k_1,k_2} \approx 0$, the Wiener filter becomes

$$X_{k_1,k_2} = Y_{k_1,k_2} \frac{H^*_{k_1,k_2}}{|H_{k_1,k_2}|^2} = \frac{Y_{k_1,k_2}}{H_{k_1,k_2}}. \qquad (12.23)$$

which is just the basis of DFT-based deconvolution.

But if $H_{k_1,k_2} \approx 0$, the Wiener filter becomes

$$X_{k_1,k_2} = Y_{k_1,k_2} \frac{H^*_{k_1,k_2}}{\lambda^2}. \qquad (12.24)$$

which keeps X_{k_1,k_2} from getting too large. Of course, this X_{k_1,k_2} is wrong, but since $H_{k_1,k_2} \approx 0$ there is no way to recover X_{k_1,k_2} from Y_{k_1,k_2}=$H_{k_1,k_2}X_{k_1,k_2}$, so the best thing to do is to at least keep the unknown values of X_{k_1,k_2} small, so they don't overwhelm the correct values of X_{k_1,k_2}.

Example: Noisy deconvolution of images.

The letters image is convolved with a 2-D Gaussian PSF, blurring it. Now a small amount of noise is added to the blurred image. Reconstruct the image from its blurred version as best you can.

Solution:

The *noisy* blurred image is in the next figure.

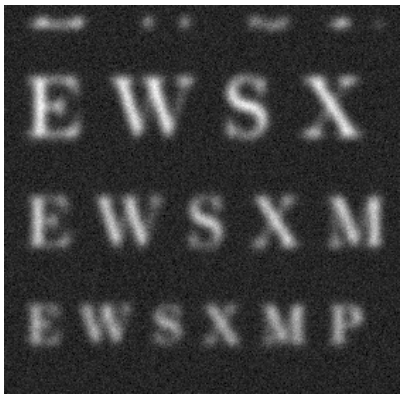


Figure 12.9: Noisy blurred image.

The image deconvolved from noisy data ($\lambda = 0$) is in the next figure. It is swamped with noise.

The image deconvolved from noisy data ($\lambda = 5$) is shown in the next figure. While far from perfect, it is a reasonable reconstruction. Try different values of λ to see which one gives the best reconstruction.

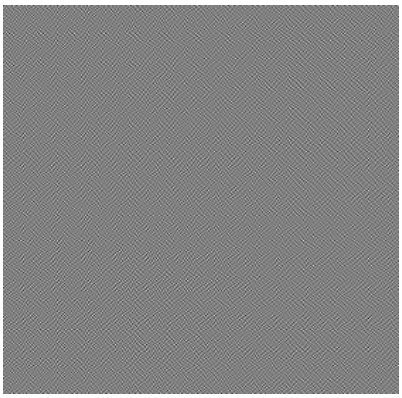


Figure 12.10: Image deconvolved from noisy data.

Figure 12.11: Image deconvolved from noisy data with regularization.

The Matlab code for this example:

```
clear;load echart.mat;
X=255-echart;X=X(2:257,:);
H=exp(-[-10:10].*[-10:10]/20);
FH=fft2(H'*H,280,280);
FX=fft2(X,280,280);FY=FH.*FX;
Y=real(ifft2(FY));
Y=Y+500*randn(280,280);
imagesc(Y),colormap(gray),axis off
FY=fft2(Y);Z=real(ifft2(FY./FH));
imagesc(Z),colormap(gray),axis off
FW=conj(FH)./(abs(FH).*abs(FH)+5);
```

```
W=real(ifft2(FY.*FW));
imagesc(W),colormap(gray),axis off
```

12.6 Median Filtering

While not LSI, median filtering is very useful in eliminating noise that consists of isolated points. This is called *salt-and-pepper* or *shot* noise, and it arises from bit errors or Poisson noise.

Median filtering replaces each pixel with the median of the 9 pixels surrounding it (including itself):

- $y[m,n]$=median of these 9 pixels:
  $\{x[m–1,n–1]$, $x[m–1,n]$, $x[m–1,n+1]$
  $x[m,n–1]$, $x[m,n]$, $x[m,n+1]$ and
  $x[m+1,n–1]$, $x[m+1,n]$, $x[m+1,n+1]\}$.

Figure 12.12: Example of median filtering.

```
clear;load echart.mat;
X=255-echart;X=X(2:257,:);
Y=X+200*floor(rand(256,256)+0.1);
Z=medfilt2(Y);
subplot(221),imagesc(Y),colormap(gray)
subplot(222),imagesc(Z),colormap(gray)
```

Chapter 13

Matlab: A Short Introduction

"A computer will always do exactly what you tell it to do. But that may not be what you had in mind"–a quote from the 1950's.

This Appendix is a short introduction to Matlab for basic discrete-time signal processing. It is not comprehensive; only commands directly applicable to discrete-time signal processing are covered. Programming concepts and techniques are not included, since they are not used anywhere in this book.

13.1 Introduction

Matlab is a computer program developed and sold by the Mathworks, Inc. It is the most commonly used program in signal processing, but it is used in all fields of engineering.

"Matlab" is an abbreviation of MATrix LABoratory. It was originally based on a set of numerical linear algebra programs, written in FORTRAN, called LINPACK. So Matlab tends to formulate problems in terms of vectors and arrays of numbers, and often solves them by formulating them as linear algebra.

For example, Matlab computes the roots of a polynomial by computing the eigenvalues of the companion matrix formed from the polynomial coefficients. When using Matlab, one usually formulates a problem in terms of arrays or vectors of numbers.

13.1.1 Getting Started

When you run Matlab, a *prompt* >> will appear when it is ready. Then you can type commands. Your first command should be >>`cd mydirectory`, to change directory to your working directory.

We will use `this font` to represent typed commands and generated output. You can get help for any command, such as `plot`, by typing `help plot`.

Some basic things to know about Matlab:

- Putting a **semicolon** ";" at the end of a command suppresses output; without it Matlab will type the results of the computation. This is harmless, but it is irritating to have numbers flying by on your screen.
- Putting **ellipses** "..." at the end of a command means it is continued on the next line. This is useful for long commands.
- Putting "%" at the beginning of a line makes the line a **comment**; it will not be executed. Comments are used to explain what the program is doing at that point.
- `clear` eliminates all present variables. Programs should start with a `clear`.
- `whos` shows all variables and their sizes.
- Matlab variables are case-sensitive: `t` and `T` are different variables.
- `save myfile X,Y` saves the variables `X` and `Y` in the file myfile.mat for use in another session of Matlab at another time.
- `load myfile` loads all variables saved in myfile.mat, so they can now be used in the present session of Matlab.

- `quit` ends the present session of Matlab.

13.1.2 Scripts (m-Files)

An Matlab program is a list of commands executed in succession. Programs are called "scripts" or "m-files" since their extension is ".m."

To write a script, at the upper left, click:
File→New→script
This opens a window with a text editor.
Type in your commands and then do this:
File→Save as→myname.m

Make sure you save it with an .m extension. Then you can run the file by typing its name at the prompt: >> `myname`. Make sure the file name is not the same as a Matlab command! Try using your own name.

You can access previously-typed commands using uparrow and downarrow on your keyboard.

To *download a file* from a web site or the CD, right-click on it, select **save target as**, and use the menu to select the proper file type (by file extension).

13.2 Basic Computation

13.2.1 Basic Arithmetic

- Addition: `3+2` gives `ans=5`
- Subtraction: `3-2` gives `ans=1`
- Multiplication: `2*3` gives `ans=6`
- Division: `6/2` gives `ans=3`
- Powers: `2^3` gives `ans=8`
- Others: `sin,cos,tan,exp,log,log10`
- Square root: `sqrt(49)` gives `ans=7`
- Conjugate: `conj(3+2j)` gives `ans=3-2i`

Both `i` or `j` represent $\sqrt{-1}$; answers use `i`.
`pi` represents π. `e` does not represent 2.71828.

13.2.2 Entering Vectors and Arrays

To enter *row vector* [1 2 3] and store it in `A`
type at the prompt `A=[1 2 3];` or `A=[1,2,3];`

To enter the same numbers as a *column vector* and store it in `A`, type at the prompt *either* `A=[1;2;3];` *or* `A=[1 2 3];A=A';` Note `A=A'` replaces `A` with its transpose. "Transpose" means "convert rows to columns, and vice-versa."

To enter a vector of consecutive or equally-spaced numbers, follow these examples:

- `[2:6]` gives `ans=2 3 4 5 6`
- `[3:2:9]` gives `ans=3 5 7 9`
- `[4:-1:1]` gives `ans=4 3 2 1`

To enter an *array* or matrix of numbers,
type, for example, `B=[3 1 4;1 5 9;2 6 5];`
This gives the array `B` and its transpose `B'`

$$B = \begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix} \qquad B' = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 5 & 6 \\ 4 & 9 & 5 \end{bmatrix}$$

Other basics of arrays:

- `ones(M,N)` is an $M \times N$ array of "1"
- `zeros(M,N)` is an $M \times N$ array of "0"
- `length(X)` gives the length of vector `X`
- `size(X)` gives the size of array `X`
  For `B` above, `size(B)` gives `ans=3 3`
- `A(I,J)` gives the (I,J)th element of `A`.
  For `B` above, `B(2,3)` gives `ans=9`

13.2.3 Array Operations

Arrays add and subtract point-by-point:
`X=[3 1 4];Y=[2 7 3];X+Y` gives `ans=5 8 7`
But `X*Y` generates an error message.
To compute various types of vector products:

- To multiply element-by-element,
  use `X.*Y` This gives `ans=6 7 12`.
  To divide element-by-element, use `X./Y`

- To find the inner product of `X` and `Y` (3)(2)+(1)(7)+(4)(3)=25, use `X*Y'` This gives `ans=25`
- To find the outer product of `X` and `Y` $\begin{bmatrix}(3)(2) & (3)(7) & (3)(3)\\(1)(2) & (1)(7) & (1)(3)\\(4)(2) & (4)(7) & (4)(3)\end{bmatrix}$ use `X'*Y` This gives the above matrix.

A common problem is thinking you have a row vector when in fact you have a column vector. Check by using `size(X)`; here that gives `ans=1,3` which tells you that `X` is a 1×3 (row) vector.

- These functions operate on each element of an array separately, giving another array: `sin,cos,tan,exp,log,log10,sqrt` `cos([0:3]*pi)` gives `ans=1 -1 1 -1`
- To compute n^2 for $n = 0, 1 \ldots 5$, use `[0:5].^2` gives `ans=0 1 4 9 16 25`
- To compute 2^n for $n = 0, 1 \ldots 5$, use `2.^[0:5]` gives `ans=1 2 4 8 16 32`
- "." is needed for point-by-point operation.

Some other array operations:

- `A=[1 2 3;4 5 6];(A(:))'` Stacks A by columns into a column vector and transposes the result to a row vector. Here, this gives `ans=1 4 2 5 3 6`
- `reshape(A(:),2,3)` Unstacks the column vector to a 2×3 array which, in this case, is the original array `A`.
- `X=[1 4 1 5 9 2 6 5];C=X(2:8)-X(1:7)` Takes differences of successive values of `X`. Here, this gives `C=3 -3 4 4 -7 4 -1`
- `D=[1 2 3]; E=[4 5 6]; F=[D E]` This *concatenates* the vectors `D` and `E` (i.e., it appends `E` after `D` to get vector `F`) Here, this gives `F=1 2 3 4 5 6`
- `I=find(A>2)` stores in `I` locations (indices) of elements of vector `A` that exceed 2. `find([3 1 4 1 5]<2)` gives `ans=2 4`
- `A(A>2)=0` sets to 0 all values of elements of vector `A` exceeding 2. `A=[3 1 4 1 5]; A(A<2)=0` gives `A=3 0 4 0 5`

Matlab indexing of arrays starts with 1, while signals and systems indexing starts with 0. For example, the DFT is defined using index n=0, 1 ... N–1, for k=0, 1 ... N–1. In particular, `fft(X)` computes

```
fft(X)=X*exp(-j*2*pi*[0:N-1]'*[0:N-1]/N);
```

13.2.4 Solving Systems of Equations

To solve the linear system of equations

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} = \begin{bmatrix}17\\39\end{bmatrix}$$

using `A=[1 2;3 4];Y=[17;39];X=A\Y;X'` gives `ans=5.000 6.000`, which is the solution $[x\ y]'$.

To solve the complex system of equations

$$\begin{bmatrix}1+2j & 3+4j\\5+6j & 7+8j\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} = \begin{bmatrix}16+32j\\48+64j\end{bmatrix}$$

`[1+2j 3+4j;5+6j 7+8j]\[16+32j;48+64j]` gives `ans=`$\begin{smallmatrix}2-2i\\6+2i\end{smallmatrix}$, which is the solution.

These systems can also be solved using `inv(A)*Y`, but this is a bad idea, since computing the matrix inverse of `A` takes much more computation than just solving the system of equations. Computing a large matrix inverse can lead to numerical difficulties.

13.3 Plotting

13.3.1 Basic Plotting

To plot a function $x(t)$ for $a \le t \le b$:

- Generate, say, 100 values of t in $a \le t \le b$ using `T=linspace(a,b,100);`
- Generate and store 100 values of $x(t)$ in `X`
- Plot each computed value of `X` against its corresponding value of `T` using `plot(T,X)`

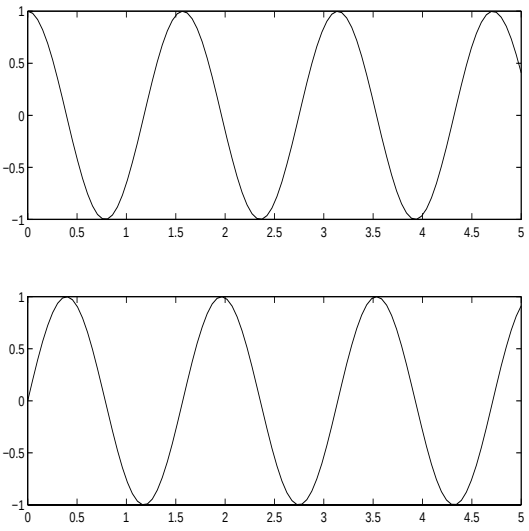
- If you are making several different plots, put them all on one page using `subplot`. `subplot(324),plot(T,X)` divides a figure into a 3-by-2 array of plots, and puts the `X` vs. `T` plot into the 4th place in the array (the middle of the right-most column).

Print out the current figure (the one in the foreground; click on a figure to bring it to the foreground) by typing `print` Print the current figure to a encapsulated postscript file myname.eps by typing `print -deps2 myname.eps`. Type `help print` for a list of printing options for your computer. For example, use `-depsc2` to save a figure in color.

To make separate plots of $\cos(4t)$ and $\sin(4t)$ for $0 \leq t \leq 5$ in a single figure, use the following:

```
T=linspace(0,5,100);X=cos(4*T);Y=sin(4*T);
subplot(211),plot(T,X)
subplot(212),plot(T,Y)
```

These commands produce the following figure:



The default is that `plot(X,Y)` plots each of the 100 ordered pairs `(X(I),Y(I))` for `I`=1...100, and connects the points with straight lines. If there are only a few data points to be plotted, they should be plotted as individual ordered pairs, not connected by lines. This can be done using `plot(X,Y,'o')`

13.3.2 Plotting Problems

Common problems encountered using `plot`:
`T` and `X` must have the same lengths; and
Neither `T` nor `X` should be complex; use `plot(T,real(X))` if necessary.

The above `linspace` command generates 100 equally-spaced numbers between a and b, *including a and b*. This is *not* the same as sampling $x(t)$ with a sampling interval of $\frac{b-a}{100}$. To see why:

- `linspace(0,1,10)` gives 10 numbers between 0 and 1 inclusive, spaced by 0.111
- `[0:.1:1]` gives 11 numbers spaced by 0.1

Try the following yourself on Matlab:

- `T=[0:10];X=3*cos(T);plot(T,X)` This should be a very jagged-looking plot, since it is only sampled at 11 integers, and the samples are connected by lines.
- `T=[0:0.1:10];X=3*cos(T);plot(T,X)` This should be a much smoother plot, since there are now 101 (not 100) samples.
- `T=[1:4000];X=cos(2*pi*440*T/8192);` `sound(X,8192)` This is musical note "A." `sound(X,Fs)` plays `X` as sound, at a sampling rate of `Fs` samples/second.
- `plot(X)`. This should be a blue smear! It's about 200 cycles squished together.
- `plot(X(1:100))` This "zooms in" on the first 100 samples of X to see the sinusoid.

13.3.3 More Advanced Plotting

Plots should be labelled and annotated:

- `title('Myplot')` adds the title "Myplot"
- `xlabel('t')` labels the x-axis with "t"
- `ylabel('x')` labels the y-axis with "x"
- `$\omega$` produces ω in `title,xlabel` and `ylabel` Similarly for other Greek letters. Note ', not ', should be used everywhere.

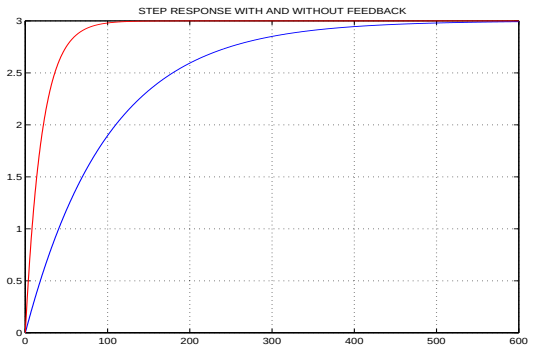
- `axis tight` contracts the plot borders to the limits of the plot itself
- `axis([a b c d])` changes the horizontal axis limits to $a \leq x \leq b$ and the vertical axis limits to $c \leq y \leq d$.
- `grid on` adds grid lines to the plot
- `plot(T,X,'g',T,Y,'r')` plots on the same plot (`T,X,Y` must all have the same lengths) `X` vs. `T` in green and `Y` vs. `T` in red.

13.3.4 Plotting Examples

A good way to learn how to plot is to study specific examples. Four specific illustrative examples follow.

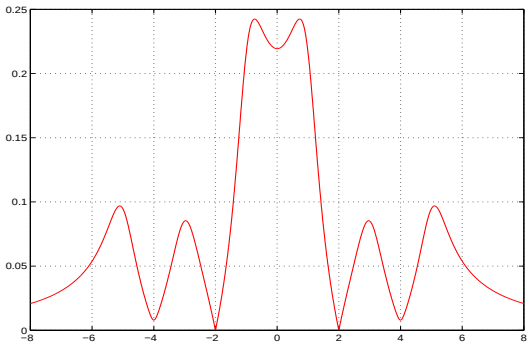
This example shows how to plot two different functions in a single plot, using different colors, and insert a title. Matlab code for this plot:

```
clear;T=linspace(0,600,1000);
A=0.01;K=0.04;B=A+K;
SA=3*(1-exp(-A*T));
SB=3*(1-exp(-B*T));
plot(T,SA,'b',T,SB,'r')
title('STEP RESPONSE WITH
AND WITHOUT FEEDBACK')
grid on,print -depsc2 m1.eps
```



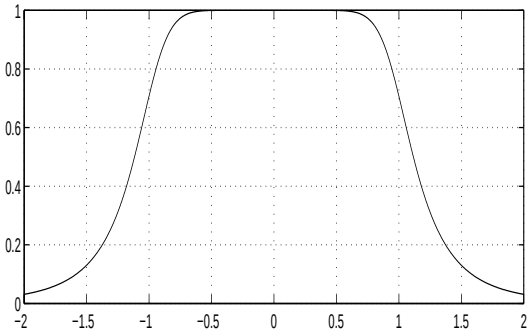
This example demonstrates `./` and `.*` Matlab code used for this example:

```
clear;W=linspace(-8,8,1000);V=j*W;
N=(V+2j).*(V-2j).*(V-.1+4j).*(V-.1-4j);
D1=(V+.5+1j).*(V+.5-1j).*(V+.5+3j);
D2=(V+.5-3j).*(V+.5+5j).*(V+.5-5j);
plot(W,abs(N./D1./D2),'r'),grid on
```



This example shows how to use a loop: `H=H./(V-P(I));` is executed for `I=1,2,3,4,5`. Matlab code for this example:

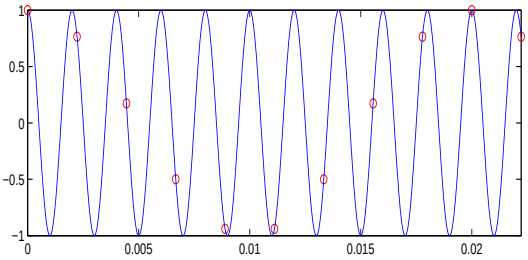
```
clear;W=linspace(-2,2,1000);V=j*W;
P=exp(j*2*pi*[3:7]/10);H=ones(1,1000);
for I=1:5;H=H./(V-P(I));end
subplot(211),plot(W,abs(H)),grid on
```



This example shows how to use `hold` to superpose two plots, how to plot individual points, use `subplot` to change the aspect ratio of a figure, and `axis tight` to tighten it. It also uses `[a:0.001:b]`, not `linspace(a,b,1000)`, to sample every 0.001. Matlab code for this example:

```
T1=[0:1/45000:1/45];
X1=cos(2*pi*500*T1);
T2=[0:1/450:1/45];
```

```
X2=cos(2*pi*500*T2);
subplot(211),plot(T2,X2,'or'),hold,
subplot(211),plot(T1,X1),axis tight
```



13.4 Partial Fractions

13.4.1 Rectangular-to-Polar Complex Conversion:

If a Matlab result is a complex number, then it is presented in its rectangular form `a+bj`. Matlab recognizes both `i` and `j` as $\sqrt{-1}$, so that complex numbers can be entered as `3+2j`.

To convert a complex number `X` to polar form, use `abs(X),angle(X)` to get its magnitude and phase (in radians), respectively. To get its phase in degrees, use `angle(X)*180/pi`

Note `atan(imag(X)/real(X))` will **not** give the correct phase, since this formula is only valid if the real part is positive. `angle` corrects this.

The real and imaginary parts of `X` are found using `real(X)` and `imag(X)`, respectively.

13.4.2 Polynomial Zeros:

To compute the zeros of a polynomial, enter its coefficients as a *row* vector `P` and use `R=roots(P)`. For example, to find the zeros of $3x^3$–$21x$+18 (roots of $3x^3$–$21x$+18=0) use `P=[3 0 -21 18];R=roots(P);R'`. `ans= -3.0000 2.0000 1.0000`, which are the roots.

To find the monic (leading coefficient one) polynomial having a given set of numbers for zeros, enter the numbers as a *column* vector `R` and use `P=poly(R)`. For example, to find the polynomial having $\{1, 3, 5\}$ as its zeros: `R=[1;3;5];P=poly(R)`, giving `P=1 -9 23 -15`. The polynomial is therefore x^3–$9x^2$+$23x$–15.

Note that polynomial are stored as row vectors, and roots are stored as column vectors.

Pole-zero diagrams are made using `zplane`. To produce the pole-zero diagram of $H(z)$=$\frac{z^2+3z+2}{z^2+5z+6}$, type `zplane([1 3 2],[1 5 6])`.
The unit circle $|z|$=1 is also plotted, as a dotted line.

13.4.3 Partial Fraction Expansions:

Partial fraction expansions are a vital part of discrete-time signal processing, since they are used for inverse z-transforms. and their computation is onerous. Matlab computes partial fraction expansions using `residue`. Specifically,

$$H(s) = \frac{b_0 s^M + b_1 s^{M-1} + \ldots + b_M}{a_0 s^N + a_1 s^{N-1} + \ldots + a_N}$$

has the partial fraction expansion (if $M \leq N$)

$$H(s) = K + \frac{R_1}{s-p_1} + \ldots + \frac{R_N}{s-p_N}$$

The poles $\{p_i\}$ and residues $\{R_i\}$ can be computed from coefficients $\{a_i\}$ and $\{b_i\}$ using

`B=`$[b_0\ b_1 \ldots b_M]$`;A=`$[a_0\ a_1 \ldots a_N]$
`[R P]=residue(B,A);[R P]`

The residues $\{R_i\}$ are given in column vector `R`, and poles $\{p_i\}$ are given in column vector `P`.

To compute the partial fraction expansion of $H(s) = \frac{3s+6}{s^2+5s+4}$, use the command

```
[R P]=residue([3 6],[1 5 4]);[R P]
```

This gives $\begin{bmatrix} 2 & -4 \\ 1 & -1 \end{bmatrix}$, so `R`=$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ and `P`=$\begin{bmatrix} -4 \\ -1 \end{bmatrix}$ from which we read off $H(s)$=$\frac{2}{s+4}$+$\frac{1}{s+1}$.

In practice, the poles and residues both often occur in complex conjugate pairs. Then use

$$Re^{pt} + R^* e^{p^* t} = 2|R|e^{at}\cos(\omega t + \theta),$$

R=$|R|e^{j\theta}$ and p=a+$j\omega$, to simplify the result.

To compute the partial fraction expansion of $H(s) = \frac{s+7}{s^2+8s+25}$, use the command

```
[R P]=residue([1 7],[1 8 25]);[R P]
```

This gives $\begin{bmatrix} 0.5000-0.5000i & -4.000+3.000i \\ 0.5000+0.5000i & -4.000-3.000i \end{bmatrix}$ from which we have $H(s)=\frac{0.5-j0.5}{s+4-j3}+\frac{0.5+j0.5}{s+4+j3}$

which has the inverse Laplace transform

$h(t)=(0.5–\text{j}0.5)e^{(-4+j3)t}+(0.5+\text{j}0.5)e^{(-4-j3)t}$

From `abs(0.5-0.5j),angle(0.5-0.5j)`,

$h(t)=2\frac{\sqrt{2}}{2}e^{-4t}\cos(3t–\frac{\pi}{4})=\sqrt{2}e^{-4t}\cos(3t–\frac{\pi}{4})$.

Both $h(t)$ expressions are valid for $t>0$.

If $H(s)$ is proper but not strictly proper, the constant K is nonzero. It is computed using

```
[R P K]=residue(B,A);[R P],K
```

since `K` has size different from `R` and `P`.

To find the partial fraction expansion of $H(s)=\frac{s^2+8s+9}{s^2+3s+2}$, use the command

```
[R P K]=residue([1 8 9],[1 3 2]);[R P]
```

`K` gives $\begin{bmatrix} 3 & -2 \\ 2 & -1 \end{bmatrix}$, `K=1` so `R`$=\begin{bmatrix} 3 \\ 2 \end{bmatrix}$, `P`$=\begin{bmatrix} -2 \\ -1 \end{bmatrix}$ from which we read off $H(s)=1+\frac{3}{s+2}+\frac{2}{s+1}$.

Double poles are handled as follows:
To find the partial fraction expansion of $H(s)=\frac{8s^2+33s+30}{s^3+5s^2+8s+4}$, use the command

```
[R P]=residue([8 33 30],[1 5 8 4]);
```

`[R P]` gives $\begin{bmatrix} 3 & -2 \\ 4 & -2 \\ 5 & -1 \end{bmatrix}$, so `R`$=\begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$, `P`$=\begin{bmatrix} -2 \\ -2 \\ -1 \end{bmatrix}$
We then read off $H(s)=\frac{3}{s+2}+\frac{4}{(s+2)^2}+\frac{5}{s+1}$.

In practice, we are interested not in an analytic expression for $h(t)$, but in computing $h(t)$ sampled every Ts seconds. These samples can be computed directly from `R` and `P`, for $0\le t\le T$:

```
t=[0:Ts:T];H=real(R.'*exp(P*t));
```

Since `R` and `P` are column vectors, and `t` is a row vector, `H` is the inner products of `R` with each column of the array `exp(P*t)`. `R.'` transposes `R` without also taking complex conjugates of its elements. `real` is necessary since roundoff error creates a tiny (incorrect) imaginary part in `H`.

13.4.4 Frequency Response:

`polyval(P,W)` evaluates the polynomial whose coefficents are stored in row vector `P` at the elements of vector `W`. For example, to evaluate the polynomial $x^2–3x+2$ at $x=4$, `polyval([1 -3 2],4)` gives `ans=6`

The continuous-time **frequency response** of

$$H(s)=\frac{b_0s^M+b_1s^{M-1}+\ldots+b_M}{a_0s^N+a_1s^{N-1}+\ldots+a_N}$$

can be plotted for $0\le\omega\le W$ using

```
B=[b_0 b_1 ... b_M];A=[a_0 a_1 ... a_N]
w=linspace(0,W,1000);
H=polyval(B,j*w)./polyval(A,j*w);
subplot(211),plot(w,abs(H))
```

13.4.5 Discrete-Time Commands

- `stem(X)` produces a stem plot of `X`
- `conv(X,Y)` convolves `X` and `Y`
- `fft(X,N)` computes the N-point DFT of `X`
- `ifft(F)` computes the inverse DFT of `F`.
  Due to roundoff error, use `real(ifft(F))`.
- `sinc(X)` compute $\frac{\sin(\pi x)}{\pi x}$ for each element.

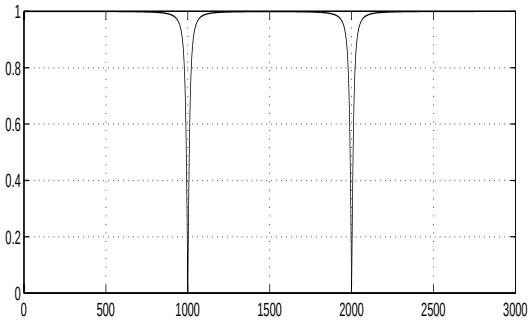
13.4.6 Continuous-time Comb Filter

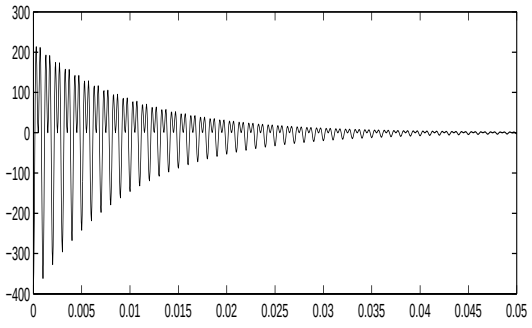
The following two plots are the impulse and magnitude frequency responses of a comb filter that eliminates 1-kHz and 2-kHz sinusoids, using poles with a real part of –100.

Matlab code used for these plots:

```
F=linspace(0,3000,100000);
W=j*2*pi*F;A=100;
%Compute frequency response:
Z=j*2*pi*1000*[-2 -1 1 2];
N=poly(Z);D=poly(Z-A);
```

```
FH=polyval(N,W)./polyval(D,W);
subplot(211),plot(F,abs(FH))
%Compute impulse response:
T=linspace(0,0.05,10000);
[R P K]=residue(N,D);
H=real(R.'*exp(P*T));
subplot(212),plot(T,H)
```





13.5 Image Processing Toolbox

To read in an image into Matlab, use

```
A=imread('mandrill.tif','tif');
```

Most image formats can be handled.

To display a grayscale (black and white) image:

```
imagesc(X),colormap(gray),axis off
```

13.6 Symbolic Toolbox

Matlab's Symbolic Toolbox is a great help in eliminating tedious algebra. We give three examples.

To check the complex partial fraction expansion example given above, we can use

```
syms t;
X=laplace(sqrt(2)*exp(-4*t)*cos(3*t-pi/4));
simplify(X) This gives the output
```

```
(s+7)/(s^2+8*s+25)
```

To compute the bilinear transform in the example at the end of the IIR filter design chapter,

```
syms z;s=4*(z-1)/(z+1);
H=(s+0.1)/((s+0.1)*(s+0.1)+16);
simplify(H) This gives the output
```

```
10*(z+1)*(41*z-39)/(3281*z*z+2*z+3121)
```

which agrees with the output of `bilinear`.

To compute the inverse DTFT of $j\omega$ for the differentiator, we simplify the integral as much as possible. Noting $j\omega$ is imaginary and odd, and $e^{j\omega n}$=cos(ωn)+j sin(ωn), the cosine integral is zero by symmetry. Using j^2=–1, this leaves

$$h[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} j\omega e^{j\omega n} d\omega = \frac{-1}{\pi}\int_{0}^{\pi} \omega \sin(\omega n) d\omega. \tag{13.1}$$

This is computed in Matlab using

```
-int('w*sin(w*n)',0,pi)/pi
```

which gives the output

```
-(sin(pi*n)-pi*n*cos(pi*n))/(pi*n*n)
```

which is indeed $\frac{(-1)^n}{n}$ for $n \neq 0$
(Matlab can't figure out that sin(πn)=0).

Some symbolic transforms and functions:

```
laplace,ilaplace,fourier,heaviside
ifourier,ztrans,iztrans,impulse
```